1-1-1976

Scheduling the central processing unit in a product information system.

F. Paul Fuhs
University of Massachusetts Amherst

SCHEDULING THE CENTRAL PROCESSING UNIT IN A PRODUCT INFORMATION SYSTEM

A Dissertation Presented

by

F. PAUL FUHS

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August, 1976

Business Administration

SCHEDULING THE CENTRAL PROCESSING UNIT IN A

PRODUCT INFORMATION SYSTEM

A Dissertation

By

F. PAUL FUHS

Approved as to style and content by:

Van Court Hare Jr
Van Court Hare, Jr., Chairman, Dept. of Management, SBA

Joseph L. Sardinas Jr
Joseph L. Sardinas, Jr., Member, Dept. of Accounting, SBA

Hugh J. Miser
Hugh J. Miser, Member, Dept. of Industrial Engineering
& Operations Research

Jack Wolf, Associate Dean
School of Business Administration

ACKNOWLEDGEMENT

I would like to express my gratitude to Professor Van Court Hare, Jr., my dissertation committee chairman and Ph.D. program advisor, who has followed the gestation of the ideas encompassed in this work for many years. I would also like to thank the other members of my committee, Professor Hugh J. Miser and Professor Joseph L. Sardinas, Jr., for the time and help that they have given to me at various stages of this work. My appreciation is also extended to Professor William Margolis of Virginia Commonwealth University for his suggestions on Appendix 1. I wish to thank Mrs. Jane Williams for her care and expertise in typing this dissertation.

My greatest sense of appreciation is to my wife, Kathleen. Her never ending help and continual patience contributed most to the completion of this work. To her I dedicate this dissertation.

To Kathleen

The Perfect Scheduler

ABSTRACT

SCHEDULING THE CENTRAL PROCESSING UNIT IN A PRODUCT INFORMATION SYSTEM

August, 1976

F. Paul Fuhs, B.S., Spring Hill College;
M.S., Purdue University; B.D., Boston College;
PhD., University of Massachusetts

Directed by: Dr. Van Court Hare, Jr.

Within the near future we can expect the application of distributed data bases to a novel customer service, the Product Information System. In such a system the customer, either directly or through a firm's representative, queries the corporate data base containing information on product descriptions, availability, and prices. This study examines the relative effect of three CPU scheduling disciplines on system response time and lost potential sales under various conditions of CPU utilizations, main memory capacity, and paging loads on secondary storage devices. These three factors form a three dimensional system space in which the CPU scheduling disciplines can be studied. A model of a Product Information System is developed and a computer simulation of this model is used as an experimental tool to study the behavior of the three CPU scheduling disciplines.

The first discipline considered is a modified FIFO discipline in which the CPU is fed by two queues: a pre-emptive queue composed of operating system routines and a non-pre-emptive queue composed primarily of data base management system routines. The second discipline is a variance reduction algorithm which compares a query's progress through the system at selected milestones with expected time values and at each milestone adjusts a query's priority accordingly. The *a priori* rationale for the use of discipline two is the uniformity of query response. The third discipline monitors the status of the queues feeding the secondary storage devices. If these queues are not saturated, higher priority is given to the queries that are ready to generate paging activity. If the queues are deemed saturated, discipline three reverts by default to discipline two.

Our data supports the statement that in the future development of Product Information Systems serious consideration must be given to the CPU scheduling discipline employed in such data base systems. Specifically, we demonstrate that within the range of highly probable system parameter settings the relative behavior of the three disciplines varies. Of these three disciplines, none can be considered best under all Product Information System conditions, although discipline three excels over a wider range than the others. When main memory is sufficient to contain all necessary query processing routines and the load on the secondary storage subsystem is light, the disciplines manifest a statistically significant ranking order at a CPU utilization above 0.75. If the average system response time is the performance variable, the rankings from best to worst are disciplines 3, 2, 1. Given main memory sufficiency under a high CPU utilization the relative advantage of both discipline 2 and 3 over discipline 1 rapidly deteriorates as the load on the secondary storage subsystem is increased until under a high loading situation discipline 1 emerges as the best discipline to employ. Under the extreme condition of a CPU utilization of 1.0 the phenomenon of routine entrapment is demonstrated in which some routines which are processing queries become locked in the CPU queue due to newly arriving routines having higher priorities.

We demonstrate that when main memory capacity is not sufficient and the paging of program pages is required, that is, in a virtual storage environment, the relative ranking of the three scheduling disciplines can be affected. Discipline three is shown to be superior over a wider range of CPU utilizations and secondary storage device loadings, given demand program paging, than when the system is run under memory sufficiency. This is due to the fact that in a virtual storage environment the number of times priority setting occurs per query is increased. This results in a greater priority setting sensitivity for discipline 3.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

I N T R O D U C T I O N

Statement of Thesis

In developing and maintaining a Product Information System the choice of a Central Processing Unit (CPU) scheduling algorithm by the Data Base Administrator is critical to good data base performance.

Product Information Systems

The evolution of computerized information systems in the business sector of our economy began in the 1950's with the focus upon such internal operations of the firm as payroll, accounts receivable, accounts payable, and inventory control. These systems were then gradually coalesced into management information systems. During the 1960's the information systems of businesses were still primarily directed towards the internal operations of the firm with a stress on the centralization of information as more and more application areas were automated. For the most part the customer interfaced with the firm's information system only indirectly. A customer was primarily an object about which information was to be gathered (e.g., marketing surveys, sales forecasts) or for whom information was to be collected. In both cases the consumer was a passive part of the information system. The target or user of the information system was still the firm itself.

At present there are definite signs that the customer will not only be drawn more closely into the orbit of business information systems but also that the consumer as user will become a more integral part of the firm's total information system. In the near future customers of products and services will become users of information systems supplied by businesses with the express purpose of acquainting customers with their products. Advertising will have reached a new dimension. The customer will be given the capability to query selected business data bases containing product descriptions, availability, and cost. We will refer to these information systems as Product Information Systems.

In order for future Product Information Systems to be realizable, four factors must be considered. First, there must be a market for such a service. Second, this market must be perceived by firms. Third, such systems must be technologically feasible. Fourth, such systems must be cost effective, i.e., profitable for the implementors.

The market for Product Information Systems is still in the embryonic stage. Customers are becoming more acclimated to carrying out business transactions in a computer environment. People's fear of interacting with data communication devices is waning. In purchasing goods and services, customers already interface with computerized airplane and hotel reservation systems, automated credit checks, and point of sale transaction systems (POS). Soon electronic funds transfer systems (EFTS) may also be widely accepted, provided appropriate audit trails and verification are developed. The speed and the accuracy of the handling of current computerized transactions are recognized as beneficial by both businesses and customers alike.

A Product Information System would play a double role, profitable both to the firm and the customer. First, it would be an implicit advertising medium. Secondly, it would provide the customer with an easy medium for comparative shopping. Physical browsing and the desire to touch merchandise will always be a part of our purchasing habits as individual customers. Yet, as product areas diversify there arises the need for customers to find specifically desired products and to obtain information about the existence of sets of previously unknown products which may satisfy felt needs. The lack of a direct cost for this service will be appealing to the customer. This is especially true when coupled with the fact that comparative shopping would be at the discretion of the customer as to the time of day and the extent of query. For the customer such shopping could be cost effective in that more products and firms could be canvassed in a given time interval without the necessity of leaving one's own premises.

One can discuss the technological feasibility of Product Information Systems from two viewpoints. First, would Product Information Systems be a gradual evolutionary development of currently existing technology or should they be considered a radical technological quantum jump? Secondly, specifically what hardware and software configurations would be necessary to implement a Product Information System?

Future Product Information Systems can be looked at as logical extensions of existing order processing. We have already reached the stage where customer orders are captured and entered into the seller's computer systems directly from the customer's premises on both the wholesale and retail level, even without human assistance on the part of the seller (Resnick, 1974; Computerworld, 1975c).

While a customer goes through three phases in acquiring goods, selection of the goods, placement of an order, and payment for the goods, the first two phases are most clearly interconnected. Frequently phases one and two can be performed in rapid temporal succession. Therefore the personnel, the hardware, and some of the software dedicated to order processing could be extended to the function of providing a Product Information System for the customer.

Large volume catalog sales corporations will probably be among the first to develop such systems, since they already have order processing networks. The Product Information System is a logical extension of the printed catalog and the microfiche parts listing systems. Catalogs because of printing and mailing costs are necessarily restricted in their informational content. They are also restrictive in that they do not portray very current information and, when they do, it is usually through cumbersome supplements. In the near future we shall see Product Information Systems set up whereby a customer can call a company representative by telephone to present a query concerning a product or set of products. The representative in turn would communicate with the firm's computer system through a CRT in some form of stylized language instead of manually searching through a catalog or parts list. This informational triad, the inquirer, the representative, and the computer system are already in wide use. From the users' point of view the response would be in real time.

Real time systems differ from time sharing systems. In real time systems all programs have been debugged before the system becomes operational. Main memory requirements are known. The terminal operator may request data processing tasks from a preselected menu (Stimler, 1965).

The hardware and software needed to support initial developmental stages of Product Information Systems are currently available. CRTs, modems, lines, and channel types abound for communications. In addition to communication facilities, the Product Information System most likely would include hierarchical secondary storage devices and a virtual storage operating system. Hierarchical secondary storage devices would be appropriate, since the informational demand on products would not be expected to be uniform. A virtual storage operating system would be beneficial in memory management and computer resource allocation.

The communication link between the user and the system could be shortened with even today's technology so as to exclude most of the human data entry representatives of the firm. This could be accomplished through the use of TOUCH TONE phones in a simple standardized query language with output coming to home T.V. sets. Some representatives would, however, be needed to help prompt customers in the use of the system.

The critical area in the adoption of a Product Information System is cost. Knight (1966) warns that:

> Special consideration has to be given to the fact that there are definite limits to the maximum computing power that can be obtained at any one time. As the bounds of technological knowledge are reached, additional computing power is purchased at a very high price.

Yet, there are a number of factors which point to reduced developmental hardware costs over the next decade. Grosh's law that computing technology constructs machines four times as powerful at twice the cost still appears to be verified. Large scale integrated circuits have led to the realization of the CPU as a single chip. The cost of electronic circuits has dropped by more than a power of ten during the 1960's and may drop by another power of ten in the next decade (Withington, 1969). New technologies like magnetic bubble memories and charge-coupled devices will produce faster processors. The trend in processors is already toward parallel operations (vector and array processing), distributed computation, micro-programming and minicomputer controllers of disk utilization (Auerbach, 1974a).

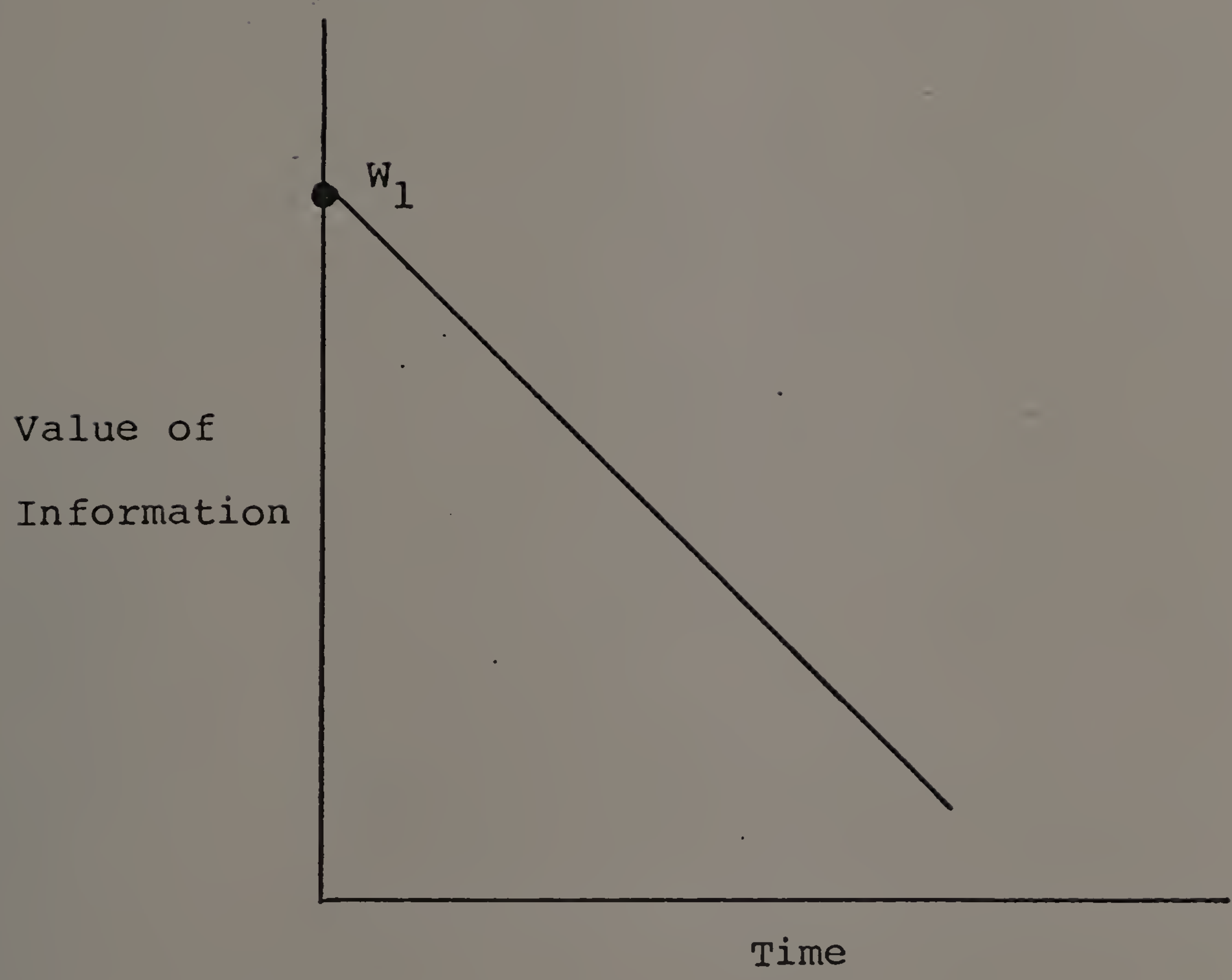
The cost of time sharing systems has also dropped and in some systems the operational cost is already approaching $2.00 per connect hour (Computerworld, 1975a). The use of distributed information centers using minicomputers is another implementation alternative for reducing costs in Product Information Systems. This would reduce line charges. Another possible reduction in line charges will result from using satellite communication networks, coupled on one end to concentrators linked to CRTs and on the other end to mainframes. Communication satellites are already competitive with land lines over very long distances, and through larger capacities and increased power they will begin to compete with land lines over shorter distances (Martin, 1969). COMSAT is but the beginning of private industry's involvement with communication satellites.

As for secondary storage devices the specification for IBM's Future Systems (FS) included the development of disk, drum, and tape subsystems (Lundell, 1973b). Arthur D. Little, in projecting price/performance over the next ten years points out that systems of the 1985's will still use tape, disk drives, and impact printers (Farmer, 1974b).

In projecting the impact of software costs on new information systems, Arthur D. Little is less optimistic. Although highly flexible, fully automated data management software will be integrated with the hardware, the software costs still consume a large share of the total information system cost. In fact, software system development costs have come to equal the hardware development costs for even our third generation machines (Opler, 1967). Yet, software reproduction costs for use in multiple systems are minimal when compared with reproducing hardware components. Also specialized tools for software engineering are being used more often, like high level language preprocessors. Even operating systems are being developed in high level languages. For example, the MULTICS VS operating system was developed primarily in PL/1 (Organick, 1972).

While hardware costs are continually decreasing and the increasing rate of software costs is dampened by Data Base Management Systems (DBMS), preprocessors, and standardizations, other problems exist in the development of Product Information Systems, many of which are historically based. The 1960's saw a rash of developmental effort into Management Information Systems with expenditures in the billions of dollars. From this work came the sobering realization that a truly effective MIS is indeed a difficult task. Failures were numerous and were attributable in the design phase to unrealistic goals, poor cost estimation, and principally to a lack of participation by top management (Voich, Mottice, and Shrode, 1975). This lack of involvement was compounded by gross underestimation of the resources, including time, that are necessary for information system development. The failure of many MIS projects was also attributable to the systems analysts themselves. For many information systems that subsequently failed even the failure itself was slow in being recognized. This was due in part to the fact that before and after implementation systems analysts did not expend enough effort in relating the value of the information generated by their systems to the decisional responsibilities of the managers who used these systems.

The value of information to either a manager or a customer is usually a function of time. Figure 1 represents



Value of Information as a Function of Time

Figure 1

such a function. As the time interval between the query and the answer increases, the value of the information for the decision maker decreases. Considered in its simplest form as a linear equation with negative slope,

$$V = w_1 - w_2 t$$

where V is the value of information to the user,

w_1 is the maximum value that can be predicated upon the information before any decay attributable to time,

w_2 is the slope of the line and

t is the time interval between the query and the answer.

From this figure we can see that the value of information can be increased either by shortening the time interval between query and answer or by shifting the entire line upward through the presentation to the user of better quality information. In the former case the upper limit is at w_1. Performance evaluation of information systems during the 1960's concentrated on shortening the response time, and too frequently the quality of the information presented to the user was overlooked. This quality is as important to a Product Information System as a Management Information System. During this era performance evaluation of information systems in the area of data base tuning was also in its infancy and few were knowledgeable in the use of hardware and software monitors.

During the late 1960's a more realistic approach to the development of information systems began. Goals were trimmed. The users' informational needs were more clearly sought and delineated and data base systems were developed for particular sets of users. Generalized data base management systems were developed and more widely adopted. These software systems can be tailored for both managers and customers.

Concurrent with the development of data base systems was the development of linguistic interfaces between man and machine; computer languages and syntactic and semantic analyzers. There were dashed expectations also in the area of linguistic interfaces, as evidenced by the failure of automating Russian to English translations. There are obvious implications to these failures. We are more ready to admit that in man's communication with the computer, man must be willing to forego some of the richness of his human language. At least for the near future we must settle for a somewhat stylized form of man-machine communication in both MIS and Product Information Systems. This applies to query languages as well as to programming languages.

Problems in the design and implementation of information systems are not restricted to managers, systems analysts, and performance measurement tools. In the research and development of information systems a dichotomy long ago was established and has persisted to the present day. On the one hand arose groups of people, who may be classified as being in the area of data bases. On the other hand we find groups of people interested primarily in operating systems. The problem exists that these two types of groups rarely communicate. The groups interested in data bases concern themselves with data base structure, accessing methods, selection of secondary storage devices, and privacy. For these groups the operating system is a given black box, in the systems analysis sense of the term. The groups of people interested in operating systems are responsible for the development of the system software and sometimes the hardware which supports the multiprogramming environment of a computer system. For these groups the data base is considered a black box and what is important is memory and processor management, accessing rates, device allocation, channel and device utilizations, and security.

Before the advent of Data Base Management Systems and as long as DBMS could be viewed as distinct from operating systems, one could continue to compartmentalize these software areas and the technical personnel which support them. However, we now see that more information systems are becoming data base oriented and many of these data base systems are becoming the prime user of computer resources. Dedicated data base systems are no longer an oddity to be contended with as just another set of programs to be processed by the hardware and operating system.

The CODASYL data base task group in 1971 suggested that the functions of organizing and maintaining a data base should be given to the Data Base Administrator. This administrator is considered to be either an individual or more likely a group of individuals acting as a committee. Among the responsibilities of the Data Base Administrator is the determination of the content, size, structure, and authorized use of the data base. Newpeck (1973) outlines a fairly comprehensive description of the Data Base Administrator's responsibilities, based in part on the CODASYL recommendations. Therefore the part of the Product Information System which relates to the development, functioning, and maintenance of the data base is to be entrusted to a Data Base Administrator.

There is today a greater necessity for the Data Base Administrator to critically evaluate some functions once thought of as pertaining solely to the operating system. Among these reasons is the fact that already DBMS are being incorporated directly into operating systems (Moreira, Pinheiro, and D'Elia, 1974). Canning (1972) points out that among the operational activities of the Data Base Administrator he should exert some control over computer scheduling so as to provide priority use of the data base. The algorithm which allocates CPU time among contending processes is one of the functions of the operating system that the Data Base Administrator must now begin to consider among his other responsibilities.

The CODASYL report, while initiating guidelines for the structure and operation of future data base systems, provided no assistance to the Data Base Administrator in the way of attacking the problems of CPU algorithm selection. While much has been written on CPU scheduling algorithms, most of this work has concentrated on batch or general time sharing systems, not data base systems. In addition the modeling of CPU scheduling performance has been primarily analytical with the necessary accompanying restrictive assumptions. Frequently in the literature on comparative CPU scheduling algorithms the overhead generated by the scheduling routines themselves is neglected in both analytic and simulation models. Many queueing models of CPU activity assume Poisson arrivals because of their mathematical tractability. Yet, for many systems this assumption remains unvalidated. Many analytic queueing models applied to CPU scheduling algorithms assume an infinite arrival source and disregard the cyclic nature of many processing operations in information systems.

In summary, the Weltanschauung that we propose for the Data Base Administrator of future Product Information

Systems is one which encompasses not only the logical and physical structure of the data base itself, but also the CPU scheduling algorithm, which is one of the prime influences on data base performance among the routines comprising the operating system. This does not mean to imply that other areas of data base administration are to be deemphasized. While traditionally the operating system has been "off limits" to the Data Base Administrator, this restriction should no longer be perpetuated, especially as new applications of information systems are developed.

CHAPTER I

CPU SCHEDULING AND DATA BASE SYSTEM OPERATIONS

The Data Base Administrator's ability to maintain an adequate data base has been affected during the last ten years by two antithetic trends. The first trend deals with the support structure of a data base. Operating Systems coupled with sophisticated Data Base Management Systems are becoming more complex and as a result it has become more difficult for both the online and offline analysis of system performance data. On the other hand, the Data Base Administrator is beginning to be aided by a trend towards the increased number and usage of software and hardware performance monitoring devices.

The Data Base Administrator is beginning therefore to have at his disposal tools for the collecting of performance data on data base systems. Minicomputers and other stand alone devices are appearing on the market with appropriate transducers to measure computer system performance. In the near future the increased diversity of mainframes and peripheral equipment will begin to exert pressure on the manufacturers to include performance monitors as standard features of their basic computer architecture. Primitive software performance routines are even now an integral part of operating systems. Witness IBM's incorporation of the Measurement Feature (MF-1) in their Virtual Machine System (VMS 2). This can measure the resource utilization within the system at all times. Yet, if performance monitors are implemented only in software, two disadvantages occur. First, they require additional storage to operate. Second, they are victims of the Heisenberg Indeterminacy Principle. They interfere with the environment which they are supposed to be measuring. In addition, software monitors can give inaccurate measurements. At the Virginia Commonwealth University Computer Center research on the performance of CPU utilization by both a software monitor (IBM's SMF) and a hardware monitor (1155B Tesdata Measurement System) has shown a wide variation between the software and the hardware measurements over the same time span. This discrepancy was truly significant in that, while the SMF data reported a CPU utilization of 43%, the hardware monitor registered over 90%. Tavitian (1975) reports similar problems with present day software monitors.

In relation to the subject matter of this paper we can expect in the future the development of hardware monitors which will gather performance data on a continuing basis and be a foundation upon which the Data Base Administrator can dynamically select CPU scheduling algorithms in response to changes in the size of query routines, changes in main memory size, and changes in the accessing patterns of the users.

Computer performance evaluation is a coin having two sides. On the one side we find how a system is actually performing under the set of present existing conditions. Hardware and software monitors are on this side of computer performance evaluation. The other side of computer performance evaluation is more open ended and geared to the scientific exploration of the world of possibilities. What if this variable or that were changed? On this side of the coin we find analytic models and simulation models. Yet, as the complexity of computer information systems grows, many of the analytic models are forced into more and more unrealistic assumptions. Simulation models allow a plethora of system interrelationships to be portrayed and is gaining a wider acceptance as a research tool. Such computer languages as SIMSCRIPT II.5, GPSS, GASP, and ECSS II as well as the development of such simulation packages as SCERT, CASE, and SAM have brought simulation to an ever growing number of people interested in computer performance evaluation. Most uses of computer simulation have been applied to computer systems divorced from any consideration of a data base. It is hoped that this work will stimulate others to apply simulation techniques to the study of problems related to data base management systems.

The purpose of this work is to help the Data Base Administrator in the selection process of CPU scheduling algorithms for the future development of Product Information Systems. We examine the functioning of three CPU scheduling algorithms in the context of a Product Information System over different query and editing routine work loads, different accessing loads and different memory sizes. To examine the operational characteristics of these algorithms adequately in a realistic product information system environment we describe and use a simulation model as our test bed. This model was written in the SIMSCRIPT II.5 language and compiled on the version D SIMSCRIPT compiler.

The first CPU scheduling algorithm schedules the use of the CPU by query processing routines according to the FIFO (first in, first out) discipline. The second algorithm is directed to the minimization of the average loss in potential sales associated with the customer response time by the reduction of the variance in response time for the customer base. This is accomplished by giving a higher CPU priority to the processing of the queries that have a higher forecasted response time. The third algorithm considered has device utilization as its primary goal. It attempts to minimize the average potential sales loss by selecting, when any random access device (RAD) queue is empty, the routines that are predicted to have the smallest block time. A RAD queue is empty when there are no outstanding page requests for that associated device. Block time is the amount of time a routine runs on a CPU before it generates a page fault. A page is an arbitrarily defined unit of either program code or data which can be moved between main memory and secondary storage. Currently used page sizes vary between 512 bytes and 4096 bytes (Donovan, 1972). A page fault occurs when the address mapping hardware detects that a referenced page is not in executable memory. A page fault can be a program page fault or a data page fault. The former occurs when the next instruction to be executed resides only in secondary storage. The latter occurs when the executing program itself generates input requests for data existing outside executable memory.

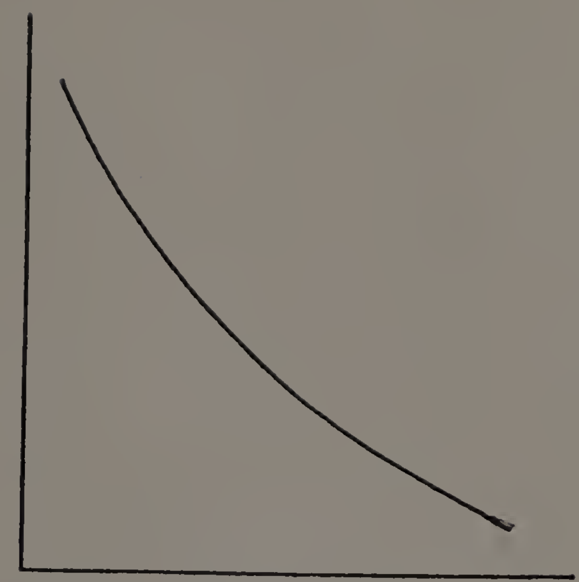
These CPU algorithms are related in the following way. While the first algorithm is used in some simple systems and acts as a standard of comparison, the second CPU algorithm focuses primarily upon the customer as a user of the Product Information System. The third CPU algorithm concentrates on system resource allocation, specifically RAD utilization.

The objective of a Product Information System from the viewpoint of the firm supplying such a system to customers is increased profit due to increased sales resulting from customers becoming aware of products. Once the data base has been constructed and the appropriate accessing methods chosen, potential sales represent the dollar value of sales attributable to the Product Information System itself under the assumption of a zero time interval between customer query and response. In other words, potential sales represents sales irrespective of query delay. Since response time delays are not completely reducible and are attributable in part to the CPU scheduling algorithm employed to process queries, there will always be a gap between potential sales and actual sales.

Customers lose interest in using an informational source if they perceive that the amount of time they must wait for a response is inordinate. This loss of interest can be expressed objectively as a loss function, whereas the response time increases the loss in potential sales increases. Figure 1 depicts typical relationships between response time and system cost and lost potential profit. Figure 1a shows system costs as correlated negatively with response time. Greenberger (1966) presents a number of possible cost or loss curves for delays in information systems, one of which is depicted in Figure 1b. The sigmoid curve in Figure 1b points out some of the recognizable characteristics of customers' reactions to waiting for information, which reactions can be expected to be operable in Product Information Systems. First, the time r_1 represents

Total Information

System Cost

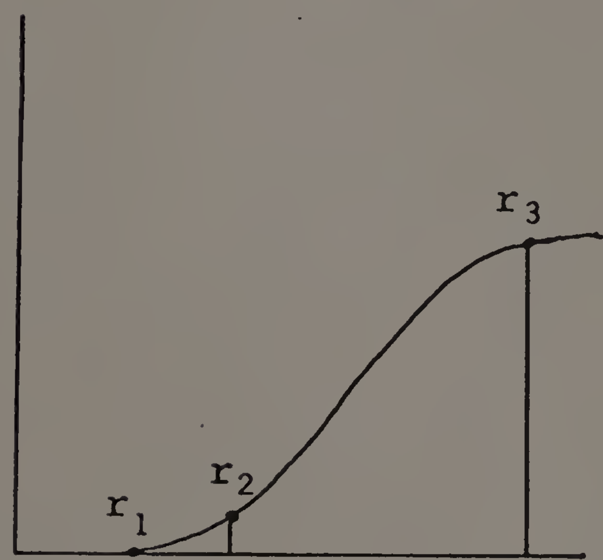


Ave. System Response Time

Figure 1a: Information System Cost as a function of Response Time

Lost Potential

Profit on Sales



Ave. System Response Time

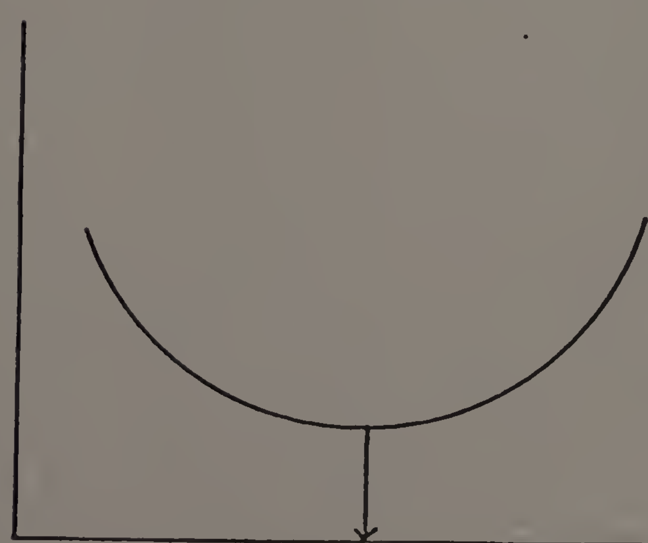
Figure 1b: Lost Potential Profit as a Function of Response Time

Total Information

System Cost +

Lost Potential

Profit on Sales



Ave. System Response Time

Figure 1c: Targeted Response Time

an average insensitivity to the delay in the customer's reception of information. The average customer similar to any time sharing user will accept some level of delay. Beyond this, it is not unreasonable to expect that as the response time is increased the rate of dissatisfaction will continually increase until some point r_3, beyond which only a very few customers would tolerate. The targeted response time as shown in Figure 1c is a tradeoff point between system cost and lost potential profit. This point can be expected to lie beyond r_1 in Figure 1b and most likely lie somewhere near the foot of the sigmoid curve. Obviously, actual values for r_1, r_2, and r_3 as well as the precise slope of curve between r_2 and r_3 will differ depending upon the information system in question. The actual values for a Product Information System remain a subject for future research and are determined in part by the nature of the information in the data base and in part by the psychological attitude of the customer.

A CPU scheduling algorithm influences response time by its intrinsic overhead and by its priority handling discipline. A CPU scheduling algorithm itself demands the use of the CPU as a resource. From the point of view of the computer system the CPU scheduling algorithm is merely necessary overhead. From the point of view of the routines which process queries the CPU algorithm is a competitor for CPU usage. Therefore, the more instructions that a CPU algorithm executes in setting priorities on query routines, the more the response time tends to increase. Yet, this is only partially true, since one trusts that the investment in this overhead will bear suitable returns by adjusting the response time downward through increased parallelism in operations or some other means of lowering the potential sales loss.

The Data Base Administrator is faced with many unknowns in the consideration of the CPU scheduling algorithm. Will a given algorithm actually result in an increased response time for the customer either through its own overhead or through its inefficiency? Does one CPU scheduling algorithm give a lower response time and, if so, under what system parameters and user work loads can this be expected to occur? Are some CPU algorithms more insensitive to their processing environments than others?

In this paper we attempt to answer these questions for the three CPU scheduling algorithms described below.

We shall proceed to set forth our contribution to the study of CPU scheduling algorithm performance in Product Information Systems in the following way. In the remainder of Chapter 1 we will continue to explore the problem areas related to response time and CPU scheduling. Chapter 2 delineates and describes the subsystems which comprise a Product Information System. These include the customer base, the terminal subsystem, the operating system, the data base management subsystem, the random access device subsystem, and the data base itself; Chapter 2 also reviews the literature relevant to each subsystem. Chapter 3 describes in detail the operation of the three CPU scheduling algorithms, the experimental environment, and the hypotheses; in this chapter we then defend and explain the use of simulation modeling as a method for studying the behavior of the three CPU scheduling disciplines. We then set forth the simulation model of a Product Information System and present the experimental parameters under which the model was run. We then consider the statistics needed to insure a valid model and interpretation of the experimental results. Chapter 4 presents the results of the experiments and discusses the implications of these results.

While the selection of an appropriate CPU scheduling algorithm is not the only determinant of response time, nevertheless many of the other determinants of response time also affect the performance of the CPU scheduling algorithm and vice versa. There are therefore secondary influences on response time. One example of this is the balance, or lack of it, in channel and secondary storage device utilizations. If a channel or random access device becomes overloaded, the queue feeding it grows and the response time increases. This is especially aggravated when a number of routines must be executed before an answer to a query can be given, as in a Product Information System. Channel and device balance, however, are a function of not only what accesses are being made to which channels and devices, but also the arrival rate of page requests to secondary storage devices for both program and data pages. Under high channel and device utilization, Chen (1973) has shown that in a hierarchical secondary storage configuration a minimum average accessing time can be obtained only by favoring the faster devices. Therefore as overall device utilization increases, the accessing load should therefore be shifted to the faster devices, even though proportionately this would seem to make the accessing load unbalanced. What is important to note is that this arrival rate of page requests for information from the RAD subsystem is determined in part by the CPU scheduling algorithm. Thus, if the Data Base Administrator wishes to consider the balancing of his channels and devices for maximum parallelism, he must also consider how the CPU algorithm will effect the arrival rate of page requests.

The Data Base Administrator can retain a balanced channel utilization without moving pages of information from one device to another by the appropriate stringing of these devices on multiple channels. Thus, accessing loads on channels can be shifted automatically to available channels dynamically. The Data Base Administrator is still faced with the problem of the lack of balance on the devices themselves.

The Data Base Administrator's first task in establishing a balanced RAD configuration is to determine the customers' accessing pattern, i.e., which products the customers are querying. This is called the product frequency distribution. As accessing patterns change, the logical and physical organization of the data base become out of phase with the accessing pattern. Device imbalance can be detected by hardware or software monitors. Short monitoring routines can be constructed and linked to the Data Base Management System so that, when the monitor is active, it will trap the product identification numbers and subsequently generate the product frequency distribution. A hardware monitor can record the device utilizations. The data collected from these monitors can then be used as a basis for device reallocation and for recombining product information over the hierarchical storage devices. A Data Base Administrator's task of balancing access loads on the RAD subsystem is highlighted by the dependency relation shown in Figure 2. The role of the CPU scheduling algorithm is critical in this decisional process.

Product frequency distributions determine optimal device to channel connections and page allocation on random access devices. Both of these in turn determine channel and

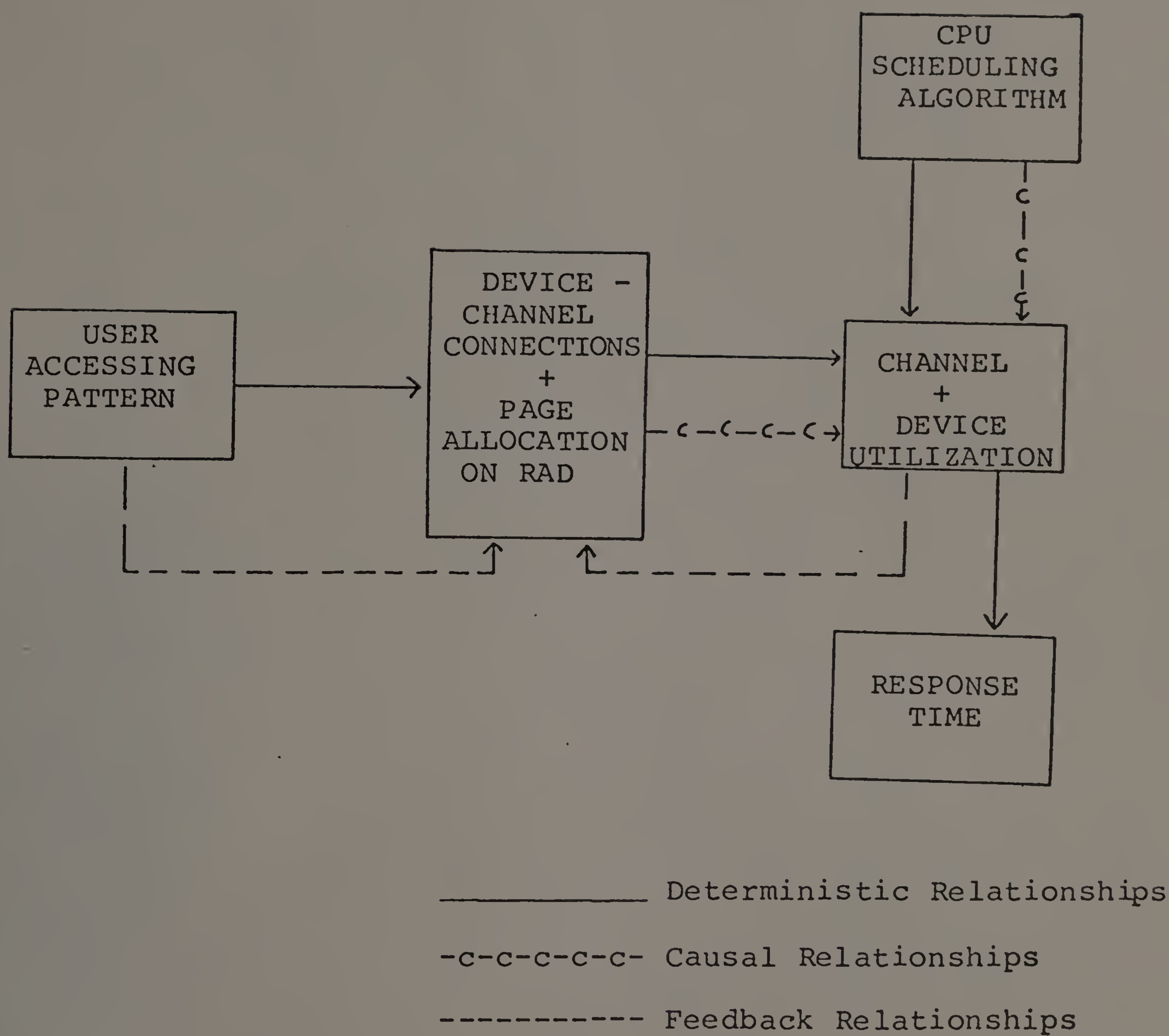


Figure 2: Relationships between CPU Scheduling Algorithms and Other Determinants of Response Time for a given Hardware and Software Configuration.

device utilizations. The balancing of these utilizations is a partial determinant of response time. The channel and device utilizations are also simultaneously determined by the CPU scheduling algorithm.

As Figure 2 shows, the Data Base Administrator interacts in this set of dependencies at stage 2 by controlling the device to channel connections and the allocation of pages to the RAD subsystem, using feedback information from the product frequency distribution (user accessing pattern) and the channel and device utilizations. Figure 2 shows two types of relationships: deterministic and causal. If a relationship is solely deterministic, the DBA has no control over it. If, on the other hand, a relationship is both deterministic and causal, the DBA has partial control. Here the DBA can make decisions or choices for the inputs. Once chosen, however, the outputs then become deterministic.

Fine tuning the data base as page reallocation can have various degrees of complexity. The smallast amount of disturbance is generated by intra-device page reallocation on the same hierarchical level. A level is a set of devices having the same accessing characteristics. Next is inter-device page reallocation, in which pages are moved from one device to another while remaining on the same hierarchical level. Next there is inter-level page reallocation, where a page is moved to either a faster or slower device. This is commonly called migration. Lastly, there is a complete reorganization of the data base over the existing channels and devices. The last two types of fine tuning may also necessitate a change in the coupling of devices to channels. One can consider fine tuning as ending when a consideration of new devices to support the data base is implemented.

For the scope of this paper we assume the Data Base Administrator has the ability to keep the devices and channels balanced either automatically through hardware and software or by manually adjusting the data base. Our concentration will be on the CPU scheduling algorithm and how it affects response time, but one should keep in mind how the CPU algorithm influences other data base decisions besides CPU processing.

Response time is also influenced by the size of query routines. One may expect that as Product Information Systems evolve, new and more sophisticated query languages and editing routines will be introduced, producing larger query routines. These larger routines will exert greater demands on the CPU as a resource of the computer system. The size of main memory also influences response time. For as main memory is reduced more program page faults are generated, which cause an increase in the demand paging usually to the fastest devices, since program pages are usually allocated to the fastest random access devices.

This in turn can produce imbalances in the channel and device utilizations, causing again delays in response time. These program page faults result in longer response times, even when devices and channels remain balanced, since when a block occurs, such executing routines are taken off the CPU and put on a wait queue.

CHAPTER II

SUBSYSTEMS OF A PRODUCT INFORMATION SYSTEM

A Product Information System can be characterized in general as an information system which supplies a set of customers with information concerning product specifications, prices, and availability. The development and maintenance of such a system is the responsibility of the firm that sells these products. The objective of a Product Information System is to stimulate sales by providing the customer with up-to-date answers to his product queries in an acceptable time frame. In a dedicated Product Information System the Data Base Administrator has the responsibility for the development and maintenance of the data base and the delineation of the specifications for the hardware and software which directly relate response time to data base accession. This chapter sets out the design characteristics of a Product Information System from the viewpoint of the Data Base Administrator.

A Product Information System may be developed as a set of interconnecting subsystems. These subsystems are depicted and labeled alphabetically in Figure 1. The numbers in this figure represent the sequence in which each part of each subsystem comes into play in the processing of a customer's

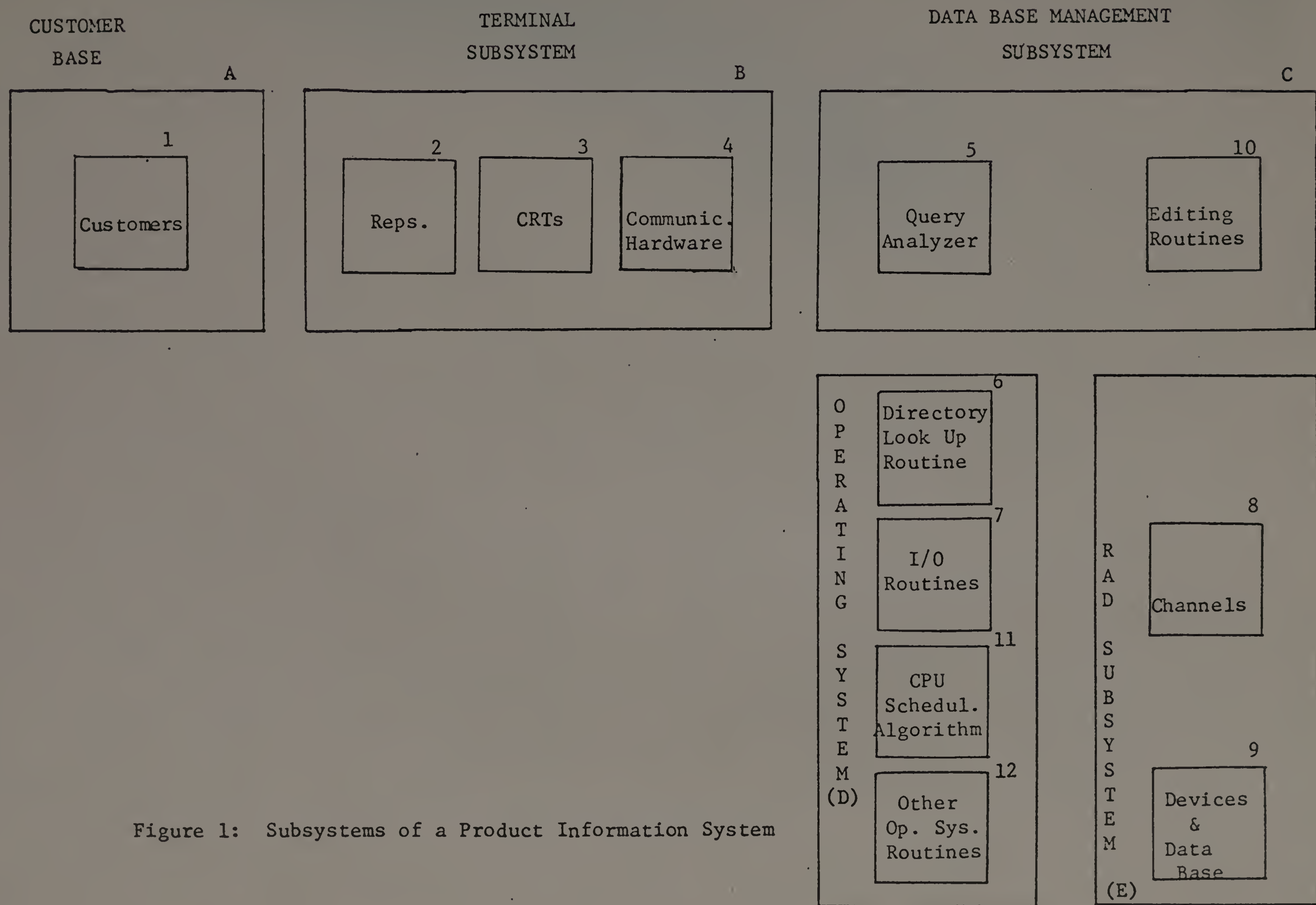


Figure 1: Subsystems of a Product Information System

query. The first subsystem is the customer base (A), the set of customers for whom the information is targeted. Linked to the customer base is the foremost extension of the computer subsystem, the terminal subsystem (B). The terminal subsystem has both a human and a machine component. A representative of the firm (2) acts as an interface between the customer and the computer. A customer's query comes into the firm by telephone and is reduced by the representative to a fixed query format. The representative types in the query at a CRT (3), which query is transmitted by the communications hardware (4) to main memory. The third subsystem then begins to operate on the query. The query analyzer routine (5) of the data base management subsystem (C) transforms the representative's input into data item names and operations. The directory look-up routine (6), which is part of the operating system (D), then maps each data item name to a secondary storage device and address within that device. Requests for data are queued for accession by the operating system. Later the I/O routines (7) of the operating system read in the necessary data from appropriate channels (8) and secondary storage devices (9). Then the second member of the Data Base Management Subsystem (DBMS), the editing routines (10) reformat the data accessed from secondary storage into easily comprehensible information before the information is returned to the CRT. The editing routines take as their input both the data accessed from secondary storage and the operations to be performed on this data as specified by the query analyzer. The DBMS acts then as a bridge between the incoming query and the physical location of the answer. The DBMS has a transformation function which translates incoming logical queries into a form that can be used by the operating system for physical accession. Secondly, the DBMS acts as another bridge, taking the raw page inputs from the random access device (RAD) subsystem (E) and transforming them into answers.

The operating system (D), although a complete system in itself, is considered as but one of the subsystems in a Product Information System. Its functions may be generalized under the concept of resource management in a multiprogramming environment. An operating system then can be viewed as having a controlling function in that it arbitrates a constantly changing set of demands for scarce resources. Its functions include security, integrity, and the allocation of such resources as CPU time, main memory space, and devices. Virtual storage operating systems are a class of operating systems that have been in existence for many years. Their continued use in information systems, at least for the next decade, is assured, especially if IBM continues to stress them, while at the same time withdrawing future support for non-VS operating systems. The distinguishing features of VS include their controlling of the movement of information between main memory and secondary storage as well as their attempted solution to the memory fragmentation problem. In a virtual storage environment, information is considered as clustered into large blocks called segments or into smaller uniform clusters called pages. With main memory size as a constraint, information is moved between main memory and secondary storage in pages as needed. A virtual storage operating system attempts to assign main memory dynamically to contending program and data pages. Two routines that belong to a virtual storage operating system which are of interest in this work are the directory look-up routine (6) and the CPU scheduling algorithm (11). The CPU scheduling algorithm (11) assigns priorities to the routines contending for CPU service, namely (5), (6), (7), (10), and (12).

The random access device (RAD) subsystem (E) has a hardware and a software component. The hardware includes primarily secondary storage devices (8), which are hierarchically organized devices, and the selector channels (9), which connect the devices to the mainframe. The software component of the RAD subsystem includes the product data resident on the secondary storage devices as well as possibly both pages of DBMS routines and operating system routines.

If main memory is sufficient to hold all operating system and DBMS routines, then only data pages representing the product information are subject to paging. Otherwise, program pages also contend for RAD accession. Where main memory is a limiting factor on system performance, the Data Base Administrator may select the DBMS routines that have the highest CPU reference frequency for permanent residence within main memory and allocate the others to one of the fastest RAD devices. As we shall see when we consider the structure and operation of virtual routines, the size of main memory in relation to the total size of the DBMS has far-reaching effects on response times.

Page accession from RAD devices can occur either by memory management using segmentation or demand paging on the one hand or look-ahead paging on the other hand. In demand paging, a program or data page is brought into main memory from secondary storage when a block occurs. In look-ahead paging, pages are brought into main memory in anticipation of their need. Look-ahead paging can be used successfully for program pages, provided the sequence of program code is known beforehand. We cannot assume that look-ahead paging will be applicable in a Product Information System environment, even though the size of each processing routine is known before the information system becomes operable. In a Product Information System the content of the queries coming into the system as well as the processing time of each routine must be considered a random phenomenon. Yet, for main memory management all is not random, for the sequence of the DBMS routines as units is predeterminable and the probability that a given routine will be needed in processing a query can be calculated. This latter information can be used in constructing a page replacement strategy. When demand paging occurs, provisions must be made to make room in main memory for the page coming in. Demand paging, therefore, can initiate page replacement algorithms which select pages for removal from main memory.

We have looked at the operation of a Product Information System from the viewpoint of its subsystems and the modules within each of its subsystems. We exclude from consideration in this paper other functions that must be carried out by a DBMS but which could be performed in a Product Information System during off hours, presumably at night. Some of these functions are the addition of new information into the data base, the deletion of outdated product information, and the changing of logical relationships within the data base. Providing backup facilities, e.g., off-loading the data base to tapes and physical reorganization of the data base, can be expected to consume considerable non-prime processing time. The correction of erroneous data entered into the system, however, will no doubt be a candidate for immediate update.

A Product Information System can also be portrayed as a system of processing stages and queues. Considering a Product Information System in this way highlights the relation between the user's query and the system response time. Response time is the accumulation of queue waiting times and various service times in this cyclic network queueing system. Figure 2 shows a Product Information System from this viewpoint. Parallel operations between main memory processing, RAD subsystem and terminal subsystem processing, however, exclude the possibility of a simple sum of all queueing delays and service times. In Figure 2 the customer's query (1) enters a customer telephone queue (2), where calls are temporarily held until a representative of the firm is available to handle the call. When available, the representative sitting at a CRT greets the customer (3a) and listens to the customer's query. The representative considers how the query is to be formulated for input to the computer (3b). This is termed the <u>think time</u>. After the query has been typed on the CRT (3c), it is moved over a multiplex channel (4) to a dynamic input buffer (5). This buffer is a DBMS input queue, a waiting station as it were for subsequent processing by the query analyzer (6). These input buffers in main memory are allocated by the operating system from a common buffer pool. The output from the directory lookup routines (7), which is a set of page accession requests from the RAD subsystem, is

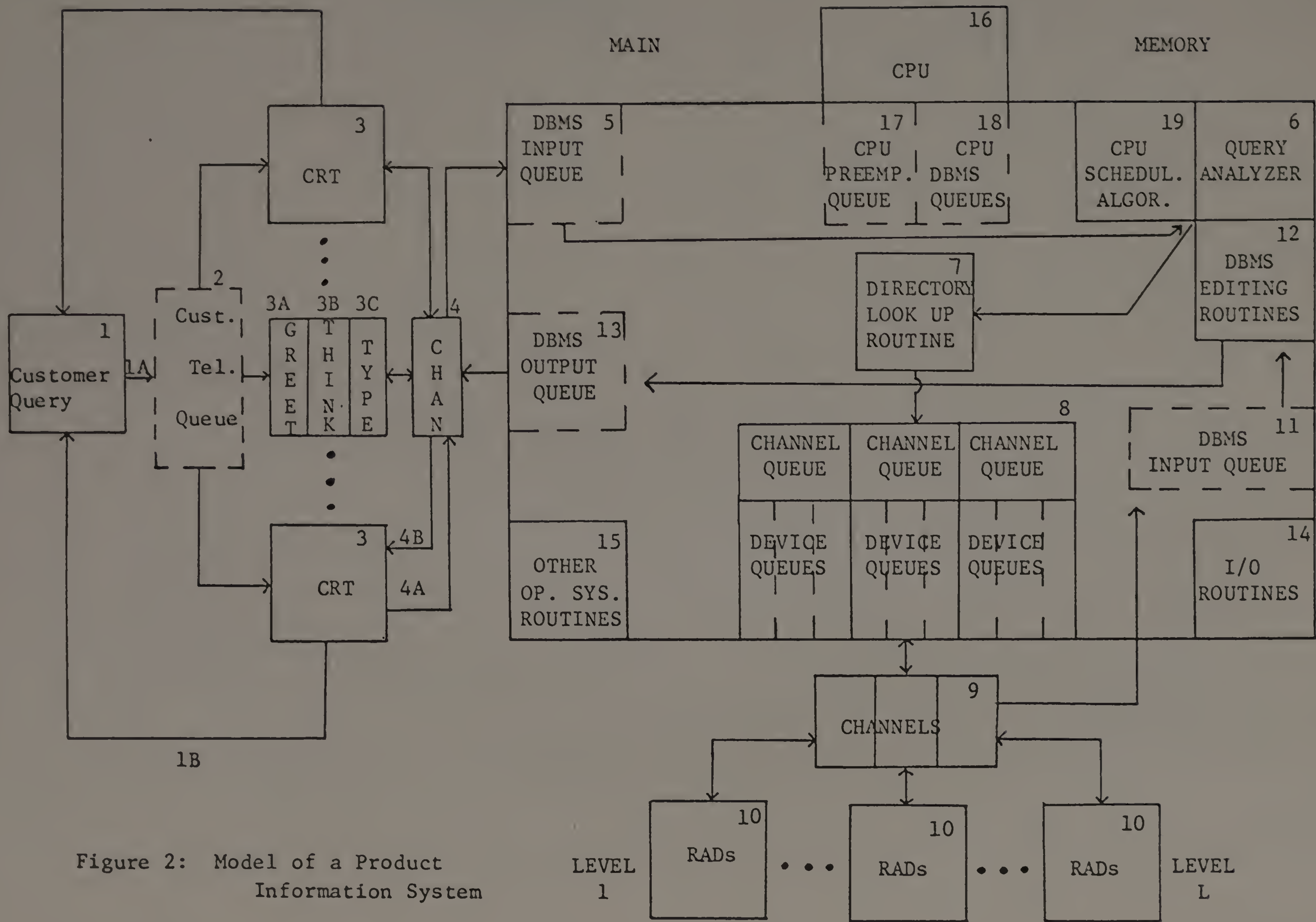


Figure 2: Model of a Product Information System

then put into appropriate channel and device queues (8). The channel and device queues, while most likely remaining in main memory, are logically distinct, although they need not be physically distinct. One of the operating system's I/O routines (14) then selects needed accessions from the channel and device queues. Retrieved pages are then placed in the DBMS input queue (11). The DBMS editing routines (12) process the data pages in the DBMS input queue, then move its output to a dynamic output buffer (13). These DBMS buffers are the output counterparts of the dynamic input buffers. Upon leaving main memory and returning to a CRT the query would have completed a cycle through a network of queues.

This network is shown to be even more complex, when we recognize the effect of demand paging for program pages on response time. When one of the DBMS routines processing a query incurs a program page fault, a subcycle of queueing waits and additional servicing occurs. From the point of view of the query being processed, its control of the CPU is relinquished. The relinquishing of control of the CPU is usually brought about automatically by a hardware interrupt. The query then remains suspended until the necessary page is retrieved from the RAD subsystem. It is <u>as if</u> the query were sent on a diversion from the CPU to a channel and device queue, then through a channel to a device and back again through a channel to main memory, where the CPU scheduling routine places it again in a CPU queue, until it can resume being processed on the CPU.

Within the computer supporting a Product Information System there is another system of queues which influence response time: the CPU queues. The CPU pre-emptive queue (17) and the CPU DBMS queues (18) feed into the CPU and contend for processing time. The CPU pre-emptive queue is composed of I/O routines (14), the CPU scheduling routines (19), and other operating system routines (15). The CPU scheduling algorithm (19) maintains the CPU pre-emptive queue and the CPU DBMS queues, as well as gathers necessary statistics on query activity in order to accomplish this priority handling function. A CPU scheduling routine has various synonyms: the process scheduler, the dispatcher, the low-level scheduler (Madnick and Donovan, 1974). This scheduler keeps track of whether routines are ready for CPU running, whether they are actually on the CPU, or whether they are in the blocked state. Also the CPU scheduler determines what priority is to be given to each routine. It links each routine into its proper place within the CPU queues. It sets up the necessary transfer to the next routine when the current routine on the CPU is terminated (Donovan, 1972).

The "other" operating system routines (15) handle the maintenance of dynamic I/O buffer functions. Data being read into these buffers must then be transferred either to a DBMS input queue or back to a CRT. Other operating system routines (15) also control memory management and other operating system functions like recovery after a system crash. The CPU DBMS queues (18) contain images of the query analyzer (6), the directory look-up routine (7), and the editing routines (12). In most information systems the directory look-up routine is considered part of the operating system rather than part of the DBMS.

Obviously one can make minor substitutions or suggest alternative designs for a Product Information System, but from the Data Base Administrator's point of view we consider the above functional processing units and potential response time delay points as of prime importance. We now look at each subsystem of a Product Information System and relate the relevant literature to its functioning.

The Customer Base

The set of customers targeted by a firm to use a Product Information System is considered the customer base. In order for a Product Information System to be successful a firm obviously must have products which generally satisfy the consumer. Having satisfactory products, a firm's objective is then to get the customer to use the information system. Advertising and other promotional activities will foster this objective. But once the customer begins to use the Product Information System, the most difficult and longest lasting objective then comes into play: to keep strong the customer's desire to use the system. Two principal factors influence a customer's decision to continue to use such a system: the information provided by the system must be relevant and timely. In this work we will assume that accessibility, recall, and precision are adequate. Accessibility refers to the customer's being able to obtain the information he desires. This assumes his query is not blocked from entering the terminal subsystem. In the design phase of a Product Information System the Data Base Administrator should stipulate as one of the system requirements that there be a low probability that a customer will be put on telephone hold before being able to converse with a representative. Recall is a measurement of the proportion of relevant material actually retrieved. The precision of a response is the proportion of retrieved material actually relevant (Salton, 1972). Our concern will be with the problem of obtaining an acceptable response time for the customer base by the use of an appropriate CPU scheduling algorithm.

Characterizing the Accessing Pattern

The Data Base Administrator needs a means of characterizing the customers' requests for information. The relative frequency distribution and the absolute frequency distribution objectively portray the informational needs of the customer base. It is reasonable to assume that some products would exhibit substantially higher accessing rates than others. The study of inventory systems consistently shows product differentiation in both the cost (value) of inventory and the amount of sales from inventory. The "ABC" method groups inventory costs into three categories. Greene (1967) says

> In many inventories it can be observed that a <u>few</u> of the products, group A, account for the <u>greatest</u> cost; some of the items, group B, account for a <u>small</u> amount of the cost; and group C accounts for a <u>very small</u> amount of the cost.

Differentiation in product sales is frequently expressed by the 80 - 20 rule; 80% of the sales is attributable to 20% of the products.

The customers' accessing pattern can be partially specified for the Data Base Administrator through the relative frequency distribution as shown in Figure 3. This distribution can be specified as follows. Given a set of n pages $p_1, p_2, \ldots p_n$ constituting a data base and a set of corresponding relative access frequencies $a_{p_i}, a_{p_2}, \ldots a_{p_n}$ such that $a_{p_i} = f_{p_i} / A$ for i = 1 to n, where f_{p_i} = the number of

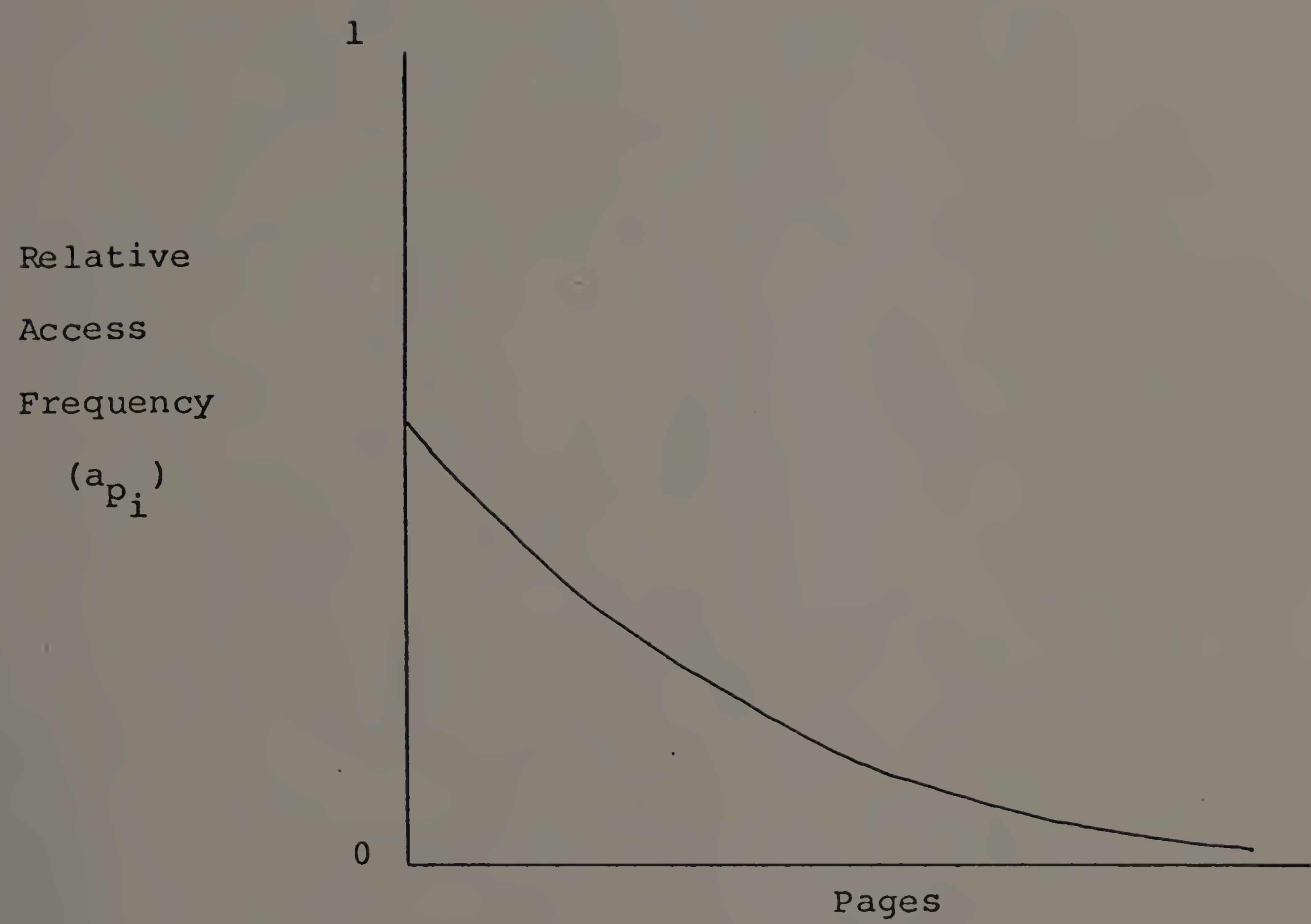


Figure 3: Relative Access Frequency Distribution

accesses per selected time period for page i and A = the total number of accesses to the data base per time period, order the access frequencies and their corresponding pages in monotonically decreasing order. This will constitute the relative access frequency for the given set of pages stored in the data base.

From the Data Base Administrator's point of view, the relative access frequency distribution partially expresses the accessing pattern of data base usage. It expresses the access frequency of each page relative to each other page in a descending sequence along the abscissa. Although this distribution is discrete, it can be considered continuous in a system having a large number of pages in the data base. Over time this distribution will change, reflecting changes in the customers' informational needs.

There are two sources of changes that affect the relative access frequency distribution. The first is a change in the users' informational need for a given page, expressed as a change in the rate of accessing a given page in relation to the rate of accessing other pages. This will change the relative position of the pages along the abscissa as well as the height of pages along the ordinate in Figure 3. The second is the addition of information to or deletion of information from the data base. This introduces new pages and deletes old pages along the abscissa. The relative access frequency distribution is of use to the Data Base Administrator in keeping the channels and secondary storage devices balanced.

The Data Base Administrator must also consider the absolute amount of accessing, since the absolute amount of accessing also affects the performance of the data base. As the utilization of system resources increases, the system response time also increases. Also, as the absolute amount of accessing increases, the operational characteristics of the CPU scheduling algorithm can be expected to change and thereby influence the response time.

We will use the term response time in two related senses. The more generic use is defined from the viewpoint of the customer himself. Here response time is the time interval between the time at which a customer calls the firm and the time the customer hangs up the telephone after having received answers to his queries. We will refer to this as <u>user response time</u>. In Figure 2 the user response time is the interval beginning at (1A) and ending after (1B) when the customer hangs up the telephone. The user response time can be used to calculate customer utilization in terms of the number of customers processed per hour. The user response time itself reflects customer cycle time and the amount of time the terminal subsystem is busy. A CRT station is considered to be dedicated to a single customer until that customer finishes a transaction, signalled by his hanging up the telephone. A transaction is a set of customer queries. User response time can also be used for determining the number of CRTs, personnel, and telephone lines necessary to support the terminal subsystem.

Response time will also be used in a more restrictive sense, as the time interval beginning when the customer has completed formulating a single query and ending when the first character of an answer to the query appears on the CRT. Here response time is defined from the point of view of the computer and will be referred to as system response time. System response time is the time interval during which the computer part of the Product Information System is operating alone. Here the customer is merely waiting for an answer. To the customer the system response time represents a dead time interval in which he must just wait. In Figure 2 this time interval begins either during (3B) or at the latest at the beginning of (3C). The system response time interval ends at the start of (4B).

Both definitions of response time influence the throughput of a Product Information System. User response time provides the basis for a measure of both the achievable throughput rate and the throughput rate capability of the information system. Stimler (1965) defines the throughput rate capability as "the maximum number of fixed-length-input messages arriving at a uniform rate that the system or subsystem can completely service per hour in a no-error environment." A message in the context of user response time is a transaction. The more specific use of response time, namely, system response time, allows the determination of a measure of the number of queries processed by the computer per time period. These measures taken together provide inferences as to where bottlenecks are developing, whether in the representative-customer interface or in the computer system's processing of queries.

System Response Time as a Delay in Human Conversation

The Data Base Administrator's primary area of concern is the system response time. From his viewpoint the system response time is a more controllable variable than the user response time. The system response time is the most crucial, representing as it does the "dead time" in the conversation between the customer and the representative. Carbonell, et al. (1968) point out that "in human conversations there is a quite apparent intolerance for even relatively short periods of silence." Riesz and Klemmer (1963) studied the effect of transmission delay upon the perceived quality of telephone circuits. In the telephone conversations initial delays between parties of 600 to 1200 msecs, were tolerated. After exposure to delays of 2400 msecs, not only was there no adaptation to the effects of the delay but there was also a rejection for delays at the 600 and 1200 range. While one might expect more tolerance on the part of customers in a Product Information System, one must be on the lookout for a similar backlash. It is important to realize that the system response time is not only imbedded in the user response time but that the system response time becomes as it were an extension of the human conversation. A customer will be unwilling to use the Product Information System if he perceives that this dead time is too long. Schwartz (1974) poignantly summarizes the customer's reaction to waiting for information.

> After a certain point, waiting becomes a source of irritation not only because it may in itself be wearisome, boring, and annoying, but also because it increases the investment a person must make in order to obtain a service, thereby increasing its cost and decreasing the profit to be derived from it. This loss to the waiter is related to the fact that time is a finite resource; its use in any particular way implies the renunciation of other rewards and opportunities.

Waiting for information then is perceived as a cost or investment. Owen (1967) points out that the objective of a decision maker (customer) when considering an investment in information is clear. The decision maker hopes the investment will reduce uncertainties involved in his problem. The important point is that he must assess, before the fact, the value of the information gleaned. Will the uncertainty about the problem be sufficiently reduced and, if so, will this reduction be worth the cost?

Once a customer has begun to use a Product Information System, this judgment will be a function of his past response times. This has implications for the Data Base Administrator, when he is developing page placement strategies for secondary storage.

The customer's perception of the actual length of the system response time is dependent upon his cognitive state. If the system response time is perceived as boring, we can expect the subjective time rate, the feeling that time is progressing quickly or slowly, would decrease (Graef, 1970). This would tend to aggravate an already poor system response time. The customer's intolerance of poor system response time is the sum of many factors, including the perceived benefit for the information obtained and the subjective value placed by the customer on his time. Carbonell, et al. (1968) model the acceptability of a given response time as a function of a number of parameters.

> In talking of subjective acceptability we must make a distinction between different forms of it. First we could talk of $\theta_{k,u,T}$ corresponding to a given computer task $\underline{k}$, to a given user $\underline{u}$, and to a given instant of time T. Next we have $\theta_{k,u}$ which characterizes the acceptability for the same user $\underline{u}$ and the same task $\underline{k}$, but without specifying a particular occurrence, so it may be the averaged result of many repetitions of the same computation. As a third possibility

> we have θ_u which averages the acceptability of a system by a given user across tasks and repetitions. Finally, we could talk of θ which corresponds to the degree of acceptability (in terms of response times) of a given time sharing system in general, for a population of users.

While studies have been made of user behavior on time-sharing systems in the university environment (Hunt, Diehr, and Garnatz, 1971; Scherr, 1965), and the research environment (Boies, 1974), these studies have focused primarily on system resource utilization. Yet, marketing studies already provide indications that people are willing to pay more for information than it is actually worth. Green, Robinson, and Fitzroy (1967) demonstrated in a study of both students and executives that both groups consistently paid about 20 percent more for perfect information than would be warranted by a strictly Bayesian decision. There are possible implications for consumers, who in shopping, are faced with the uncertainty of product prices and who are plugged into a number of firms' Product Information Systems. They may be found to be more tolerant of system delays than expected because of bloated expectations. Nevertheless, since little, if any, research has been conducted on users of future Product Information Systems, the precise shape of the loss curve in a Product Information System will have to be determined experimentally. This can be done by eliciting from the customer base their degree of satisfaction with the information system as would be the case in any marketing survey of a new product or service. Such feedback has already been used in the development of existing time sharing systems (Scherr, 1966).

It is reasonable to expect that a customer is indifferent to a subjectively defined short dead time and beyond that point as the system time increases the probability that the customer will no longer use the Product Information System in the future increases monotonically. Such a dissatisfaction probability distribution can be constructed by customer feedback. For each selected response time value r the probability that the customer will no longer use the Product Information System, f(r), can be determined, very simply.

$$f(r) = \left(\sum_{i=1}^{n} s_i \right) / n$$

where n = selected sample size of customers

$$s_i = \begin{cases} 0 & \text{if user } i \text{ is satisfied} \\ 1 & \text{if user } i \text{ is dissatisfied} \end{cases}$$

This probability distribution can then be translated into a lost potential sales function. First, we assume that the firm has basic data on the number of customers with whom it conducts business and the average sales ($) per customer per year is known. Average potential sales represents the difference between the average sales ($) per customer per year for the firm having a Product Information System and without the Product Information System. This information can be assumed to be present; otherwise a firm would not have embarked on the design and implementation of such a system in the first place. This figure would therefore be an upper limit on additional sales, since it represents additional sales from a Product Information System under negligible response time delays. This upper limit would be adjustable as any variable that can be influenced by advertising. One can then combine the average potential sales ($) per customer per year with the dissatisfaction probability distribution to generate a lost potential sales distribution. This loss function can be viewed from different angles. First, it represents lost potential sales. Secondly, it acts as a measure of user dissatisfaction with the Product Information System. Thirdly, it is an inverse measure of system performance.

A pilot project, initiated even before the full system became operational, can be used to gather such sample data. Once operational, data would be collected on a continuing basis, thereby providing the Data Base Administrator with necessary feedback on system performance.

Choosing a Target Response Time

Since the response time to different queries varies as well as the customers' reaction to system generated delays, the Data Base Administrator must decide upon a target response time. Weingarten (1966a) suggests putting response time in a quantitative constraint form. Here a targeted upper limit with an associated probability is chosen. For example, a maximum response time of 30 seconds for 90 percent of the queries may be chosen as an acceptable level of information system performance.

A better approach to the problem of setting a target system response time is one that includes an analysis of the trade-off between total information systems costs and the loss in potential profit from lost customer sales. Figure 1 in Chapter 1 illustrates this trade-off. It is important to note that lost potential profit on sales is also a function of user response time as well as system response time, since user response time affects the throughput of a given information system configuration. In the calculation of lost potential profit on sales, as the user response time increases, the number of customers supportable by the system per time period decreases. Therefore, increased response time has a doubly adverse effect on lost potential sales. First, it restricts the number of customers who can use the information system. Secondly, through dissatisfaction some of those customers who use the system will cease using it. While these two factors are compensatory in nature, this is not the type of homeostasis that an information systems designer seeks to attain.

Reducing Average Potential Sales Loss Through CPU Scheduling

Once an average target system response time has been chosen, the general sigmoid shape of this curve points to CPU scheduling methods for decreasing the average potential sales loss associated with system response time. When comparing the operational characteristics of different CPU scheduling algorithms, one must first determine a standard for comparison. We choose as such a standard the FIFO scheduling discipline, modified by operating system pre-emption. This is one of the simplest CPU scheduling algorithms to implement and demands a small overhead to be maintained by the operating system. This modified FIFO CPU scheduling algorithm can be explained as follows. Given two queues _A_ and _B_ that feed the CPU, let queue _A_ contain the operating system routines that are to be given pre-emptive status, if they are ready to use the CPU. Let queue _B_ be composed of data base management system routines and non-pre-emptive operating system routines that are ready to run on the CPU. Now queue _B_ is ordered on a first-come, first-served basis. As long as queue _A_ is empty and queue _B_ is not empty, the next routine selected to run on the CPU will come from queue _B_. Whenever queue _A_ obtains a member and the CPU is either free or running a non-pre-emptive routine, this member from queue _A_ is given control of the CPU. In the event a member of queue B were running on the CPU, when the routine from A pre-empted it, the B routine is returned to the head of the queue B. The rationale for such a scheduling discipline is that selected operating system routines should get preferential CPU treatment, when the running of these routines is necessary either for routine synchronization or for parallel processing. An example of the latter is an interrupt handling routine for initiating a channel's transference of data into main memory.

With a given customer base generating different queries, each having a unique system response time, one must consider not only the average response time but also the variation in this response time. Human interfaces will vary in processing queries. Individual computer routines processing queries will vary in the time needed to complete such processing. This variation gives us our first method for reducing average potential sales loss. Referring again to Figure 1b in Chapter 1 we see that all response times less than r_1 could be allowed to be delayed up to r_1 with no accompanying loss in potential sales. In other words, given a choice between two routines, X and Y for CPU service at a given point in time, if routine X is predicted to have its associated query answer back to the user in less than r_1 and routine Y is predicted to have a response time associated with it of greater than r_1, then it may be wise to select Y to run next on the CPU rather than the other DBMS routine.

If the general shape of the loss function is sigmoid, one may take advantage of the variation in the system response time in even another way. In the interval $R_3 - R_2$ a decrease in the variation of the system response time can sometimes result in a decrease in potential loss in sales, even though this is accompanied by an increase in the average system response time. Figure 4 demonstrates this. Here $\overline{R}_1$ represents the average system response time under the CPU discipline 1, and $\overline{R}_2$ represents a higher average system response time under the CPU discipline 2. The right side of the solid lined curve superimposed on the sigmoid curve presents high lost potential sales values because of the curvature of the sigmoid curve. Appendix 1 discusses this in more detail.

The above discussion now gives us a viewpoint from which to generate a different CPU scheduling algorithm, the second CPU scheduling discipline we will consider. An algorithm which would implement response time variation reduction can be developed, if the Product Information System captured the average times a query took in passing through various milestones in its being processed. Since system response time is the sum of individual sequential processing stages, each stage comprising a set of routines,

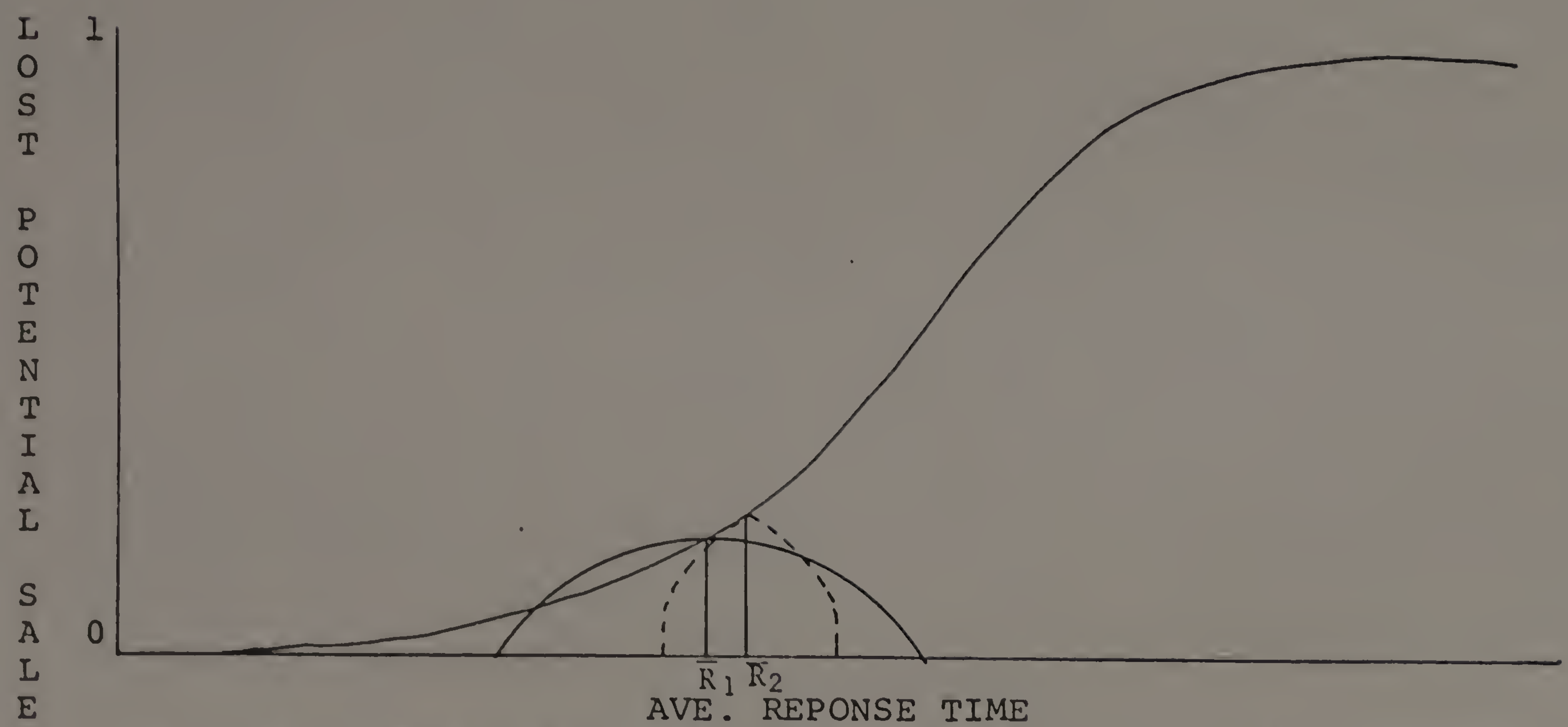


Figure 4: Lost Potential Sales as a function of Ave. Response Time

the beginning and ending times of selected routines can be used as milestones. From this information one can determine for a given query a predicted system response time, updated at various processing points. In addition, this information allows concurrently processed queries to be ordered in relation to their predicted system response times. This ordering in turn can be used as the basis of a CPU scheduling algorithm's priority scheme. This discipline will be referred to as CPU scheduling discipline 2. It will attempt to reduce lost potential sales both by taking advantage of the r_1 time span and the potential savings due to a reduced response time variation.

Another approach to reducing the average potential sales loss is by reducing the average system response time through increased secondary storage utilization. Here we can take advantage of the fact that the characteristics of the DBMS routines in a Product Information System can be known before the system becomes operational. The size of each routine can be determined, not only in terms of its physical size, but also in terms of the average number of executable instructions. In addition, since the size of main memory is predetermined as well as the proportion of code to be resident in main memory, this allows the consideration of a CPU scheduling algorithm that will tend to keep the secondary storage devices accessing when they would otherwise be idle. A CPU scheduler can accomplish this increased parallelism dynamically, setting higher priorities for CPU selection on routines that have higher probabilities of becoming blocked or finishing their need of the CPU per unit time. This algorithm should apply this strategy, however, only at the points in time when the queues feeding the secondary storage devices become empty. There is no expected gain in choosing for execution a routine which would tend to increase the queue load on an already swollen set of RAD queues. We will therefore consider that this algorithm will revert to operating like the CPU scheduling discipline 2 whenever the RAD queues are not empty. The effects of this CPU scheduler will also be studied in the Product Information System simulation model and will be referred to as CPU scheduling discipline 3.

The Terminal Subsystem

In a Product Information System the terminal subsystem is composed of the firm's representatives, the CRTs, and the necessary communication hardware, which links the CRTs to the mainframe. The representatives are catalog order clerks specially trained to be able to input queries at the CRTs and then respond to the customer's verbal query. Transaction time, which is almost synonymous with user's response time, is shown in Figure 5. Transaction time does not

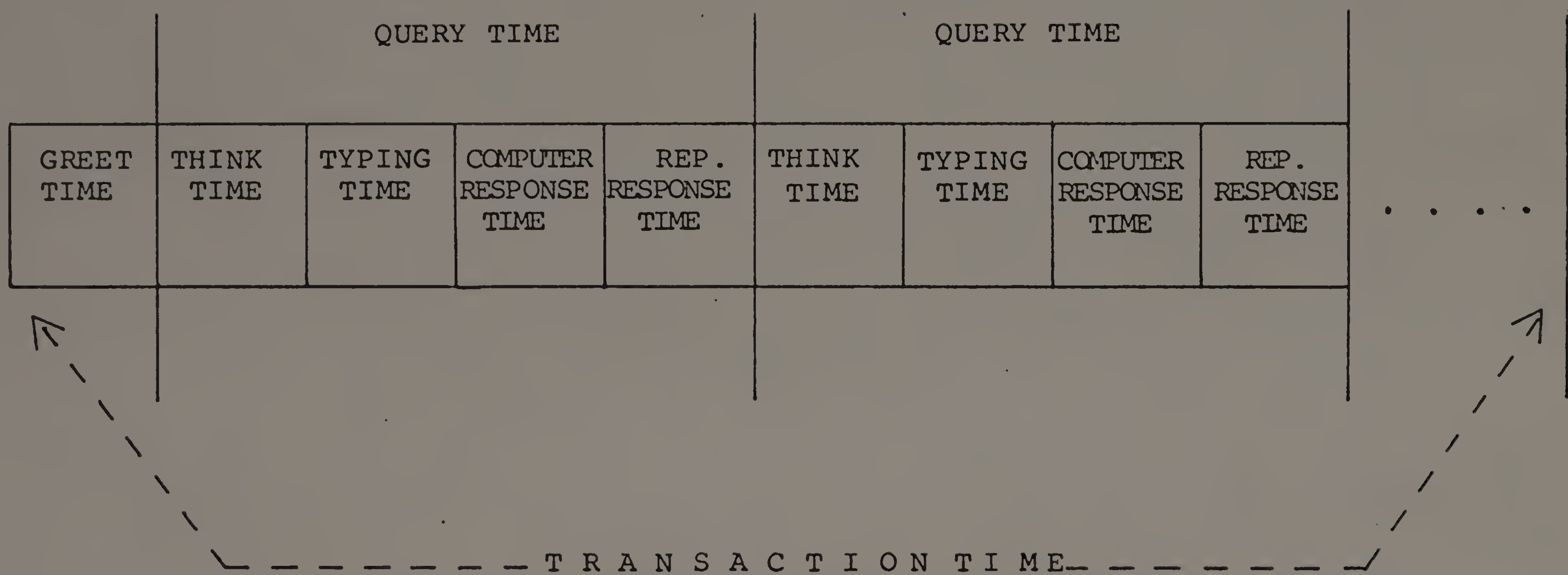


Figure 5: Query and Transaction Time

consider any delays attributable to the customer being placed on telephone hold, since the customer is considered as not having entered the information system until greeted by a representative. User response time does, however, include any delay due to being put on telephone hold.

Transaction time is the interval beginning with the greeting by the representative and terminating with the customer hanging up his telephone. This distinction between transaction time and user response time is justified since, while user response time is the total time spent by the customer to get an answer to a set of queries, transaction time is a measure of how long it takes the Product Information System to handle the customer. These are measures of performance from two different viewpoints. Transaction time is composed of a set of query times prefaced by a short greeting from the firm's representative. Each query time begins with a time period called the <u>think time</u>, during which the customer explains what information is desired and the representative concurrently decides how this query is to be formulated in the query language. The representative then keys in the query during the <u>typing time</u> interval. After the computer system responds, the representative answers the customer during the <u>representative response time</u>. A number of these intervals may be required to satisfy the informational needs of a customer at a particular moment.

The throughput rate of the information system is measured in queries per hour and is determined in part by the speed of the representatives. The think time and the representative response time can be limiting factors for a Product Information System, especially if the representatives are poorly trained. A measure of a representative's performance for the think time interval can be expressed as:

$$\frac{t_c}{t_c + t_r} \times 100$$

where t_c = the time interval during which the customer is expressing his query and

t_r = the time interval from the end of the customer's formulation of his query to the beginning of the representative's typing the query.

This measure of human performance is a measure of parallel processing in terms of how effective the representative is in formulating the query entry, while the customer is still explaining the query.

If the average t_c is known, then the overall average time (Z) that can be attributable solely to a representative in processing a query can be measured as:

$$Z = \sum_{t=1}^{T} \left[\frac{(s2_t - s1_t}{\bar{q}} - (\bar{t}_c + {}^{s}r_t) \right] / T$$

where t = a transaction.

$\bar{t}_c$ = the average time interval during which the customer is expressing his query.

T = the set of transactions for which data has been collected concerning a representative.

s1 = a time stamp indicating the beginning of the greet time for a transaction.

s2 = a time stamp indicating when the customer hung up the telephone.

$\bar{q}$ = the average number of queries per transaction.

$^{s}r_t$ = average system response time per query.

Sl and s2 could be generated by hardware connected to the communication lines between the customer and the representative. If the probability of a customer being put on telephone hold is small when initiating a transaction, the user's response time can be approximated as s2 - sl. When newly trained representatives are put on the Product Information System, data should be collected in order to develop a learning curve, which can then be used both to forecast Z and to measure the individual performance of each representative.

From the s1 and s2 time stamps one can also measure the percentage of idle time at any CRT station expressed as:

$$I = \left[\left[H - \sum_{t=1}^{T} (s2_t - s1_t) \right] / H \right] X\ 100$$

where H = the total time horizon over which the time stamps have been collected.

This measure (I) in conjunction with the percentage of customers put on telephone hold before entering the Product Information System can be used to determine the number of CRT terminals needed at each locus of terminals to support the Product Information System.

The CRTs and the communication hardware act as an intermediary between the representative and the mainframe. A number of attempts have been made to model the interaction between a terminal user and the computer, although most of the studies have been in the area of computational processing (Greenbaum, 1968; Brown, 1970; Scherr, 1965; Denning, 1968; Coffman and Wood, 1966). Boies (1974) presents empirical data on user behavior, duration, and frequency of terminal sessions, but again in a computational environment. Altshuler and Plagman (1974) discusses the user/system interface in corporate data base environment.

The hardware implementation of the terminal subsystem of a Product Information System can be configured in a number of ways. Data Communication/Data Base information systems are currently available (Smith, 1976). These incorporate

1) a network control program (NCP), which handles all line and terminal disciplines, message reception and assembly, and transmission. A network control program may reside in a communications controller like an IBM 3705,

"which permits a wide variety of remote communications terminals over dedicated and switched narrow and voiceband lines and dedicated broadband lines." (Auerbach, 1973). In addition, this NCP can perform communications line control, character and block checking, dynamic buffering, polling, addressing, and error recovery procedures.

2) a message control system (MCS) which determines the routing for each message.

3) transaction processing programs, each of which process a particular type of transaction. These programs are small and merely pass the message to the Data Base Management System, while recording and forwarding back to the MCS any access failures due to such conditions as requested record not found.

4) a Data Base Handler or DBMS, which controls all access to the data base.

Figure 6 illustrates the data transfer between these modules. These modules may be part of a centralized Product Information System, or they may be part of each node computer in a network of computers.

At the remote end of the NCP in a Product Information System there will be alphanumeric CRTs. This type of CRT is grouped into two main families: stand-alone units and clustered CRTs (Asten, 1973). Stand-alone units contain the necessary memory for the screen, the character generator,

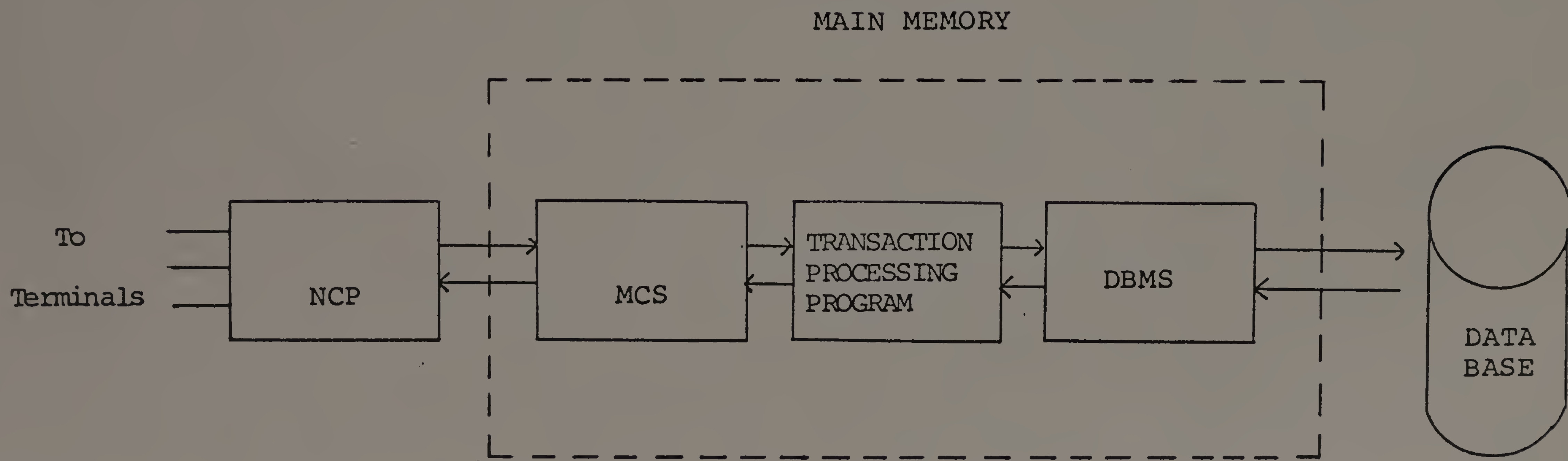


Figure 6: A Generalized DC/DB Design.

power supply, and all circuit logic, while the controller driven CRTs usually have only the CRT tube and the circuitry for the keyboard. The memory and character generator remain in the controller. Controller driven CRTs are usually less expensive but only if enough of them are tied into a single controller. In the development of a Product Information System this tradeoff will have to be taken into account.

Messages may be collected from the CRTs by polling over multipoint lines. The use of nomographs incorporating such variables as number of terminals, number of controllers, modem characteristics, line speed, and message rate have been shown to be useful for first cut design and analysis for multipoint communication lines with polling (Thananitayaudom, 1976). In addition to modems, messages may be processed by concentrators before being sent over long transmission distances and may be routed through message switching centers (Bacon and Bull, 1973; Martin, 1970; Kimbleton and Schneider, 1975). The cost of leasing lines on dedicated digital networks is dropping drastically when compared with traditional private line charges, owing to stiffer competition among the carriers, but mainly because of improved utilization of the frequency spectrum for data transmission (Frank, 1976a, Frank, 1976b).* Yet, if Product Information

---

*Frank states, "I think that we have seen everything that is going to happen in the next five years already here." Others agree that the telecommunications revolution is not

Systems are implemented using communication networks over large distances involving telephone lines, one can expect serious problems of line noise (Hebert, 1976).

There are principally two modes of initiating data transmission. In the normal response mode, a secondary station responds only to polls from the primary station. In the asynchronous response mode a secondary station may initiate a transmission (Auerbach, 1974b). Most remote CRTs today are used on voice-grade facilities, which limit the transmission speed to a practical maximum of 4800 bits / second over the dial network and 9600 bits / second over leased or private lines (Gepner, 1975). The transmission of query and response messages may itself be in half duplex or full duplex and may be either asynchronous or synchronous. If a Product Information System were implemented using time division multiplexors, the asynchronous time division multiplexors perform about the same as their synchronous counterparts over a wide range of parameters as to their arrival rates at the computer, the average delay at the computer, and the job completion time (Pack, 1973). The asynchronous multiplexors may in fact provide considerable savings in transmission costs due to reduced wasted transmission capacity.

---

going to happen as fast as people want because of various state, federal, and local pricing and regulatory agencies (Upton, 1976).

Transmission time is a function of the rate at which messages are generated and their propagation distribution. Denning (1968) points out that, although terminal operators tend to work in spurts, active periods following inactive, and although the character arrival rate from a given CRT varies erratically, yet with many terminals, the total character rate from all the terminals should remain constant. Jackson and Stubbs (1969) studied computer delays and user delays in the communication subsystem of three time sharing systems using teletypewriter-like terminals and Touch Tone telephones. They found that user transmission time and computer transmission time represented only 15% to 40% of the total time. Secondly, they found that 1% to 5% of an average session at the terminal the user to computer channel was transmitting, while the computer to user data transfer was an order of magnitude greater.

Part of the low input rate at a terminal is due to the think time. Scherr's studies (1966) on the MULTICS system indicate an approximately exponential distribution for think time, if we disregard the typing of simple system commands that we find in a programming environment. Schwetman and Deline (1969) in a study of the RESPOND time sharing system at the University of Texas have shown think time peaking at six seconds. If one considers the think time distribution just after six seconds, the general shape of an exponential curve is presented. Fuchs and Jackson (1970) studied many variables related to the communication subsystem of four time sharing systems, including think time.* These systems included two scientifically oriented systems, one business, and one inquiry/response system. They found that for a large number of variables the geometric distribution can be used to describe most of the discrete processes and the gamma distribution can be used to model all of the continuous random variables under consideration. Since the exponential distribution is a member of the class of distributions which comprise the gamma distribution (where the coefficient of variation is 1), the exponential distribution may be used as an approximation. These findings lend considerable support to future efforts of modeling and designing the terminal subsystem of a Product Information System.

---

*It has been suggested that some time sharing parameters like channel interarrival time, which is the time elapsing from the instant the computer finishes a request for computation by a particular console's channel until the instant the next input message from that channel is received at the computer, may be hyperexponential (Coffman and Wood, 1966).

Fuchs, however, did not find that the data in the inquiry system he studied would support the hypothesis of a think time having a hyperexponential distribution. A gamma or exponential distribution provided a better fit.

The Random Access Device Subsystem

Data Bases

A Product Information System will most likely be implemented as a data base system, rather than as a traditional file system, whether it be implemented as a stand-alone system or as an integrated part of the firm's inventory system. Appendix 2 illustrates some of the main differences in design strategy and operation between file systems and Data Base Systems. Shneiderman (1974) presents a bibliography on data base structure useful for the design of both file systems and Data Base Systems. Summarizing current opinion on the use of Data Base Systems, Ward (1974) states:

> The interest in data base management systems is there and the trend toward data base is strong and irreversible. In spite of its problems, most people who become familiar with the concept feel data base advantages are worth the price.

Nolan (1974) concurs to the acceptance by management of the data base concept.

The term data base has been defined in various ways: some definitions being very generic, others too specific to a particular implementation and thereby limiting. While Lucas (1973) correctly points out that to define a data base as "all the relevant information on file for that firm," is too generic, his qualification that this information must be in machine-readable form only goes part way in providing an adequate definition. The term data base has also been defined too narrowly. It has been considered as a file or set of files whose content includes "the information needed by management for response to a wide variety of non-routine, management control needs." (Blumenthol, 1969). This definition, however, prejudices the use of data bases towards solely managerial functions. The definition of a data base can also suffer by a too narrow definition when put into the Procrustean bed of a predetermined view of what specific modules constitute a data base system. For example,

> A data base is the collection of data that represents those facts defined to be of interest to an enterprise. It is an implied (non-disjoint) collection of conceptual record-sets, the conceptual model of that enterprise. It is in addition a (disjoint) collection of internal record-sets, the internal model containing the stored data (ANSI/X3/SPARC Study Group, 1975).

A more balanced definition of a data base, which mediates between the above extremes is

> A data base is an integrated set of files, tables, arrays, and other data structures. In a data base, separate files may exist, but they are linked together to facilitate providing the information needed by a segment of an organization or, ultimately, the organization as a whole (Couger, 1975).

Martin (1973) warns

> There is a temptation to view a data base as a big kettle of alphabet soup that is the raw material input to an information system that processes, refines, extracts, and converts this amorphous mass into information. Unfortunately, this view

is quite misleading, for the data base must be rigorously defined, intelligently selected, and a logically structured complex if valuable information is to be produced at a reasonable cost.

Data Structures and Storage Structures

The Data Administrator's problem of maintaining a data base for a Product Information System is partially rooted in both the physical and the logical associations of data items, partially in the users' accessing pattern, and partially in the selection of memory devices that support the data base. A data base can be viewed in terms of its data structure or its storage structure (Katzan, 1971). Figure 7 shows the distinction between these two structures.

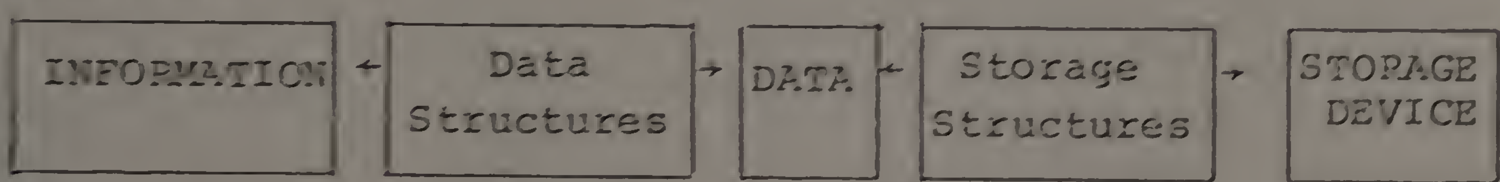


Figure 7: Data structures and storage structures. The data structure represents the logical relationship between the data items, while the storage structure represents the physical structure of the data items on the random access devices. The physical association between data items is expressed by the spatial proximity between data items.

Data Structures

Since information can be defined as data in a meaningful context, there is an implicit assumption that what elevates information above data is the logical aspect imparted to the individual items of data. Comprehensive treatments of data structures are available (Stone, 1972; Flores, 1970). A common form of expressing associations is through a hierarchical or generic relation. A generic relationship exists if a data item can be viewed as a species with respect to a genus data item. For example, the two strings, Massachusetts and Maine, are data items exhibiting a common species-genus relationship to the data item, States of the U.S.A. One way in which these relationships can be implemented is by tree structures.

Salton (1962) has demonstrated how tree structures can be implemented in the form of 2-dimensional matrices and has presented an algorithm for manipulating such tables. Tree structures have been designed for handling large ordered indices which must be updated and retrieved on drums and disks (Bayer and McCreight, 1972). Sussenguth (1963) showed that the use of tree structures for expressing relationships between data items could be an effective compromise between the fast searching but inflexible updating capability of binary searching on the one hand and the slow searching but easy updating capability of chain structures. Later, Coffman and Eve (1970) applied hash coding to searching, adding, and deleting elements from tree structures. Nievergelt (1974) surveys the main results which have been obtained on binary search trees and file organization and concludes that they are among the most flexible methods for organizing large files as well as being reasonably efficient for all of the common operations on a file. Today IBM's Information Management System (IMS) is an example of the embodiment of hierarchical relationships in a Data Base Management System.

The Relational Approach and the Network Approach to data structures are alternatives to the Hierarchical Approach (Date, 1975). The Network Approach is typified by the system proposed by CODASYL's Data Base Task Group. Codd's work on applying set theory to data structures has given impetus to the field of data relational models (Codd, 1970). The relational view of information as contrasted with the hierarchical or generic view has been termed the fourth generation of information management systems (Whitney, 1973). Chen (1973) has introduced the entity-relationship model attempting to unify these different logical data models. The Relational and Network Approach can be implemented in the form of vectors, matrices, stacks, queues, simple lists, two-way lists, ring structures, threaded lists, cellular lists, and inverted lists. Knuth (1975) and Berztiss (1975) have excellent treatments on list structures. Dodd (1969), Lefkovitz (1969), and Martin (1969) have more business system related treatments of the different kinds of logical relationships used in data base systems. Bachman (1965) has demonstrated how chain structures can be used in an actual system, the Integrated Data Store (IDS) system.

Evaluation of Data Structures

While the literature is vast on the subject of data structures and file organization, it still leaves much to be desired on their experimental evaluation, especially in data base systems whose data bases are dynamically changing. For example, at the present time Relational data bases are just becoming commercially available and their implementation is still open to experimentation. The first problem, as Ghosh points out, is that it is difficult to reduce the data structure evaluation problem to a single measure of performance. Trade-off differences in retrieval time, turnaround time, memory space, and maintenance time often seem irreconcilable. Optimal file organization must be based on parameters derived from the data set characteristics, the user transaction requirements, and the hardware specifications (Severance {n.d.}; Lowe, 1968).

The distribution of record lengths in a file have been shown to influence response time and throughput (Abate, Dubner, Weinberg, 1968). Related to this is Maxwell and Severance's work in which they present an analytical model for the comparison of different techniques of representing data on secondary storage, including null and non-null values, but the performance measure they use is restricted to minimizing the average amount of data accessed from secondary storage (Maxwell and Severance, 1973).

Senko, et al. (1969) have developed a file design handbook, a collection of charts and tables, providing guidance in the selection of near optimum designs, based on IBM access methods. This group also developed the File Organization Evaluation Model (FOREM), a discrete simulation model which allows the analyst to view the effect of individual file design changes.

Some individual data structures have been compared. There are definite advantages to the use of hashing algorithms or what is called key-to-address transformations. This is the lowest level of logical association, degree 0. Here each primary key as a data item is associated or mapped to a physical address and little or no association exists at this level between records in a data base. The only logical association that exists is between the data items making up an individual record when that record is stored as a set of physically contiguous data items. However, hashing can be used in combination with other logical association methods to complete the nexus between higher logical associations, expressed, for example, as a Boolean expression, and physical storage. One example of this occurs when a query is mapped to a subset of an inverted list and from there each individual key is hashed to its physical address on secondary storage. Price (1971) presents advantages of the use of key-to-address transformations in comparison to table lookup methods for searching files.

Lum, Yuen, and Dodd (1971) provided experimental data on the comparison of a number of key-to-address transformations in large formatted file systems, showing that the division method was best, when compared with a number of well-known hashing methods.* Maurer (1968) points out that the execution time of the division method could be expected to be faster than multiplicative hashing algorithms, like the mid-square method. Later, Lum (1973) again demonstrated the superiority of the division method when he subjected the key space to probabilistic retrieval. Ghosh and Lum (1975) later showed that indeed the division method even performed better than a theoretically perfect randomization method. Other parameters have been included in the evaluation process of key-to-address transformations: storage cost for the whole file, cost of storage for one record per time unit, cost of prime area access, and overflow accesses (Maurer and Lewis, 1975).

Siler (1976) in studying models of large scale data retrieval systems, compared the performance of the inverted

---

*The methods they considered were division, digit analysis, folding, algebraic coding, radix transformation, and mid-square.

list organization with the threaded list and cellular list organizations under varying degrees of query complexity. He demonstrated that, when performance was measured in terms of combined seek and latency time, the inverted list structure was superior except for the most simple types of queries. He also found that poor performance results from mixing these list structures.

Storage Structures

Once the logical associations of a data base have been determined and a decision has been made on how these associations are to be implemented, the Data Base Administrator must address the problem of storage structures. Bachman (1972) states that "the storage structure portion of data base systems is concerned with the mapping of a logical file and its records onto storage media so that data may be reliable and efficiently retrieved at some subsequent time." Since a Product Information System would be expected to be I/O bound, as most retrieval systems are, the structuring of data on physical devices is important. While processing speeds have increased by a factor of almost 10,000 from 1955 to the present, access time to a rotating device has remained almost constant at between 10 to 100 milliseconds (Flynn, 1972).

The physical storage of data on secondary storage devices is a three leveled problem. On the first level the problem can be stated as: given an individual device or set of similar devices, allocate data to this device. We will refer to this as the type 1 storage problem. On the second level the problem is, given a set of devices connected directly through control units and channels to a computer, allocate data over these hierarchical devices. We will refer to this as the type 2 storage problem. The third level represents a higher level of complexity: the allocation of data in a total information system. This third level includes the possibility of data being distributed at nodes of a system interconnected by a communications network. We will refer to this as the type 3 storage problem.

Type 1 Storage: Storing Data on Similar RADs

Models of drums and disks as individual secondary storage devices have been constructed. Coffman (1969) constructed mathematical models using queueing theory for drums for both the cases where requests for information arrive singly and at random. Arora and Jain (1971) applied queueing theory to the analysis of queues at drums under the assumptions that the requests follow a uniform distribution, requests are serviced on a FIFO basis, request arrivals are Poisson, and service time for any request follows a negative exponential distribution.

Manocha, Martin, and Stevens (1971) point out that for a fixed head per track device the two performance parameters are queue size and the number of sectors per track. Burge and Konheim (1971) use a Markov Chain to calculate the average number of requests satisfied in a fixed head disk revolution. They model the request buffer, where the state of the buffer at the start of a cycle is represented as vector X, where x_1 is the number of requests in the buffer that reference the ith sector. Their assumptions include constant availability of the channel and an I/O load heavy enough to keep the fixed length buffer full of requests. From this analysis Denning (1972) later deduced a formula for drum efficiency that agrees almost perfectly with other simulation results. Drum efficiency is calculated as

$$e = \frac{2r}{2r + m - 1}$$

where r = the size of the request buffer feeding the drum and

m = the number of sectors on the drum.

There are a number of possible storage criteria that can be used in a Product Information System. Among these, two criteria can be singled out for consideration:

1. The access time to a record should be a function of the customer's subjective utility for that product.

2. The average access time should be minimized; and to implement this, one should employ:

Strategy 1: The most frequently accessed product records should be on the fastest devices (assuming channel balancing is not violated and Chen's considerations on balancing are heeded.*

or Strategy 2: Records, which have a large joint probability of being accessed in a single query, should be located in adjacent memory areas, e.g., same track or cylinder.

These storage criteria are mutually incompatible. Yet, at first glance it would appear that strategies 1 and 2 under criterion 2 could be implemented in a complementary manner. Data items with the same accessing frequency could be grouped into sets. Within each set, the data items could be assigned physical contiguity based on their joint probabilities.

When we critically evaluate these criteria for a Product Information System, we see there are valid reasons for accepting criterion 2 only, and under that criterion

---

*In most cases, optimal system performance does not occur when each server is utilized to the same degree but rather when 'bottlenecks' are created at the faster servers. Another is that use of slower servers may degrade rather than improve the system performance when the input load is sufficiently low (Chen, 1973).

strategy 1 alone. While subjective utility measures have been proposed for some information systems (Wilson, 1974), these do not seem applicable to Product Information Systems, since there is no known way of determining a customer's subjective utility for a product at a given point in time. If the belt on my pants breaks, a simple pin may have the highest momentary utility. It is important to point out that the success of a Product Information System for a customer is an average minimum accessing time. If today I get a slow response time caused by a slow access time to secondary storage devices on a query for horse shoes, tomorrow I may not bother querying on an expensive stereo system, which may have a high markup.

Under criterion 2, at first glance, strategy 2 seems to merit acceptance. There are many examples on the infra-record level where the use of physical contiguity obviously makes sense. For example, in a payroll file one might store in adjacent memory locations an employee's last 12 month's wages. On the record level Ghosh (1972) stresses the importance of the consecutive retrieval property as a relation between a query set and a record set, when linear searching is used, for example along a track.

> Conceptually, the best type of file organization for a linear storage medium is the one in which the records belonging to every query in {Q} can be stored in consecutive storage locations without redundant storage of any record of {R}. If

> there exists one such organization between {Q} and {R}, then the query set{ Q } is defined to have the consecutive retrieval property (C-R property) w.r.t. the record set { R}.
>
> In a C-R organization, all the records pertinent to any query can be identified by two records, namely, the first one and the last one in the organization; hence its implementation in any practical information retrieval system is very simple.

It is in the accessing of records that one sees the limitations of strategy 2 as either a primary storage strategy or even as a supplementary one to criterion 2 for a Product Information System. Yue and Wong (1973) state the first reason for rejecting strategy 2, namely inherent access randomization.

> Typically, when a large data base system operates at a high level of multiprogramming and is near saturation, long queues are readily formed contending for access to records. Consequently, the requests serviced over many consecutive time intervals are originated from the independent users, and as far as the combined request stream for auxiliary storage is concerned, the effect of serial dependence within individuals tends to be weakened. Furthermore, for each service received by a user, the job has to go through a route of several system resources (e.g., channels and processing units) and the dispatcher of each resource may also contribute to a randomization of the resultant stream of requests.

So randomization owing to query processing will tend to nullify any advantage that one might hope to derive from the joint probabilities associated with accessing information from secondary storage. The second reason for discounting strategy 2 is that frequently one can minimize average access time by having a set of sequential accesses divided on different devices. This can reduce control unit and channel contention. The third reason is related to data on drums. Weingarten (1966) has shown that sequential accesses to a drum should be in a pattern called the Eschenbach design. Here sector addresses are accessed "in the order E, 2E, 3E, etc., where one continues to skip E sectors . . . ." The rationale for this scheme is that a drum cannot detect sector X, read sector X, and then detect and read sector X+1 without encountering a complete intervening drum revolution.

Strategy 1 is therefore the best under the assumption of random independent accesses and can be expected to produce a lower average access time than strategy 2.

When we are faced with random independent accesses, what do we do for intra-device data storage, e.g., for disks? When placing data on cylinders of an individual disk, Yue and Wong (1973) show that the best method is to place the group with the highest request probability in the central cylinder and successively less probable groups should be placed adjacently on alternate sides. Grossman and Silverman (1973) also discuss this technique. Frank (1969) suggests a similar method, but from a different point of view. If records are of variable sizes, but having the same accessing frequency, all small records should be placed on the central cylinders. Minimum average access time is approached, since references to those cylinders outside the central cylinders will be less than to the central cylinders.

Thus, when we have independent random accessing, the type 1 and, as we will point out, the type 2 RAD storage problem can be solved by ordering our records based on the access frequency distribution. This gives us a unified approach to storage structures.

A minimum average access time for a set of hierarchical devices is also affected by accessing policies. In a high I/O environment, when a number of data requests are queued awaiting the use of a RAD, the manner in which these requests are accessed can significantly influence a device's throughput and average access time. Figure 8 presents the secondary storage accessing problem as a scheduling problem.

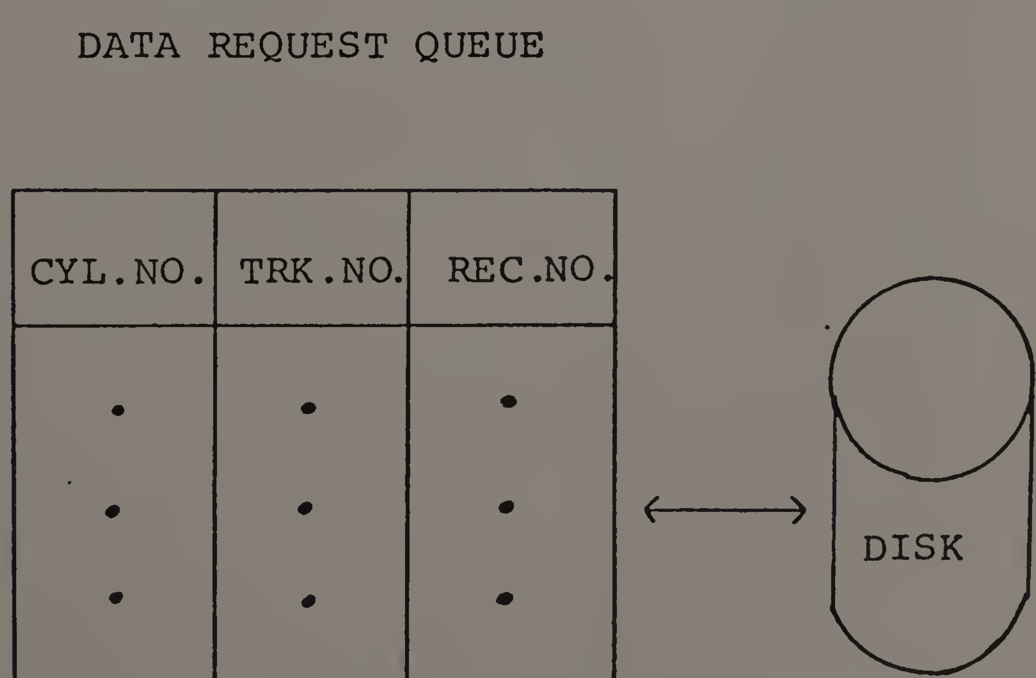


Figure 8: Queued data requests as a scheduling problem.

In this figure a data request queue feeds a disk. The scheduling problem can be stated as: given a set of data requests, order these requests so as to both minimize the mechanical access time and prevent access lockout. Access lockout is the state in which a data request remains in the request queue for an inordinate time period in relation to the other requests there.

From the point of view of a disk, this problem is one of scanning the arm back and forth across the disk. For the drum the scanning policy is directed to selecting which sectors will be accessed. A number of scanning policies have been studied for drums and disks. For drums, Denning (1967) points out that the Shortest Access Time First (SATF) policy is better than the First Come First Served(FCFS) policy. Fuller (1974) examined the Minimal Total Processing Time (MTPT) policy for drums and fixed head disks. He showed that the MTPT policy was only moderately better for drums and that occurred, if the variance in record sizes were small or the variance in waiting times were to be minimized. However, in the intra-cylinder accessing of disks the MTPT policy showed marked improvement over the Shortest Latency Time First (SLTF) policy. The degree of improvement is a function of the request arrival rate and the number of cylinders on the disk. For disks, Wilhelm (1976) has shown that under highly probable conditions an anomaly in disk scheduling exists, namely FCFS seek scheduling is superior to the Shortest Seek Time First (SSTF) policy, when the standard of comparison is the mean queue length. This anomaly exists when disk references tend to be localized, with the area of localization changing at random.

For scheduling seek accesses Denning (1967) suggested the SCAN policy. Given m cyclinders on a disk the SCAN policy moves the arm in a sweeping motion from cylinder 1 to m and back again, accessing all data requests from the queue as it sweeps. If in traveling from cylinder 1 to m or m to 1 there are no more requests in the direction that the arm is sweeping, then the direction of the arm is reversed. The SCAN policy is preferable to the SSTF policy, since the latter will produce unfavorable access lockout, especially for a file whose most frequent records are located toward the center of the disk pack or under heavy I/O loads. The SCAN policy produces only a slightly longer mean waiting time than the SSTF, but does not create access lockout as does the SSTF policy (Fuller, 1974). Gotlieb and MacEwen (1973) developed a queueing model of a movable head disk so that Denning's SCAN policy could be compared with the FCFS policy. The SCAN policy was shown to be superior to both FCFS and SSTF. Oney (1975) refined Gotlieb's queueing model of the SCAN policy, directing his attention to the mean total delay (queueing and service). His model agrees with simulation results elsewhere, but his Poisson arrival assumption must be taken into consideration before generalizing his model. It remains to be seen whether a modification to the SSTF policy, incorporating an allowable threshold to access lockout, might not prove equal or superior to the SCAN policy.

Type 2 Storage: Hierarchical Storage Devices

In developing a Product Information System, a Data Base Administrator has a diversified menu of storage devices. Random access storage devices form a hierarchy of primary and secondary devices. Figure 9 contains some representative examples:

Storage Device	Random Access Time	Equivalent Time in Nanoseconds
Cache or Buffer Store (IBM 370/155)	230 ns.	230
Main Store (IBM 370/155)	2 μs.	2,000
Large Capacity or Bulk Core Store (IBM LCS 2361)	8 μs.	8,000
Mass Store		
(IBM 2305: fixed head disk)	5 ms.	5,000,000
(IBM 3330-11; movable head disk)	30 ms.	30,000,000
(Grumman MASSTAPE)*	6 sec.	6,000,000,000

Figure 9: Accessing time in hierarchical storage.

*Access can be considered random for either the Grumman Masstape System or the IBM 3850 disk cartridge system only as to the selection of the appropriate storage device.

Storage can be dichotomized into executable and non-executable storage. Executable storage represents memory that can be directly referenced by a CPU. Since data items of a Product Information System would not be expected to reside in primary or executable memory, the Data Base Administrator would limit the scope of his storage selection principally to non-executable secondary storage. Also, since tape processing would be too slow for a real time on-line Product Information System, drums and disks would be the prime candidates for consideration.

While the advent of charge-coupled devices, bubble memories, and CRT storage will probably eventually replace the magnetic media of drums and disks, experts forecast that throughout at least the next decade it will be increaseingly difficult for a new memory technology to become cost effective to the point of displacing our current memory technologies (Computerworld, 1974; Farmer, 1974c). The next ten years will increase magnetic storage density by two orders of magnitude. Drum and disk transmission efficiency has been markedly improved due to rotational position sensing, whereby the channel and control unit are connected to the random access device only just before transmission. This frees the channel and control unit from being locked onto the storage device for the entire latency period. Radosevich (1976) points out that one must be careful in selecting job control language parameters to obtain access time gains.

Disk systems have also been recently improved by intelligent disks, analogous to intelligent terminals. Intelligent disks are controlled by controllers which take the data base access function off the CPU (Farmer, 1975). These controllers perform all indexing, searching, and deblocking operations. Intelligent disks receive commands in the form of filename, recordname, and fieldname and respond with the appropriate field value.

Performance Determination in Hierarchical Devices

Ramamoorthy (1970), one of the pioneers of memory hierarchy optimization, points out that

> The optimization of memory hierarchy involves the selection of types and sizes of memory devices such that the average access time to an information block is a minimum for a particular cost constraint.

Such optimization demands the solution to four problems, the solution to the first two of which are interrelated.

1) How much memory should be allocated to each level of a hierarchy for a given application and what devices should be chosen to implement the hierarchy?

2) What data should be stored on what part of the hierarchy?

3) Given the hardware configuration and the fact that the data are already stored there, how should the data be accessed?

4) When should data be migrated either within the hierarchy or to archival storage?

It is becoming more difficult for the Data Base Administrator to assess the relative merits of hierarchical devices in configuring a Product Information System. A few years ago drums and disks were considered easily differentiable. Today their technologies have already blended, ironically increasing the hierarchical diversity in most cases. Disks have begun to take on the characteristics of drums. Fixed head disks have similar operating characteristics to drums. The introduction in 1974 of IBM's 3348 Model 70F disk module united in individual packs both fixed and movable head technology. Even the traditional differences between tape processing and disk processing has narrowed. Grumman's MASSTAPE automatically mounts tapes. IBM's 3850 mass storage system has disk cartridges, each containing 50 million bytes, which are automatically mounted and demounted, preparatory to being read onto faster disk drives like the IBM 3330.

The determination of the number of levels to be chosen in a hierarchical configuration has been studied. Chow (1974) derived a formula for the minimum hierarchical access time and showed how the optimal number of storage levels can be determined. One of his assumptions, however, limits the generality of his results for secondary storage devices. He assumed copies of information stored on the higher levels are found in all lower levels. The storage of data on hierarchical devices has been studied for both executable and non-executable memories. In executable memories cache storage can be used as a buffer between relatively slow main memory and a fast CPU (Conte, Gibson, and Pitkowsky, 1968; Liptay, 1968). Gelenbe (1973) studied the problem of how to allocate program pages between these two storage levels. Arora and Gallo (1971), treating hierarchical levels of executable memory, presented an iterative algorithm for determining the optimal memory size of each level, given an accessing distribution. Madnick (1973) presents a number of read and store policies applicable across main and secondary storage. Williams (1973) shows that in memory hierarchies, if secondary storage devices having asymmetric read/write times are employed, multiprogramming may not be necessary.

For hierarchies not restricted to executable memories Ramamoorthy and Chandy (1970) derived a procedure for determining optimal memory sizes and the allocation of data to these memory units under a given cost constraint. These results, however, are restricted to records of equal size.

Chen (1973) extended this work on optimal memory sizes by using queueing theory to include queueing delays at various hierarchical levels. Arora and Gallo (1973) developed a model for loading data onto hierarchical devices and employed a queueing model to study the effects of varying memory sizes in an attempt to obtain a balanced system. The optimal solution is arrived at through iteration over a range of memory sizes, assuming optimal loading, as suggested by his loading rule, at every stage of iteration. Salasin (1973) showed that hierarchical memories could effectively be used to support sequential files, random accessing with a uniform distribution, or linked lists. His main assumption that copies of data are on all lower levels also limits the application of his results.

A number of performance techniques and models can be used in the study of hierarchical secondary storage devices. One can assess the performance characteristics of a hierarchical set of RADs through analytical models, simulation models, or through data collection from an actual system. Anacker and Wang (1967) present an analytical method for evaluating the performance of computer systems having hierarchical memories using the data flow rates between levels as a reference point. Shedler (1973) developed a queueing model of hierarchical storage using main memory, a drum for indexing, and a disk for data storage. Gecsei and Lukes (1974) modeled a storage hierarchy system by representing the system as a closed queueing network. This model is applicable to varying the storage configuration and/or workload. The application of this model is restricted, however, to a first-cut analysis.

RAD simulation models have also been successfully used for system performance evaluation. Nahouraii (1974) describes a software simulator, the direct access device simulator (DSIM), which can be used to evaluate hardware and software interactions. This simulator intercepts I/O instructions aimed at selected devices. The devices' hardware operations are reproduced through the use of tables.

Besides analytical and simulation models, the performance of hierarchical storage can be measured by benchmark programs run on an actual set of devices. Benchmark programs attempt to represent the workload characteristics of a system's job stream. The use of benchmarks is becoming widely used, although the standardization of benchmarks is in its infancy (Goff, 1974a; Goff, 1974b, Hillegass, 1966). Joslin and Aiken (1966) point out that it is difficult to construct a good benchmark to represent the workload. Not only is the choice of representative programs difficult, but so is the sequencing of jobs difficult to model, since jobs may enter into the system at different times and in a multiprogramming environment they can be processed differently every time they are run.

In a Product Information System the problem of constructing a benchmark would not be so much in modeling the DBMS software modules, since their content would be known before the Product Information System became operational. The problem would be in assessing what the customer load would be on the system and the accessing pattern. When these were adequately estimated, benchmarks could be profitably used for the selection of RADs.

Synthetic programs are helpful as a benchmark tool to represent a workload which can be given to vendors for the selection of new hardware or to measure untapped capacity in a system (Ciampi, 1972). Good guidelines are available for constructing synthetic programs (Sreenivasan and Kleinman, 1974; Oliver, 1974). Synthetic DBMS programs could be used in a simulated environment to select RADs, even before the DBMS modules had been completely programmed, provided the general operating characteristics of the DBMS modules and the accessing characteristics of the customers were known. Such a model of a Product Information System, driven by synthetic programs could be monitored for data base performance by both hardware monitors, which measure accessing rates and channel and device utilizations as well as by software monitors, which can record what data items or records are accessed. Software monitors are already available for measuring disk file usage (Farmer, 1974a). One should remember that in performance evaluation, hardware and software monitors are complementary in nature. Frequently both must be used in order to get an adequate picture of what is happening in a system.

Thus far we have treated the performance evaluation of hierarchical RADs under the assumption that the customer accessing distribution remains stable. While this may be true over a short period of time, this accessing distribution can be expected to change. This will obviously impact on the storage of product records. While much experimental work remains to be done in this area, some studies on data migration and reorganization are worth noting. Data migration is the movement of data from one level to another in hierarchical storage in response to changes in a data's relative access frequency distribution. Data migration is one of the design requirements for some systems presently under development (Lundell, 1973). Data migration has been studied for both primary and secondary storage. For primary storage Kaneko (1974) determined an optimal task switching policy for a set of programs operating in a CPU bound, multiprogramming environment. He assumed only the first level is executable and pages of a task migrate up a level at a time from lower levels. On the secondary storage level, Considine and Weis (1969) describe a system in which usage statistics are captured for the management of on-line data bases. Based on the criterion of when data was last used, data was migrated between on-line secondary storage and archival storage. Morgan (1974) studied the same migration problem: which data should be permanently resident and which mountable. Shneiderman (1973) presented cost tradeoffs between the cost of accessing a disorganized data base versus the cost of reorganization.

The Type 3 Storage Problem

The type 3 storage problem looks at the allocation of data over the total computerized information system. Figure 10 illustrates the general configuration and processing possibilities for the data base of a Product Information System. There are four major configurations:

1. The Centralized Dedicated Configuration
2. The Centralized Integrated Configuration
3. The Distributed Dedicated Configuration
4. The Distributed Integrated Configuration

The Centralized system creates data allocation problems of types 1 and 2. The distributed network encompasses all three storage problem types. In Figure 10 functional and

	CENTRALIZED	DISTRIBUTED
DEDICATED	-Redundant Info. -Transmission Time	-Redundant Info. -Updating
INTEGRATED	-Processing Contention -Transmission Time	-Redundant Info. -Processing Contention

Figure 10: Potential architectures for data base storage in a Product Information System.

cost problem areas are designated within the boxes. In a Centralized architecture, the secondary storage devices containing product information can be connected to a central computer, which can collect and process all queries and route answers back to the respective customers. A Centralized Product Information System can be implemented either as a dedicated or an integrated system. As a dedicated system, the main processor and the data base would be considered as resources whose time and space are allocated primarily to handling customer queries. In an integrated system, the firm's internal operations would share the same processor and/or data base. We assume a firm that would implement a Product Information System will have a large data base of inventory items. These inventory items will be needed to service both the firm's internal operations and its customers' queries. Also we assume a large customer base, accessing the Product Information System most likely over a wide geographical area.

Of these two configurations of a Centralized Product Information System, the Centralized Integrated system will perhaps prove to be the most cost effective, since data base redundancy will be minimized. A disadvantage, however, with an integrated centralized system will be processing contention with the firm's internal operations. In a Centralized Integrated system, processing contention can be lessened, if it is feasible to run heavy batch processing involving product information for internal operations during the Product Information System's non-prime times. Nevertheless, both centralized architectures suffer from high transmission costs associated with customer querying. These costs are twofold. First, there are line charges on the communication hardware. Secondly, transmission delays coupled with processing contention can increase customer response time, causing loss in expected revenue from the customer base due to customer dissatisfaction.

The Distributed Product Information System is another architectural alternative. Such a system can be implemented either as a dedicated or integrated system. In a distributed dedicated system each node contains complete query processing capability and a copy of the total product data base. A distributed integrated system, however, presents the most likely candidate for a fully developed Product Information System. Figure 11 illustrates such a system and incorporates the terminal subsystem modules of Figure 6. For clarity of illustration, Figure 11 contains explicitly the Data Communication/Data Base modules only in the main computer, but these modules will also exist in the node computers. Here each node services a geographical area and has full query processing capability, but only has the part of the data base that represents frequently accessed data. If a query enters a node and the node's data base does not contain the requisite information, the node computer forwards the access request to the firm's main computer, which contains the entire data base.

One can see from Figure 11 that the distributed integrated system is an extension of the distributed dedicated system, for the former reduces to the latter if we replicate the total data base in each node. The distributed integrated system is also an extension of both the more simple centralized dedicated and the centralized integrated system. If the data base and the node computers are removed from the nodes, then the distributed integrated system is reduced to the centralized dedicated or integrated system.

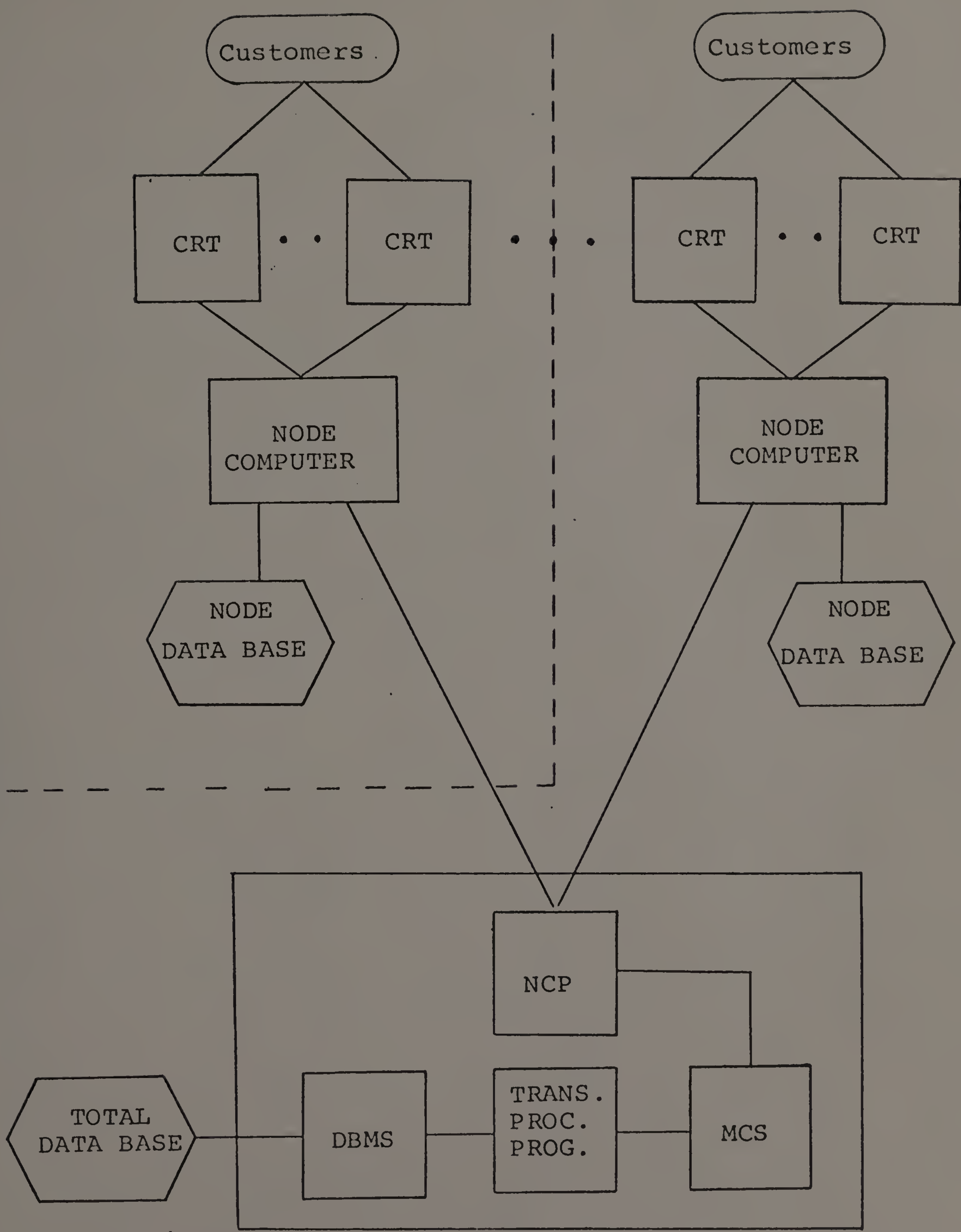


Figure 11: An Integrated Distributed Data Base System for a Product Information System.

Although each node in a distributed integrated system can be expected to contain only a portion of the entire data base, the traffic between any node and the main computer can be maintained at a low enough level to insure that the traffic will not impact too heavily on the firm's internal operations nor appreciably increase the average response time for the customer base. There are at least two reasons why the distributed integrated system recommends itself. First, since we can expect a non-uniform accessing frequency for product items, a large portion of the data base may be stored solely on the main computer. Secondly, if the traffic between the node computers and the main computer begins to increase owing to changes in the accessing patterns of the customers, the storage of product information can be reallocated between the main and node computers. In a data base environment, this reallocation can be done automatically, even if the physical structure of the data stored at the main computer is different from the data stored in the nodes. Early work by Chu (1969) on distributed data bases attacked this type 3 storage problem as a linear zero-one programming problem. Mahmoud and Riordon (1976) summarizes the work to date on optimal allocation of data bases in a distributed network and structures this type 3 storage problem as a non-linear integer programming problem, in which communications cost and storage cost form the objective function and response time delay and data reliability become the constraints.

Both the distributed integrated Product Information System and the distributed dedicated system need communication lines between the node computers and the main computer to handle the updating of product information in the nodes. The distributed integrated system may prove to be preferable over the distributed dedicated system, since the former would have less redundant data as well as less secondary storage necessary to store the data. Because of the non-uniform accessing frequency, cost savings may prove to be significant with little effect on node processing and response time for the customer base. It is with this in mind that the scope of this work confines itself to the study of CPU scheduling activity in such node computers, without an anticipated loss of generality, when applied to an entire integrated distributed system. This scope is illustrated by the dotted lines in Figure 11.

Data Base Systems and Data Base Management Systems

A Data Base remains a static entity until we link it with a Data Management System or a Data Base Management System. Naftaly, et al. (1972), point out

> The term Data Management System is often linked to computer program packages that do little more than help prepare simple reports. DMS has also meant those highly complex, all encompassing

> systems that become a way of life to their users. It has also been applied to packages within these extremes. In short, the acronym--like MIS before it--has been widely applied and misapplied.

Kelly (1970) summarizes the data management function as follows:

> The Data Management function seeks to combine, coordinate, and integrate the varying data requirements defined in diverse application areas. Data Management includes, therefore, the collection of all data required for an information system and the organization of them into a data base. This implies the physical arrangement of a hierarchy of data structures on storage devices. In turn, this function implies the need for a choice of file organization methods and associated file processing languages for handling structured data files.

Bachman (1969) differentiates data management from data base management by first pointing out that messages, programs, and data base records are all data and as such are subject to data management. The concept of a DBMS is a sub-set of the concept of a data management system. The DBMS is unique in that it imposes an interface between the data base and the end user. This screen allows the exchange of information between many people in a corporation at various levels and for various ends. In a multi-programming environment one can go a step further and say with Moreira that the function of a DBMS is to provide an interface between the user and the operating system, which deals more directly with the I/0 devices (Moreira, Pinheiro, and D'Elia, 1974).

Performance evaluation of Data Base Management Systems is in its initial stage of evolution. Snuggs, Popek, and Peterson (1974) point out that Data Base system performance studies should consider

1. Resource utilization including CPU time, channel loads, programming time, update, and maintenance time.

2. Resource utilization in terms of space for main memory and secondary storage, for application programs and Data Base Management System routines.

3. Data Base decay, the tendency for the data base to become disorganized.

4. Response time for queries.

5. Ease of use of the system.

6. User satisfaction.

All of these items are of particular interest to Product Information Systems. Checklists exist for selecting a DBMS (Prendergast, 1972). Cohen (1973) provides a detailed performance analysis of a few of the most popular DBMS. Feedback on the satisfaction of DBMS by business firms is becoming available (Gepner, 1975). Krinos (1973) showed software monitors can effectively be built into a DBMS to capture statistics from on-line systems.

In 1971 the CODASYL Data Base Task Group (DBTG) published a report containing recommendations for DBMS capabilities, which have been looked at up until recently as a generalized standard for the development of DBMS. The CODASYL Systems Committee (1971) attempted to put the DBTG proposal in perspective by comparing its DBMS features with nine other DBMS. A more current list of DBMS can be found elsewhere (Computerworld, 1975b; Leavitt, 1974).

Hare (1971) has outlined the assumptions of the DBTG in setting forth their proposal:

1. Data can be described independently of the languages which manipulate the data;

2. The users of such information systems should not be restricted to a single host language;

3. Data represents selected aspects of real operations;

4. The processing of data models selects aspects of real operations;

5. There is a close relation between selected aspects of any real operation.

Languages for DBMS can functionally be divided into three categories: those that describe the data base, those that support file generation and updating, and those that provide report generation and/or information retrieval (Cagan, 1973). Categories 2 and 3 are usually united into one processing or query language. The DBTG, however, considered categories 2 and 3 as one category. They divided the first category into two data base description languages. In all the DBTG advocated the development of three families of languages to support a DBMS. The schema Data Description Language, a single language, would allow the Data Base Administrator to describe the content and structure of the data base. Each user language that would be used to access the data base would have associated with it a subschema Data Description through which the Data Base Administrator would describe how the user's application program would view the data base. Systems like IBM's Information Management System (IMS) have a built in Data Description Language (DL/1). Senko (1975b) described a Data Description Language called FORAL, a user transaction language, which can be used for both data description and data access.

The third group of languages suggested for development by the DBTG was Data Manipulation Languages (DML). The user accesses information from the data base by issuing commands written in the DML, which are translated into calls to the DBMS. The DBMS acts as an interface between the user and the data base by associating the user's subschema structure with the data base schema, then retrieving the data, and finally presenting the data to the user in the subschema format. The DBTG report suggests a number of retrieval and updating commands. Lefkovitz (1974) lists a set of commands applicable to on-line DBMS systems.

If we consider just the retrieval aspect of DMLs, these languages can be classified into three types. First, there are Forms Controlled Languages, like MARK IV and the COGENT system, in which the user constructs his query logic by filling out a set of input forms. Next, there are languages called Procedure Oriented Languages, like General Electric's Integrated Data Store (IDS), which uses a host language like COBOL. Thirdly, there are languages called OWN DML languages, like System Development Corporation's Time-Shared Data Management System (TDMS), which functionally is similar to Procedure Oriented Languages, but use a format and set of verbs specifically developed for data management. While it should be pointed out that systems like MARK IV are primarily file management systems, nevertheless, the above classification of DMLs is as equally applicable to DBMS.

With such a plethora of possible DMLs, a user flexibility gradient exists for the developer of a Product Information System. The low end of the gradient encompasses query languages that are relatively easy to implement, yet inflexible for either the firm's representative at a CRT or the customer. At the other end of the gradient there are highly flexible query languages allowing complex

Boolean operations, synonym usage, and computer generated cueing on the data base. Interactive Query Facility (IQF) is an example of a very flexible DML for use by clerks (Martin, 1976). Along this gradient the query language designer should attempt to choose a point which maximizes the tradeoffs between the ease of learning the language, ease of implementation, and rapid execution time on the one hand and the flexibility of the language and user or clerk satisfaction on the other hand.

The ANSI/X3/SPARC Study group (1975) on DBMS proposed significant changes in the design specifications of Data Base Management Systems. This group is called the Standards Planning and Requirements Committee of the American National Standards Committee on Computers and Information Processing. It began in 1972 to investigate possible DBMS standardization. Both before and after the DBTG work of 1971 the area of DBMS has been fraught with disagreement on DBMS design in a field of continued development of non-standardized DBMS systems. Not only did the schema - subschema dichotomy of 1971 seem to have deficiencies, but the entire area of what was suitable for standardization was also left clouded after 1971. Should standards be proposed for DBMS modules or just for the interfaces between DBMS modules?

The SPARC group's interim report began by considering information systems as consisting of five basic concepts, and each concept was then designated as a "discipline." Bachman (1969) treated some of these concepts in detail elsewhere and related them to job processing.

CONCEPT	DISCIPLINE	CONTENT
Messages	Message Mgt.	I/O and data between processes.
Records	Data Base Mgt.	Fields, records, files, sets, descriptions of these, and all indices, mapping techniques, access methods, file organizations and end user languages.
Procedures	Procedure Mgt.	Program preparation, compilation, debugging, and cataloging.
Resources	Resource Mgt.	Memory allocation, task dispatching, secondary storage assignments.
Processes	Process Mgt.	Local variable handling, working storage handling.

Figure 12: Management functions in computerized information.

Figure 12 shows the relations between the scope of data base management as viewed by the ANSI group and other management functions in computerized information systems.

One cannot help but notice a narrowing of the scope of data base management. For example, this group considered as falling outside the domain of data base management the physical storage of data on secondary storage devices. "We decided that all the memory allocation problems, swapping, dispatching, and tape and disc drive assignments . . . were not part of data base management." (ANSI/X3/SPARC, 1975, p. 5).

This group also narrowed the scope of the traditional responsibilities of the Data Base Administrator. These functions they considered as being separated into at least three roles: the enterprise administrator, the data base administrator, and the application system administrator, with the enterprise administrator assuming the lion's share of the responsibility for system's design and access authorization. The enterprise administrator controls the functions of integrity and security.

It must be pointed out that the SPARC group left to the future the consideration of data base performance. This is understandable, given the enormity of their task. One, however, cannot be blinded to the fact that this narrowing of the scope of DBMS and the role of the Data Base Administrator will in the future generate problems for both system's integration and overall responsibility of information systems. When the study of data base systems evolves to the consideration of user performance problems similar to the ones we are treating in this work, the responsibility for this performance will be found to lie on many heads and across many roles. Under such a splitting of roles, it may be difficult to ascertain responsibility (culpability) when faced with measures of overall system performance like response time in a Product Information System. This is especially true, since the optimization of system performance is clearly not the sum of the suboptimizations of each individual administrator's domain.

The SPARC group substituted for the subschema - schema classification three schemas: the conceptual schema, the internal schema, and the external schema. A schema is a collection of all descriptions for an entire model. One of the strong points of the 1975 report was that it advocated making explicit the development of a model of the firm in what was termed the conceptual database schema. This schema, as a model of the firm, would be unary and would contain descriptors of the objects of interest to the firm. The conceptual schema would be under the control of the enterprise administrator. From this schema the data base administrator could develop the internal schema.

> The internal model space is the abstraction of address space in which the internal data is stored. For the purpose of the internal schema, the internal model is represented as a flat, unbounded, multi-origin, linear address space. The unit of displacement can be modeled upon such things as bits, bytes, words, internal records (internal storage records of external storage records), tracks, cylinders, volumes, etc.

> The system control data ordinarily written on a volume . . . are visible in the internal model. Performance oriented characteristics of internal data storage organization (e.g., store near, store through or see-through, multiple copies of indices or control blocks, redundant . . . copies of the same internal data are visible in the internal model. Performance oriented characteristics of external storage media (e.g., volume capacities, track lengths, latencies) are reflected in the internal data storage organization of the internal model.

The application system administrator can develop from the conceptual schema numerous external schema, one for each user. Each external schema is a partial view of the conceptual model. External schemas are analogous to the subschemas of the 1971 Data Base Task Group.

Why three schemas? The model of the firm, embodied in the conceptual schema, is the archetype for the internal and external models. While change can be expected in the conceptual model owing to business cycles, mergers, new markets, and a changing environment, this model would most likely change at a slower rate than the internal model or the external model. Technological changes will force changes in the hardware support of data base systems and thus on the internal data model. The users' need for information will rapidly change, and this change will be reflected in new external schemas, prepared for the users by the application systems administrator.

These three schemes do not introduce redundant data into the data base itself.

> While requests are transformed through successive levels of schema, data need not actually be materialized at each of these levels. For example, a complete retrieval transformation may transfer data from a disk sector to an internal storage page and from there, an external record may be prepared within a user work area. Thus, while descriptors may be traversed at each level of interface, the data itself need not be materialized at each of these levels (ANSI/X3/SPARC, 1975, p. 34).

Another advantage of the three schema approach is that all of the conceptual model need not be translated into either the internal or external model at one time or at any time for that matter. Gradual implementation of the internal and external models is afforded. Also, the model of the firm can be used in its own right, even divorced from the data base. As such, the conceptual model's schema may contain descriptors that are not intended for computerization.

> These descriptors are intended to document, to publish, and to set into context, the existence of that non-computerized conceptual data (ANSI/X3/SPARC, 1975, p. 45).

Figure 13 shows some of the relations among the three administrative roles, the three schemas, and the three mapping functions between the schemas. The mapping functions support the translation from the user's expressed requests to the physical accesses from the data base. The three schema processors generate for the administrators the

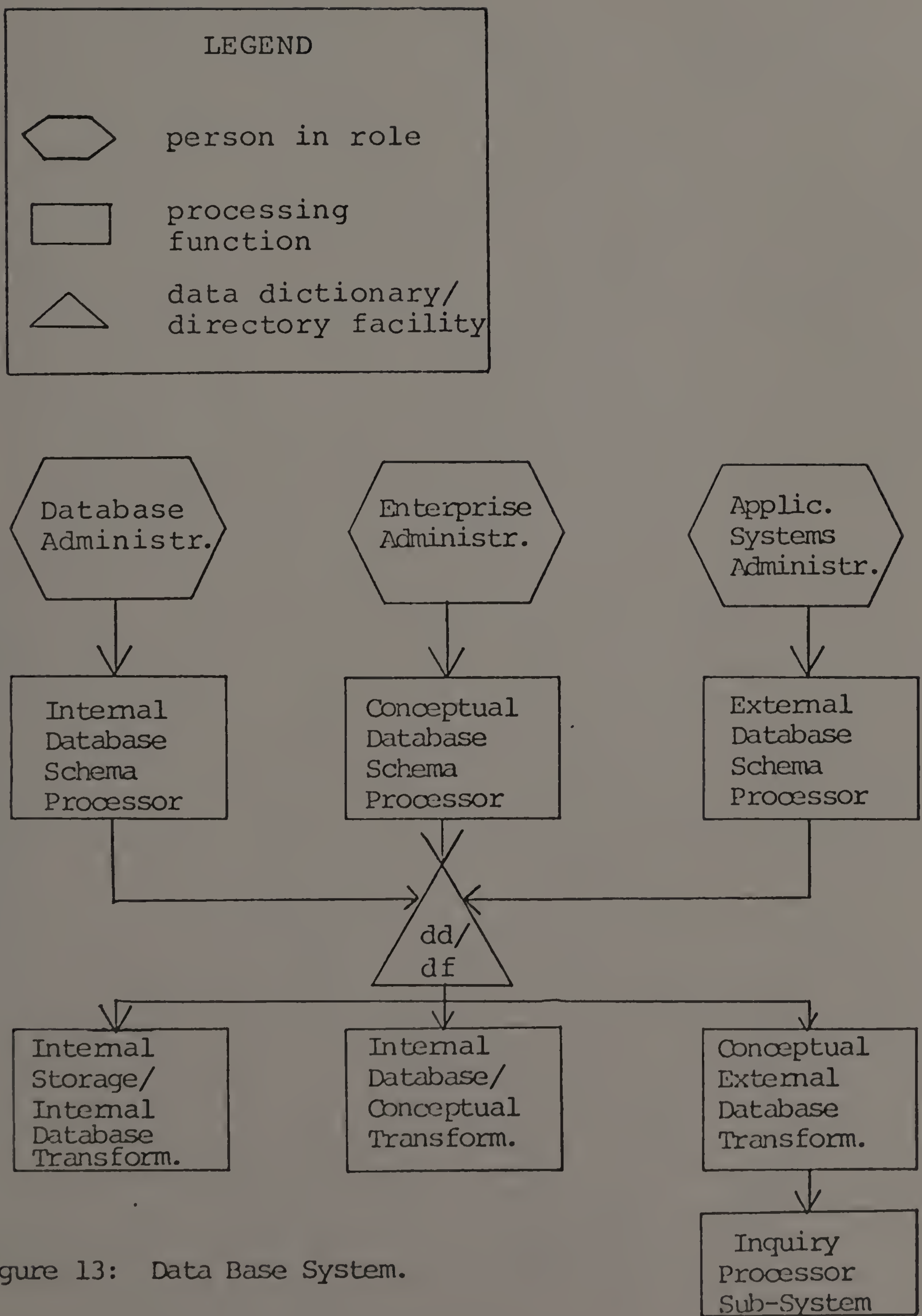


Figure 13: Data Base System.

desired schema descriptors and store these in the data dictionary/directory facility.

The data dictionary/directory facility is the principal referent used by the DBMS. This dictionary is a "metadata data base." It contains information on all three schema declarations, security, usage statistics, recovery, and restart. In addition, this dictionary contains mapping structures between the internal, conceptual, and external models.

The SPARC group attempted to outline a generalized model of a DBMS and they concentrated on the interfaces between the components of this model. Mapping from one schema to another is one of the most important interfaces. Mapping can be performed between the internal model and the conceptual model and between the conceptual model and the external models. A change in a hardware configuration necessitates only a change in the internal schema and at most in the mapping function between the internal model and the conceptual model. Here the external model and the conceptual/external mapping function would remain invariant. The presentation to a user of a new external model is implemented when the application system administrator adds the descriptors of this schema to the data dictionary/directory facility through the use of the external data base schema processor. The DBMS performs all schema processor functions as well as all mapping functions as shown in Figure 13.

We have stated that the implementation of a Product Information System should follow the design specifications of a data base system rather than a file system. In a Product Information System the inventory file(s) will be one of the main files that customers will access. In a firm that provides thousands and possibly hundreds of thousands of products, the duplication of such a set of files just for the customer base would create excessive redundancy. And yet to allow the customers to peruse everything in the inventory files would violate a firm's data security, since there would be data items in these files that a firm would consider confidential. The data base concept presents to a firm the opportunity to solve this dilemma while unifying the internal and external operations of the firm.

We now show that the design of a Product Information System as shown in Figure 2 of this chapter and the DBMS design specifications as shown in Figure 13 are compatible. Figure 14 represents the first stage of this explication. In Figure 14 the DBMS transformations have been reduced in accordance with the SPARC group's observation that

> It should be possible to prepare direct mappings between the external and internal schemas to gain efficiency of one less level of indirection when processing external requests, at the expense of the additional degrees of data independence

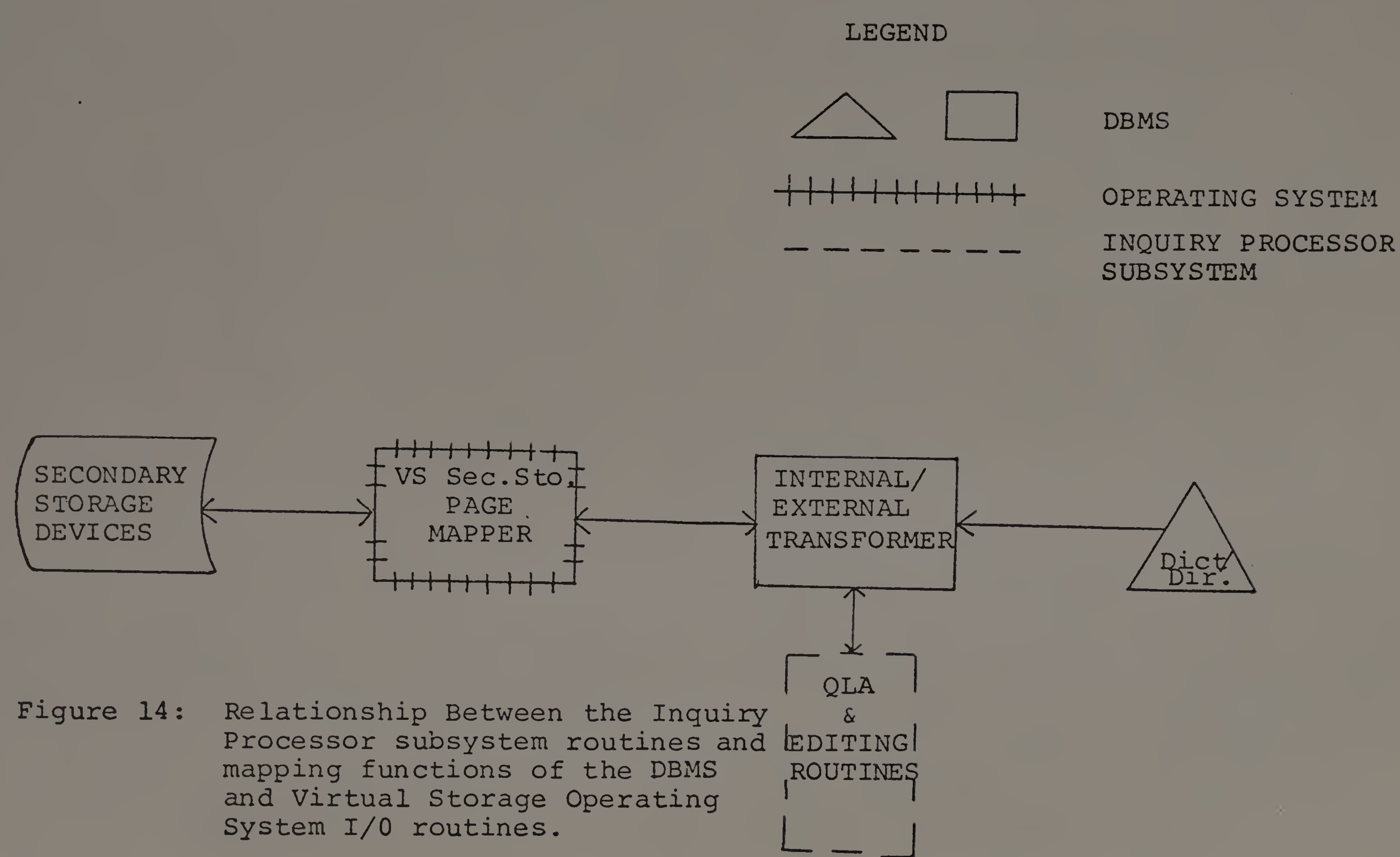


Figure 14: Relationship Between the Inquiry Processor subsystem routines and mapping functions of the DBMS and Virtual Storage Operating System I/O routines.

> provided by the conceptual schema. Conceivably, mapping processors could determine a direct External-to-Internal mapping by computing the product of an Internal-to-Conceptual mapping and a Conceptual-to-External mapping (ANSI/X3/SPARC, 1975, p. 33).

This compressed mapping is termed the Internal/External transformer in Figure 14. This DBMS mapping function in a virtual storage environment will generate data reference strings, which will then be mapped to physical secondary storage addresses by the page mapper of the operating system.

The use of the Internal/External transformer can be substantiated in a Product Information System for the following reasons. First, the customers will be using only a small part of the overall firm's informational resources and these in a very restricted area. Therefore, the conceptual model of the firm with its view of the firm in its totality is not a necessary requirement for the customer except as a mapping interface. Secondly, data security will not be jeopardized, since queries are not processed directly by a user's program but indirectly through the Inquiry Processor Subsystem. Data security can be built into either the Internal/External transformer or the Inquiry Processor subsystem or both.

One can now go one more step in reconciling Figure 2 with Figure 13. This change is merely a logical change, affecting only the way the systems designer views the

Product Information System. Physically, the routine in Figure 14 remain active and unchanged. There are three mapping functions in Figure 14: the query language analyzer, the internal/external transformer, and the virtual storage page mapper. Nunamaker, Swenson, and Whinston (1973) sum up the function of a query language analyzer as a routine or set of routines which analyze the user's query, checking for consistency in the query as well as syntax compliance. The query language analyzer may request additional information from the user.

There are two main methods of specifying access to data structures through a query language analyzer: set-oriented analyzers and graph-oriented analyzers (Senko, 1975a). In the set-oriented languages the user specifies the subset of the data he wishes to access and the system determines how to obtain and format the information. In the graph-oriented data accessing languages the user specifies the access path and the tests to be performed on each node. The DBTG and IBM's Information Management System (IMS) are two examples of the graph method of accessing through a query language.

The virtual storage operating system mapping function translates page reference numbers generated by the internal/external transformer into secondary storage addresses. The internal/external transformer and the query language analyzer have in common the fact that they both map logical structures. The designer of the mapping interfaces of a Product Information System therefore can consider the logical mapping functions of the internal/external transformer and those of the query language analyzer as a unit. He can consider the internal/external transformer as an extension of the query language analyzer. This is analogous to the application programmer who, when using a data manipulation language, considers GET statements as extensions of his COBOL language. In this work we will consider the internal/external transformer and the query language analyzer as a unit, termed solely the query language analyzer.

The Inquiry Processor Subsystem also contains editing routines, which present a formatted reply to the customer through the CRT of the firm's representative. These routines accept as input the output from the internal/external transformer. The DBMS routines and related routines that we are most interested in for the study of CPU scheduling in a data base environment using virtual storage are therefore the query language analyzer, the operating system's page mapper, and the editing routines, which are depicted in Figure 2.

CHAPTER III

CPU SCHEDULING ALGORITHMS AND THE EXPERIMENTAL DESIGN

In this chapter we consider in detail the three CPU scheduling algorithms and present the model of a Product Information System as set forth in Chapter 2 as a test bed for experimentation with the three CPU scheduling disciplines. In essence, the three CPU scheduling disciplines will be our subjects and the model of the Product Information System will be our experimental environment. We will begin by explaining in more detail each of the three CPU scheduling disciplines, their objective, their functioning, their overhead, and their tradeoffs, when measured by average query response time and average potential sales loss. Next, we will show how the Product Information System model is an experimental environment, delineating the dependent and independent variables and the values of the independent variables under which we will study the performance of these disciplines. We will then state hypotheses concerning the behavior of these disciplines in a Product Information System. Our hypotheses will be related to how each CPU scheduling discipline will perform relative to the others under selected conditions, when measured by response time and potential sales loss.

We will then explain why simulation is an adequate tool for testing these hypotheses. Then we will demonstrate how accuracy is insured at each stage of model design and use. This includes the selection of appropriate statistical tests. Finally, we will present the compile time and running characteristics of the simulation model.

In the next chapter we will consider the experimental results, substantiating our thesis that the Data Base Administrator, as the person who is responsible for the performance of a Product Information System, must seriously consider the selection of an appropriate CPU scheduling algorithm. This thesis will be shown to be true based on the experimental results showing that the relative performance of each of the three scheduling disciplines, when measured by either response time or potential sales loss, will be dependent upon such Product Information System operating conditions as CPU utilization, secondary storage loading, and main memory sufficiency. Therefore, the DBA cannot *a priori* choose one of these disciplines for all processing conditions, and we present the system conditions under which each discipline will perform best relative to others.

Three CPU Scheduling Disciplines

In Chapter 2 we introduced the three CPU scheduling disciplines under consideration in this work, giving a rationalization for their use a priori to experimentation. We now present them in more detail, point out the processing overhead associated with each, and discuss the tradeoffs inherent in each discipline. As stated in Chapter 2, the CPU is fed from two queues: one the CPU pre-emptive queue; the other, the CPU DBMS queue, which is not pre-emptive. Let the pre-emptive queue be designated queue A and the non-pre-emptive one queue B. The CPU, upon completion of the processing of a routine or upon a program page fault, will process the next routine to be executed from queue A, unless queue A is empty. If and only if queue A is empty, the highest priority routine from queue B is selected. In the event both queues A and B are empty, the CPU will enter the wait state.

Pre-emption occurs at any time a routine from queue B is executing on the CPU and another routine enters queue A. The routine from queue A gains control of the CPU and the other routine is returned to the head of queue B, retaining highest priority. This feeding rule will be applicable to all three CPU scheduling disciplines. The three disciplines differ only in the way they set their priorities within queue B. Each discipline, besides setting priorities and linking each routine together within queue B, will be responsible for keeping the address of the highest priority routine updated for easy access by the CPU. Since each node computer is dedicated to data base operations, we can ascribe to the disciplines most of the overhead in switching from one routine to another. Obviously, provisions would have to be made for specialized pre-emptive routines for recovery on machine or program malfunction. But the effects of these are considered minimal in this work. We assume the hardware will have a high "up" time and that the software has been adequately pretested.

Before we look at each CPU scheduling discipline in detail, it will be helpful to consider Figure 1. This figure gives a list of all of the principal routines expected to run on a node computer in processing queries. Figure 1 contains each routine that will enter either queue A or queue B, gives its name, whether it belongs to the DBMS or the Operating system, and describes its function. In addition, for clarity this figure under the heading "ref. no." refers the reader back to the diagram numbers of Figure 2 in Chapter 2. In Figure 1 of this chapter the heading designated "type" of routine refers the reader forward to Appendix 3, which contains the SIMSCRIPT model of a Product Information System. The SIMSCRIPT model uses the "type" attribute to distinguish between the different kinds of routines, since each has its own characteristics and belongs to either queue A or B. The coded values {1, 2, 3, 4, 5, 6} are pre-emptive routines, while {10, 11, 12} are non-pre-emptive. It is a moot

Type	Routine Name	Ref. No. (Figure 2, Chapter 2)	Pre-emptive	CPU DBMS Queue	DBMS or Op.Sys.	Function
1	CRT Input Interrupt Handler	15	Yes	A	Op.Sys.	Brings Query into main memory.
2	CPU Scheduling Discipline	19	Yes	A	Op.Sys.	Places QLA into queue B.
3	CPU Scheduling Discipline	19	Yes	A	Op.Sys.	Returns a routine to queue B after a demanded program page has entered main memory.
4	Page Fault Handler	15	Yes	A	Op.Sys.	Places page request in I/O queue.
5	CPU Scheduling Discipline	19	Yes	A	Op.Sys.	Places editing routine into queue B.
6	CRT Output Interrupt Handler	15	Yes	A	Op.Sys.	Handles output from DBMS output queue to CRT.
10	Query Language Analyzer	6	No	B	DBMS	Translates incoming queries.
11	Page Directory Look Up	7	No	B	Op.Sys.	Maps to physical RAD addresses and places page references in RAD subsystem queues.
12	DBMS Editing	12	No	B	DBMS	Edits data pages into a CRT response format.

Figure 1: Data Base Management System and Operating System Routines.

argument whether the page directory lookup routine should be included as a pre-emptive routine or a non-pre-emptive routine. We include it as a candidate for queue B', recognizing, however, that one may argue that it may be better to consider it as a pre-emptive routine because of its small size. Figure 1 contains only a single scheduling discipline but it is listed three times. This is to differentiate the various stages in the processing of a query. The same discipline comes into play a number of times in the process of a query.

The routines in Figure 1 are classified into sets: pre-emptive and non-pre-emptive. They are logically divided, such that the routines processing directly a query are considered non-pre-emptive, while those servicing the query indirectly are pre-emptive. Under direct query service are the QLA, the mapping, and editing functions as shown in Figure 14 of Chapter 2. The routines providing indirect service functions handle the inputting of queries, the outputting of answers after editing, as well as page faults, and CPU scheduling. It is important to point out that there are inherent dependency relationships between some of these routines. For example, a query analyzer routine, whose processing object is a query, is itself a processing object to both a CPU scheduling discipline and the page fault handler. We have, therefore, routines which, while servicing queries, are in turn being serviced by other routines. This, of course, presents an interesting queueing environment, since the processing rate of the two largest non-pre-emptive routines (the QLA and the editing routines) depend on how quickly service can be rendered by some of the pre-emptive routines. Another example of servicing a server is when a program page of the editing routine enters main memory owing to demand paging, the editing routine remains in the wait state until the CPU scheduling discipline activates it by putting it into queue B. Delays caused either in the start up of the execution of the CPU scheduling routine or by the length of time needed for the CPU scheduler to link the editing routine into queue B, will necessarily affect query processing time. It is for these reasons that the indirect service functions are considered pre-emptive.

In order to understand how each of the disciplines functions, we first describe how the routine "types" in Figure 1 can be scheduled. Each routine in Figure 1 must be scheduled for CPU operation either by a CPU scheduling discipline or by a master control routine of the operating system. Otherwise, we have a problem of who schedules the scheduler ad infinitum. In an interrupt-driven machine, a small master control routine, as part of the operating system kernel, can automatically be given control of the machine when interrupts occur. Assuming that all frequently occurring operating system routines are resident in main storage, this small controlling routine then activates the necessary operating system routines by placing them in queue A. The master control routine places routine types 1, 2, 3, 4, 5, and 6 into queue A in a FIFO manner. The CPU scheduling discipline schedules the query processing routine types 10, 11, and 12. Note that the CPU scheduling discipline is itself scheduled. This is not a contradiction nor a violation of Ockham's razor, since the CPU scheduling routine is here viewed as controlling only queue B, the routines that most directly process queries. The master control routine performs three functions. First, it places routines in queue A. Since this is always a FIFO queue, this can be implemented in a few machine language instructions. Secondly, it sets up the CPU to select either queue A or B. This is done by controlling a single memory location as a pointer. Thirdly, in rare instances it calls in appropriate system crash or program error recovery routines. The overhead for the master control routine is minimal. And it itself does not enter any queue or experience any delay. Therefore, this routine has not been included in the simulation model.

In processing queries, there is main memory management overhead. Workspaces must be allocated. Space for tables and buffers must be provided. The overhead associated with memory management, although an operating system function, will be allocated to the routine actually needing the memory. For example, if an editing routine needs main memory to perform a sort on data retrieved from secondary storage, the editing routine is viewed as generating a call to a memory management routine, which provides the necessary space, if available. The overhead of the called routine is allocated to the calling routine.

We now consider the structure of the Query Language Analyzer, the page directory lookup routine, and the editing routines (routine types 10, 11 and 12 in Figure 1) in order to appreciate better how the disciplines actually handle priority setting and the linking of these routines into queue B. Routines 10, 11, and 12 should be structured as shared routines. In a Product Information System many users are simultaneously presenting queries. It would be extremely inefficient with one hundred customers querying the data base of a node computer to provide one hundred images of the same query language analyzer, editing routines, etc. This would be data redundancy in the extreme. The use of shared routines obviates this kind of redundancy. While a copy of all routines would be expected to reside on secondary storage, only a single copy of each routine at most would be materialized in main memory. Each physical copy could be shared concurrently by all queries. Each physical routine is viewed as a set of virtual routines.* For example, assume for illustration only that the QLA is one routine. It physically exists once in main memory. Yet, the QLA can be viewed as a set of virtual routines, each virtual routine assigned to a single query and each being an image of the physical routine. Since each query is unique for the data that it manipulates, each query would have its own set of tables and its own workspace, while sharing the same query

---

*A shared routine is at the same time physically one and logically multiple. We will reserve the term "shared routine" to refer to a routine's physical oneness. We introduce the term "virtual routine" to connote one of the logical images of the single shared routine. Without this distinction, confusion would arise when one wishes to differentiate between an individual routine being put into a wait state due to data references to the RAD subsystem and a physical routine being placed in a wait state due to a program page fault. As an example, assume the page directory lookup routine has just finished executing for a query. The next instruction to be executed for direct processing of the query will be the first instruction of the editing routine. However, since the <u>data</u> pages referenced by the page directory lookup routine may not yet be in memory, the execution of the editing routine is held in abeyance until the last data page has been read into main memory from the RAD subsystem. In essence, as far as this query is concerned, its own virtual editing routine is in the wait state. However, the virtual editing routines of other queries may progress. Things are different, however, when ina virtual storage environment, a <u>program</u> page fault occurs. Here the physical page of a routine has been found not to exist in main memory. It must be brought into main memory and, until it has been, all virtual routines about to execute this physical page must be put into the wait state. The use of the term virtual when applied to routines should not be confused with the use of the term when applied to storage management or the ability of the operating software to mimic other machines (virtual storage and virtual machines).

processing routine code. Base registers or indirect addressing schemes can be employed so that operand addresses of a virtual routine point to unique addresses.

The use of reentrant code or what is called "pure" code is necessary, if one wishes to implement shared routines. Machine language code is reentrant if no instruction modifies another instruction or itself. Reentrant code insures that every time a routine is run the instructions are the same. The use of reentrant code is necessary not because there is only one physical copy of each routine, but because of the concurrency of query processing. All queries are processed concurrently not only in terms of the different types of routines, but also within any given routine.

Figures 2 and 3 illustrate this concurrency, which increases system parallelism without any added cost of redundancy. In Figure 2 we have a partial map of main memory in which is shown a query table, single query work-areas, and the three direct query processing routines, which are shared by all the queries. For illustration, let us assume the QLA routine is located at address 100,000 in main memory. The page directory lookup routine is located at address 200,000 and the editing routine is located at address 300,000. At a given point in time the editing routine is processing queries 1 and 2, while the query language analyzer is

QUERY TABLE

Query No.	Routine Type	Next Address	Work Area
1	12	350,000	50,000
2	12	300,000	55,000
3	10	126,584	60,000
4	10	164,432	65,000
5	10	100,000	70,000

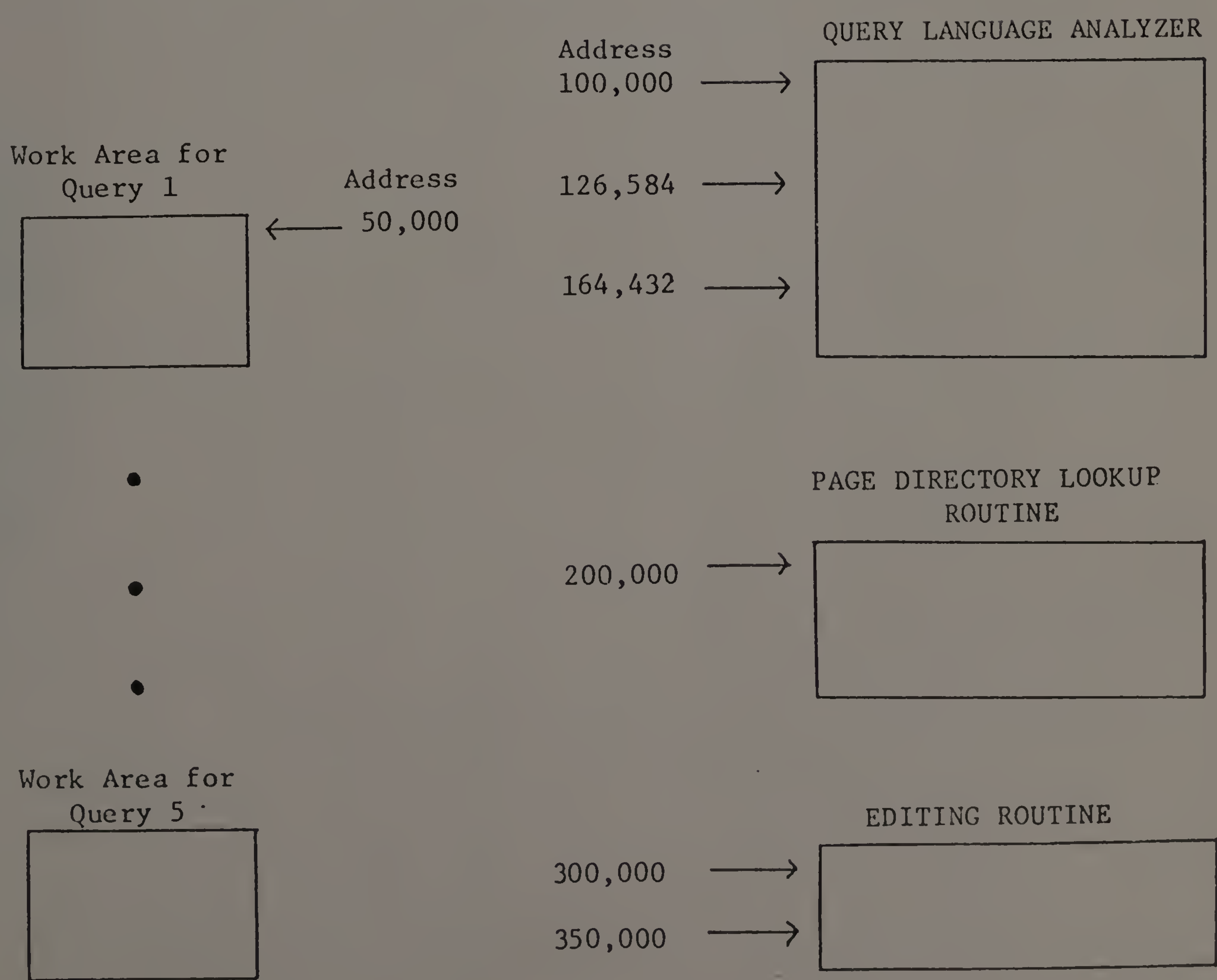
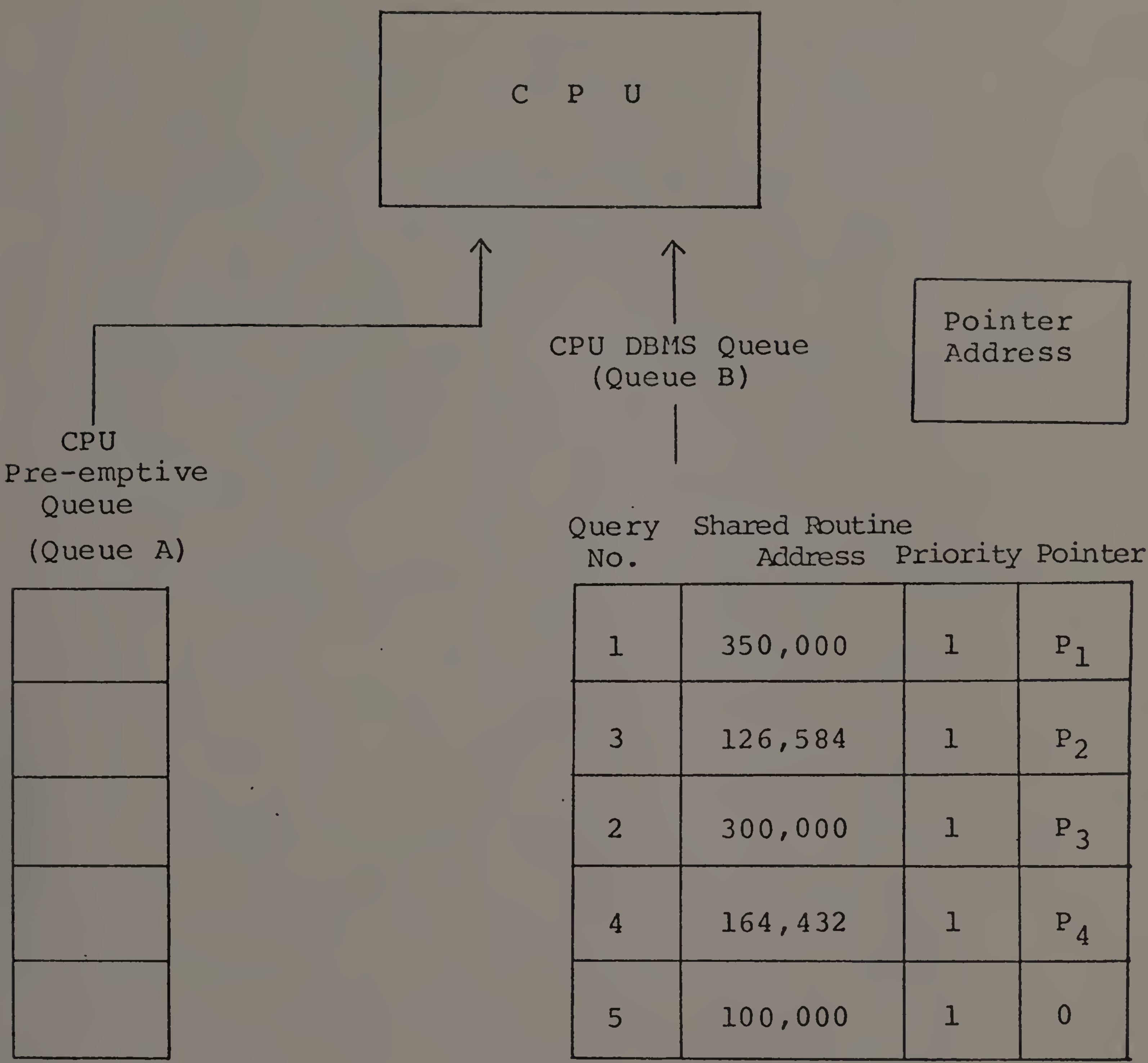


Figure 2: Shared Routines.

concurrently processing queries 3, 4, and 5. The query table specifies for each query being processed, first a unique query number; secondly, the routine type under execution for this query at this point in time; thirdly, the next address to be executed within the shared routine; and, fourthly, the address of the current workarea associated with the query. One notices that the complexity of any query may cause some queries to progress through a shared routine faster than others. In Figure 2 this is illustrated by the fact that, if all five queries entered the system at the same time, queries 3, 4, and 5 are lagging behind queries 1 and 2. Also, query 4 is further into the QLA than query 3.

The address of the next instruction to be executed for a query is maintained not only in a query table but also in the CPU DBMS queue (queue B) for scheduling purposes. This is shown in Figure 3. Figure 3 corresponds to the two queues, feeding the CPU in Figure 2 of Chapter 2. These two queues, called Ready queues, are awaiting CPU processing. In Figure 3 we are concerned with the CPU DBMS queue. This queue is composed of set members, each pointing to the next address to be executed within a shared routine. In its simplest form each member of the queue contains the associated query number, the next address to be executed within a shared routine, a priority setting, and a pointer to the next member of the queue. The CPU DBMS queue is maintained by pointers rather than by



Query No.	Shared Routine Address	Priority	Pointer
1	350,000	1	P_1
3	126,584	1	P_2
2	300,000	1	P_3
4	164,432	1	P_4
5	100,000	1	0

Legend

P = Pointer to next member of chain
0 = End of chaim

Figure 3: CPU DBMS queue as a set of virtual routines.

a table, since the number of members in this queue and their priorities will change dynamically. Figure 3 shows the implementation of queue B for use by a FIFO scheduler, e.g., discipline 1, and in this example will not be subject to changes in priority.

We now consider at what points during the time a query is processed a CPU discipline performs its priority setting function and the linking of a virtual routine into queue B. On the surface the answer seems quite straightforward. The CPU discipline should be activated whenever a virtual routine is created or whenever an existing virtual routine can be taken out of the wait state to rejoin the ready queue. When measuring the relative performance between CPU scheduling disciplines the problem is not when the scheduling occurs, but the fact that over the lifetime of a query each of the virtual routines may be scheduled a number of times before completion. This is especially true in a virtual storage environment, where a routine may undergo repeated program page faults.*

A model showing the points at which CPU scheduling activity is employed is shown in the Routine Usage Flowchart

---

*The term virtual is used here in the traditional sense of a paging system irrespective of shared routines.

of Figure 4.* One can most easily comprehend the pattern of scheduling in Figure 4 if one considers the stages that a query goes through in being processed, once the query has entered the DBMS input queue. We begin by considering the CPU scheduling routine running on the CPU. It sets the priority on the Query Language Analyzer routine and links it into queue B. This sets up the QLA to begin processing the query. After the QLA routine reaches the head of queue B, the QLA runs on the CPU until completion, if the entire physical copy of the QLA has been made resident in main memory. In a virtual storage environment with limited main memory space, the pages making up the QLA would be susceptible to page faults. The most frequently needed pages would be paged in and out as required. In a virtual storage environment, when a QLA page fault occurs while the QLA routine is executing, the virtual QLA is inactivated by being put in the wait state, while the page fault is handled by the RAD subsystem. When the QLA page has been brought into main memory, the CPU scheduler will again run on the

---

*We introduce the use of a Routine Usage Flowchart. Its primary purpose is to show the relationship between routines and the resources they use. The Flowchart reads from left to right, top to bottom along a time horizon. Each segment of the diagram is composed of an upper level and a lower level. The lower level, designated by rectangles, represents resources used by the upper level. The resources depicted in Figure 4 are the CPU and the secondary storage devices. The upper level represents what or why these resources are utilized.

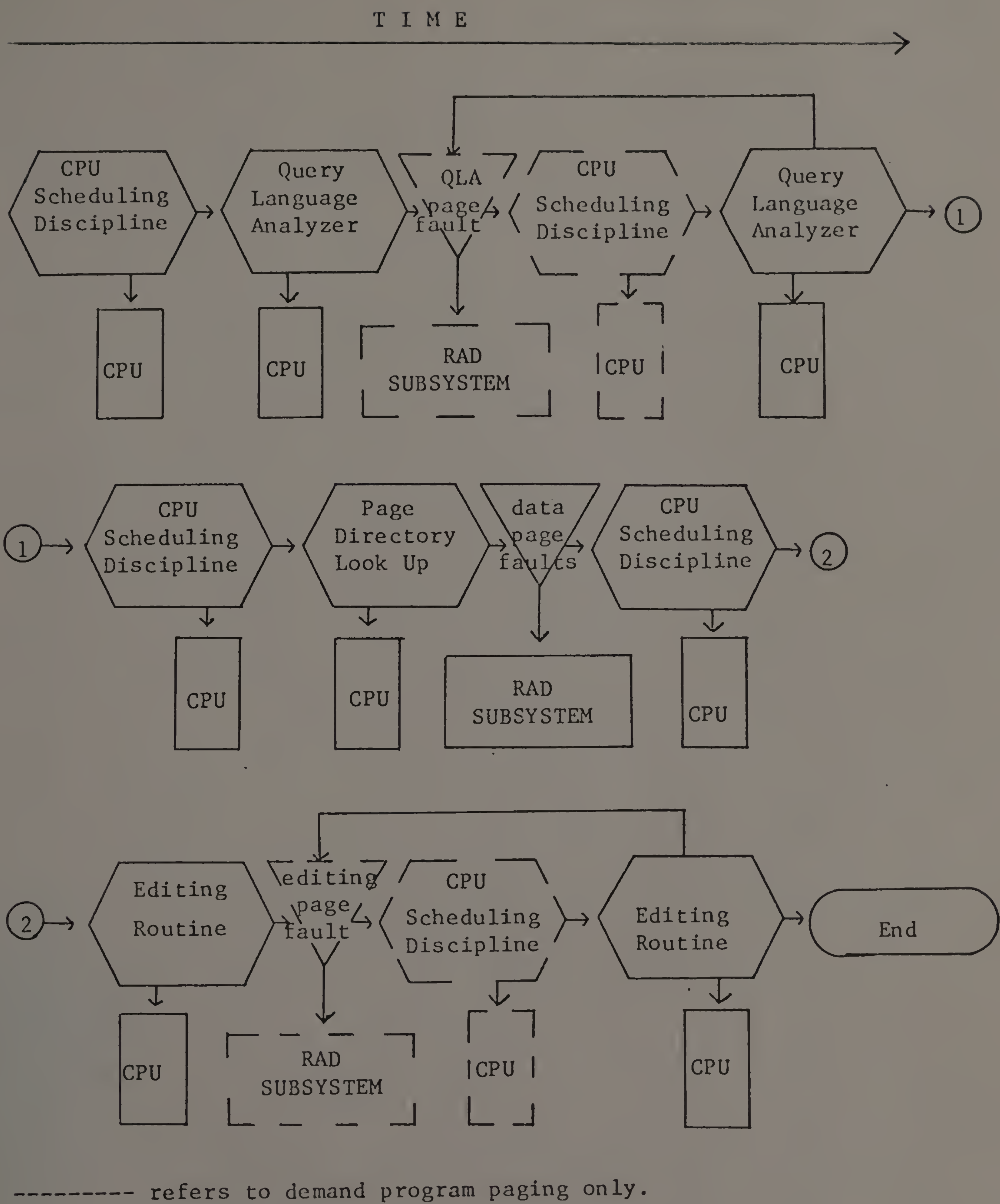


Figure 4: Routine Usage Flowchart.

CPU to reschedule the virtual QLA and link it again into queue B. The query then continues to be processed when the virtual QLA regains control of the CPU. A loop is formed in the Routine Usage Flowchart as the query is processed intermittently by the QLA between program page faults. At the termination of the QLA, the CPU scheduling discipline is run on the CPU to schedule and link to queue B the page directory lookup routine. This routine will usually be too short to undergo a program page fault. However, the page directory lookup routine will create data page references as output. These references reflect the needed data to be retrieved from secondary storage as the basis for the answer to the query. After the last data page has entered main memory, the CPU scheduling discipline can then be activated to schedule and link into queue B the editing routine. The editing routine will process and format the "raw" data brought into main memory. In a virtual storage environment there will also be a processing loop for the editing routine. The editing routine will execute intermittently as did the QLA. The result will be an answer to a query, which can be sent back to a CRT.

With the structure of the query processing routines as shared routines in mind as well as both the structure of the CPU queues and how and when the CPU scheduling disciplines are called into play, we now consider the priority setting activity of each of the CPU scheduling disciplines. How does each set its priorities for queue B? How does it link members into this queue and what is the overhead involved in terms of the number of instructions necessary to perform these functions?

CPU Scheduling Discipline 1: FIFO

The first CPU scheduling discipline which we consider for a Product Information System is the first-in-first-out (FIFO) discipline. We will refer to this as discipline 1. It will be considered for its own merits as well as being considered as a standard against which we can compare the other two disciplines. It has the smallest overhead and is the easiest to implement. Within queue B, discipline 1 schedules routines for the CPU solely on the basis of arrival time, i.e., the time at which a routine is put into the CPU DBMS queue. The overhead for discipline 1 to perform its priority handling is approximately ten machine language instructions, when the address of the last member of the queue is readily accessible. If queue B is empty, the placement of a routine into queue B is even simpler, since the pointer address of a previous member of the set does not have to be modified. Under discipline 1, if we view just queue B, priority handling is purely FIFO. If we view both queue A and queue B as a system feeding the CPU, then our CPU priority handling is a modified FIFO, modified by pre-emption of the routines in queue A. Therefore, when discipline 1 is used in our simulation model, we will refer to this as a modified FIFO discipline.

CPU Scheduling Discipline 2: Reducing Variation

The second CPU scheduling discipline which we consider for a Product Information System is a response time variation reduction discipline. The system response time has been defined as the time interval between the end of the customer's formulation of his query and the beginning of the answer as flashed on a CRT. System response time represents the dead time in the conversation between the firm's representative and the customer. We have noted in Chapter 2 that this time interval is most sensitive in a Product Information System. We have also shown in Figure 1B of the Introduction that r_1 represents the part of the response time that would be tolerable for all of the customer base. In Appendix 1 we have shown that if lost potential sales is the performance measure, variance reduction in response time could possibly lower the average potential sales loss. Discipline 2 attempts to take advantage of both the existence of r_1 and the sigmoid shape of the loss curve. The goal of discipline 2 is to continually equalize system response times as closely as possible to the perceived mean system response time by giving higher priorities to queries that are lagging behind, temporarily sacrificing those with a shorter forecasted system response time.

At selected mileposts within a query's lifetime within the computer, measurements of accumulated system response time can be taken. At each milepost these measurements can be used as a basis for comparing an individual query's "journey time" through the system against the average accumulated system response time over all queries to arrive at that milepost. This comparison can be used to set priorities on the routines processing queries. Appropriate mileposts would be the creation and termination times of the three direct query processing routines: the Query Language Analyzer, the directory lookup routine, and the editing routine. The creation time mileposts are appropriate, since immediately after a virtual routine has been created to service a query it must be scheduled into queue B. By considering also as mileposts the termination of each of these routines the system can calculate the average elapsed time for the execution of each type of virtual routine. When a virtual routine is prepared to re-enter queue B after a program page fault, the priority of this routine can be again set using the expected amount of remaining CPU time needed for routine completion along with the accumulated response time for that query.*

---

*See routine INSERT.QUEUE in Appendix 3.

The overhead for a discipline 2 scheduler is the sum of the four functions that it performs. The first function for discipline 2 is that it must capture the beginning of the system response time interval for each query. Given an estimated delay time for the terminal subsystem and a time stamp on when the query entered the DBMS input queue, this function can be implemented in less than twenty instructions.

The second function is the calculation of the statistics of average start and termination times for each of the three direct query processing routines. This function can be implemented using four counters per physical query processing routine. For example, assign four counters to the QLA. Let the counters be C_1, C_2, C_3, and C_4. Let C_1 contain the milepost start time measured from the beginning of the "dead" time in the conversation between the customer and the firm's representative and extending to the time at which the routine is to be first put into the CPU queue. Let C_2 be termination time. Let C_3 be the number of times the routine started and C_4 be the number of times the routine terminated. Whenever the QLA is activated for a query for the first time, the accumulated system response time of the query is added to C_1. C_3 is incremented by 1. C_2 and C_4 can be handled similarly for routine terminations. Using these counters one can calculate, as needed, the average accumulated system response time for a milepost by simply dividing C_i by C_{i+2} where i = 1 or 2. The number of machine language statements needed to implement this function is approximately twenty.

The third function calculates the query's priority and thus sets the priority on the associated virtual routine processing the query. The priority is set using the current time and the two functions mentioned above. It takes approximately four hundred and fifty instructions to implement this function. Appendix 3 presents a detailed description of this function under the simulation routine called INSERT.QUEUE.

The fourth function performed by discipline 2 is the positioning of a virtual routine into its correct position within queue B. The amount of overhead here is composed of a constant factor and a variable factor. The constant factor taking ten to thirty instructions insures that main memory space is available for the new member of queue B, places the member in a free memory location, and updates the computer's free and allocated space tables. The variable factor involves searching the link path of the queue for the proper logical insertion point. The length of this processing depends on the length of queue B. On the average one half of the queue will have to be searched to find the insertion point into the queue. Five machine language instructions suffice to compare the priority value of the routine to be inserted with a given member of the queue. The calculation of overhead for this function therefore is:

$$30 + 5 \times (\tfrac{1}{2} \text{ queue size})$$

The third CPU scheduling discipline we shall consider for a Product Information System has as its prime objective increased secondary storage utilization. Secondary storage utilization can be increased by setting high priorities on the routines that will generate secondary storage accessing, when RAD queues are empty. Routines can contribute to increased secondary storage accessing either by producing data references to secondary storage or by themselves undergoing program page faults. The page directory lookup routine is the prime contributor to data references to the RAD subsystem. The QLA and the editing routines can produce program page faults. Increased secondary storage utilization increases system parallelism and can reduce mean system response time.

As each routine is scheduled by this discipline, the scheduler makes a judgment whether RAD utilization can be increased. If RAD utilization cannot be increased, this discipline defaults to discipline 2. Thus discipline 3 is an extension of discipline 2. The criterion used in making this judgment is whether the RAD subsystem has empty device queues. A well-balanced RAD system will indicate a state of non-saturation when any one of the RAD device queues is empty. In selecting this criterion we take advantage of the fact that there is no profit to be gained by choosing to send additional page requests to the RAD subsystem if this subsystem is already "full," since the additional page requests would only remain dormant in the RAD queues. The lack of utility expressed in the army dictum "hurry up and wait" is readily applicable to the situation of actively attempting to add pages to non-empty RAD queues.

One would expect discipline 3 to increase RAD utilization as long as the RAD subsystem was not continually saturated. One would also expect discipline 3's effectiveness to be determined by its ability to predict the state of the RAD subsystem for a routine before that routine begins executing on the CPU. It is important to note that there is a time lag between when a routine's priority is set and when it begins executing on the CPU and when it begins to deliver requests for pages to the RAD subsystem. Discipline 3's effectiveness may begin to fall off when either queue B is long or the paging per query is high.* For example, routine

---

*In a computer system experiencing a high CPU utilization a significant change of state can occur during this lag period for two reasons. First, under high CPU utilization one might expect longer lag times. The CPU queues are in general longer. The second reason is what we term "routine entrapment," which is discussed at length in Chapter 4. Briefly, it is the phenomenon in which a routine, once scheduled, gets "lost" in a queue because many subsequent schedulings have a higher priority.

r is to be scheduled at time t_0 and placed in queue B. Assume at t_0 the RAD subsystem has empty queues. Let t_1 be the time for the scheduling discipline to finish on the CPU and the time in the queue for r be t_2. Let r's time on the CPU before creating access references to secondary storage be t_3. Let t_4 be the total pre-emptive time taken by routines interrupting r once it gets on the CPU. Now we state that discipline 3 will be an effective predictor of a non-saturated RAD subsystem if

$$\sum_{i=1}^{4} t_i < t_s$$

where t_s is the average time needed to significantly change the state of the RAD queues from non-saturated to saturated. A change of state would be considered significant, if at t_0 increased RAD utilization were realizable, but not realizable at the time r began to create page references to secondary storage.

The queue lengths on the RAD subsystem devices cannot increase in size except as a result of CPU processing. Therefore, t_s is a function of the number of potential accesses already in the CPU queues, the speed of the CPU, the state of the RAD subsystem at t_0, and the service rate of the RAD subsystem taken as a unit.

In determining the state of the RAD subsystem just prior to scheduling a routine, discipline 3 takes no consideration for the degree of RAD subsystem saturation. This could have been built into the discipline, but only at an additional overhead. This overhead would not be expected to increase priority setting accuracy, since one cannot readily determine at scheduling time exactly how many secondary storage references will be generated by the routines already in queue B. Also, for the page directory lookup routines one cannot determine exactly how many page references they will generate. A wide variation may exist. Yet, one can determine probabilistically when a routine's first reference to secondary storage will occur.* Discipline 3 therefore begins by making a binary decision: whether the RAD subsystem is saturated or not. If the RAD subsystem is unsaturated,

---

*In a Product Information System one cannot predetermine at scheduling time the number of data references to secondary storage that will be generated when the page directory lookup routine is run on the CPU. One could do this, if he is willing to make the assumption that there is a one-to-one correspondence in the number of secondary storage references between the output of the query language analyzer and the output of the page directory lookup routine. Such an assumption is not necessarily warranted, especially when one demands data independence between the logical level and the physical level in a data base system.

References to secondary storage can occur either by program page faults or by not having in memory needed data pages. The Query Language Analyzer and the editing routine encounter program page faults and both types of routines have the priority setting is determined by how rapidly the routine once it executes on the CPU will produce its first secondary storage reference (for a program or data page) or reach completion. The page directory lookup routine is very much smaller than the QLA and the editing routine and it also produces the most secondary storage references. So, the page directory lookup routine can potentially contribute the most to increased RAD utilization. Therefore, discipline 3 would favor this routine. To accomplish this, discipline 3 considers the amount of time a routine will spend on the CPU once it gets control, irrespective of pre-emptions from queue A. Discipline 3 follows the SXFS (shortest-execution-first-served) discipline, if and only if the RAD subsystem is unsaturated (Hellerman and Conroy, 1975).

---

their most frequently used program pages resident in main memory. Since the systems designers of a Product Information System will know which pages are assigned as resident to main memory and which are not, the demand paging characteristics of these routines can be determined by software monitors. One can determine the average number of instructions before a program page fault occurs by constructing an address reference pattern (Hatfield, 1972).

Some of the address references will be invariant under different queries, while others must be determined probabilistically. Thus, such an address reference pattern will be basically probabilistic in nature.

When the number of machine language instructions executed for the Query Language Analyzer and the editing routine range from 32K to one megabyte using 4K pages, there is an upper limit of only 250 pages/query routine to consider in constructing an address reference pattern.

Having discussed the priority handling of discipline 3 we can now consider how discipline 3 maintains its priority rankings in the face of the possibilities of saturation and non-saturation. Discipline 3 maintains two logical CPU DBMS queues (two non-pre-emptive, B-type queues), since it can default to discipline 2. These queues share the same physical members. The primary logical queue is maintained by priority settings based on SXFS. The shorter this time the higher the priority. Priority ties are broken based on earliest arrival to the queue. The secondary logical queue is similar to the queue structure for discipline 2. Figure 5 illustrates the implementation of these two logical queues as one physical structure. Each entity in this single physical queue has six values. The first value is the query number. The second value is the next address to be executed for this query within one of the direct query processing routines. The third value is the priority setting under discipline 2 control (under default). The fourth value is a pointer ordering the queue as in discipline 2. The fifth value is the priority setting based on attempted increased parallelism. The sixth value is a pointer ordering the queue for increased RAD utilization. One can see from Figure 5 that indeed discipline 3 is an extension of discipline 2.

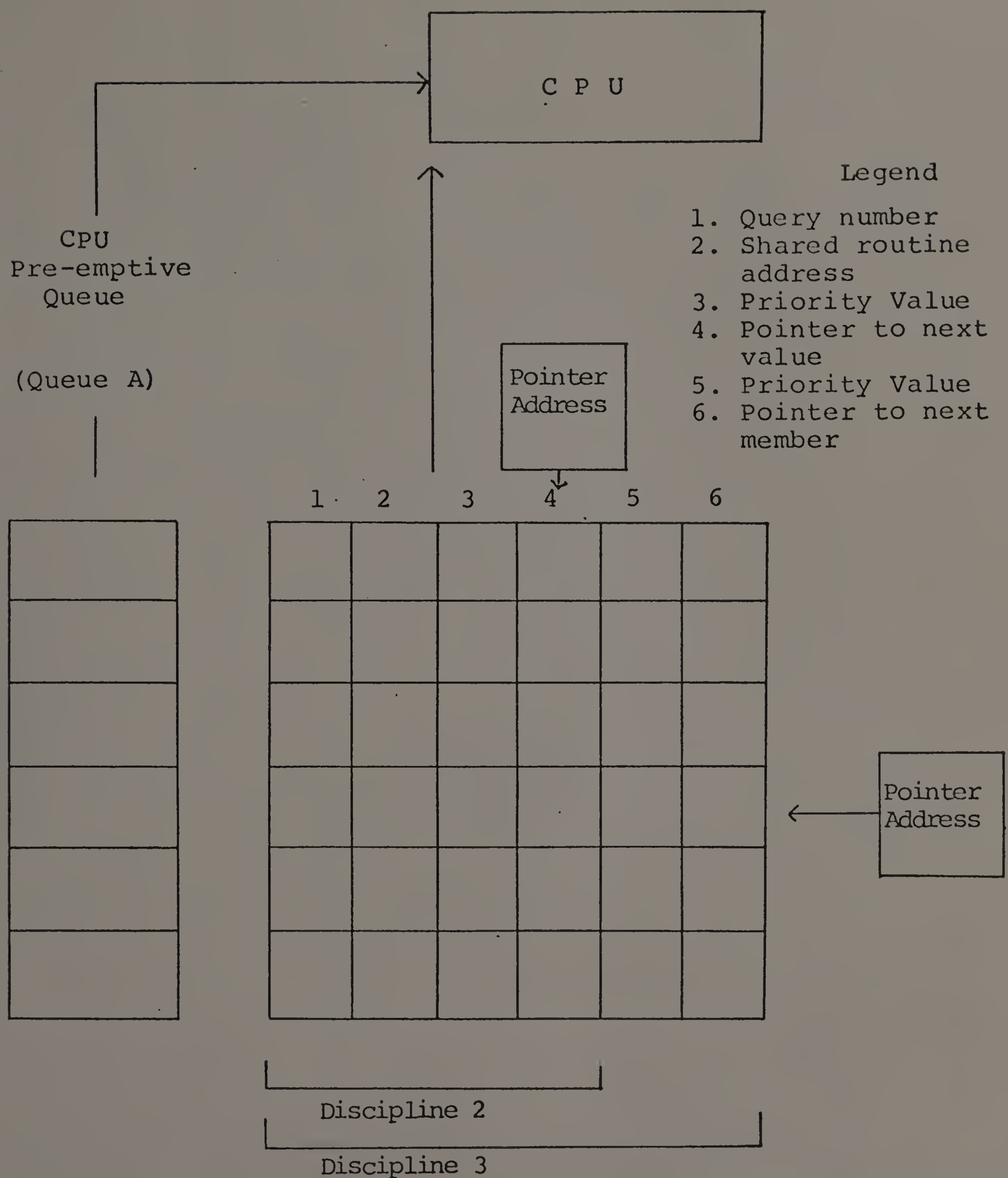


Figure 5: Example of CPU DBMS Queue Structure under Discipline 3.

When a new member is to be added to the CPU DBMS queue discipline 3 assigns a priority to the member, physically adds the new member to an available memory location, records the fact that this memory location is now occupied, and links this member into both of the logical queues by adjusting appropriate pointers. Since discipline 3 is an extension of discipline 2, its overhead contains all the functions of discipline 2 plus two unique functions. The first function determines the average number of instructions that a routine will execute before encountering a program page fault and the CPU time needed for a routine to finish processing. To do this the scheduler uses a Page Blocking Table.* Approximately ten instructions suffice for this function. The second function unique to discipline 3 is the linking of this new member into its appropriate place within the primary logical queue. The amount of overhead here is the same as for the pointer adjusting function of discipline 2.

---

*A Page Blocking Table has one entry per page number. Each entry is composed of a page number, a value for the average number of instructions before a program page fault, and the average number of instructions still needed to complete the routine. The address reference pattern supplies the necessary information for the average number of instructions before a program page fault.

When the average time to the next page fault is to be determined for a routine, the scheduler examines the appropriate entry in the page blocking table. To find this entry in the table, a binary search is performed. At this entry the priority setting for the routine is based on the

CPU Scheduling Tradeoffs

There are inherent tradeoffs for each of the three CPU scheduling disciplines, which warrant their being tested experimentally. Before experimenting one cannot tell which of the three CPU schedulers would be best for a Product Information System, when measured by mean response time or by potential sales loss. In this work we investigate the performance of each of these disciplines under representative ranges of Product Information System operation. Typical conditions that must be investigated include conditions of low and high CPU utilization, low and high RAD utilization, and different main memory capacities.

CPU scheduling discipline 1 has the advantage of a low overhead, but it also has a low discriminatory value. It is insensitive to both system conditions and the reaction of the customers to the response time. Discipline 2 has a higher overhead, but attempts to counteract this overhead by reducing the variation in the response time. Discipline 3 has an even greater overhead but attempts to offset this by

---

smaller of the two averages in the table. Because of loops and branches within any program page the actual number of instructions executed per page will most likely differ from the page size. Also, more than one page may execute in series before a page fault occurs.

The page blocking table may be updated from time to time based on historical information derived from query processing.

increased system parallelism. Discipline 3 is also the most flexible of these disciplines, being able to default to discipline 2, when it is not expected to be able to increase RAD utilization. In a virtual storage environment in which main memory capacity is not sufficient to contain all pages of the QLA and the editing routines, the effect of these tradeoffs on response time and potential sales loss becomes even more complex. When a query processing routine undergoes repeated program page faults, it undergoes repeated CPU schedulings as well. These can increase the sensitivity of the discipline, but again at the expense of additional overhead for each rescheduling. Besides these intrinsic tradeoffs one cannot be certain before experimentation that discipline 2 will actually generate a smaller variation. As we will demonstrate in Chapter 4, there are system conditions under which discipline 2 will produce results diametrically opposed to what it was designed to do. Under certain conditions discipline 2 will produce counterintuitive behavior; it will increase the response time variation.

Dependent and Independent Variables

Having discussed the operational characteristics and the tradeoffs inherent in the three disciplines, our experimental subjects, we now consider the dependent and independent variables which exist in the Product Information System environment in which these algorithms will function. The subsystems within each of the node computers have been presented in Figure 1 of Chapter 2. The routing of queries through the subsystems has been shown in Figure 2 of Chapter 2. Figure 11 of Chapter 2 has presented the entire Product Information System as a distributed integrated data base system and has related the scope of our study, the node computers, to the total Product Information System.

The dependent variables can be classified as primary and secondary. The primary dependent variables are the ones used to judge the performance of each of the disciplines; the system response time, and the potential sales loss. The secondary dependent variables include a large number of variables whose values represent system states that result from the interplay between the disciplines and the subsystems. These system states, when studied, present information on how each discipline affects different parts of the Product Information System. From these system states we can determine why a particular discipline performed better or worse than the others. These secondary dependent variables can be subdivided according to the subsystems to which they pertain.

Between the customer base and the terminal subsystem is a telephone queue; it represents the number of customers placed on "hold" because of the temporary unavailability of representatives manning the terminal subsystem. The number of customers in this queue, a dependent variable, will be a function of the customer arrival rate, the number of CRTs, the speed of the representatives, and the system response time. The secondary dependent variables in the operating system include the length of the CPU queues. For a CPU operating at a fixed service rate the length of these queues can indicate bottlenecks in query processing.

The secondary dependent variables in the RAD subsystem include input state variables, processing state variables, and output state variables. The input variables include the overall arrival rate of page requests to each channel and each device. The processing state variables include channel and device utilizations. The output variables include the overall paging rate as well as the number of pages delivered to main storage per second by each device-channel pair.

The independent variables can also be classified into primary and secondary variables. Primary independent variables include CPU utilization, the page load on the RAD subsystem, and the main memory capacity. CPU utilization is measured as the amount of time the CPU is busy divided by the total system running time. The page load on the RAD subsystem can be measured in terms of the average number of page references per query, but paging load should also consider characteristics of the RAD subsystem servicing these requests; the number and speed of the secondary storage devices as well as their channel connections. RAD device and channel utilizations are frequently used to measure paging loads. But neither are pure measures of paging load. For example, given rotational position sensing, a high device utilization can be misleading because wasted revolutions in attempted connections to a channel can give the appearance of useful accessing being accomplished.* Unless one is careful to mask out this time, <u>device</u> utilization may be more a function of how many devices are connected to a channel than the accessing load on the system. <u>Channel</u> utilization is also a function of the number of devices connected to a given channel. What we wish as a measure of paging load is a measure of the backlog of page requests waiting in the secondary storage device queues for service. Since we assume the paging load is spread over the hierarchical devices so that the devices are balanced in that the device utilizations for the secondary storage devices are approximately equal, we elect to use as a measure of the

---

*When a large number of devices is connected to a channel having rotational position sensing in a high I/O environment, access improvement may be seriously impaired. Once a device is disconnected from its channel at the beginning of a seek, it must be reconnected for reading during a short time interval just prior to reaching the desired data. If this "window" does not find the channel free, a complete revolution of the device is wasted before another attempt can be made. It may take many revolutions, therefore, before the device can be reconnected to the channel.

paging load the average number of page requests per device in the lowest (slowest) of the hierarchical levels. In the simulation runs the number of such devices varies from three to six, depending upon the configuration modeled.

Main memory capacity is measured as the average number of program page faults per routine for routines which are direct query processing routines. This relative measure of main memory capacity is more appropriate than an absolute specification.

These primary independent variables form a three dimensional set of system environments in which we can study the behavior of the three CPU disciplines. This is shown in Figure 6. Each point in this environmental space represents a single potential environment for a Product Information System. We will refer to this set of system environments as the system space.

Our objective is to see how each of the disciplines performs relative to each other, at various points in the system space. The objective is not to see how the disciplines perform under every possible point in the system space, for obviously there are an infinite number of such points. Also, a Product Information System will not remain at a single point in system space. The complexities of the queries, the size of the processing routines, and the load placed on the Product Information System by the customer

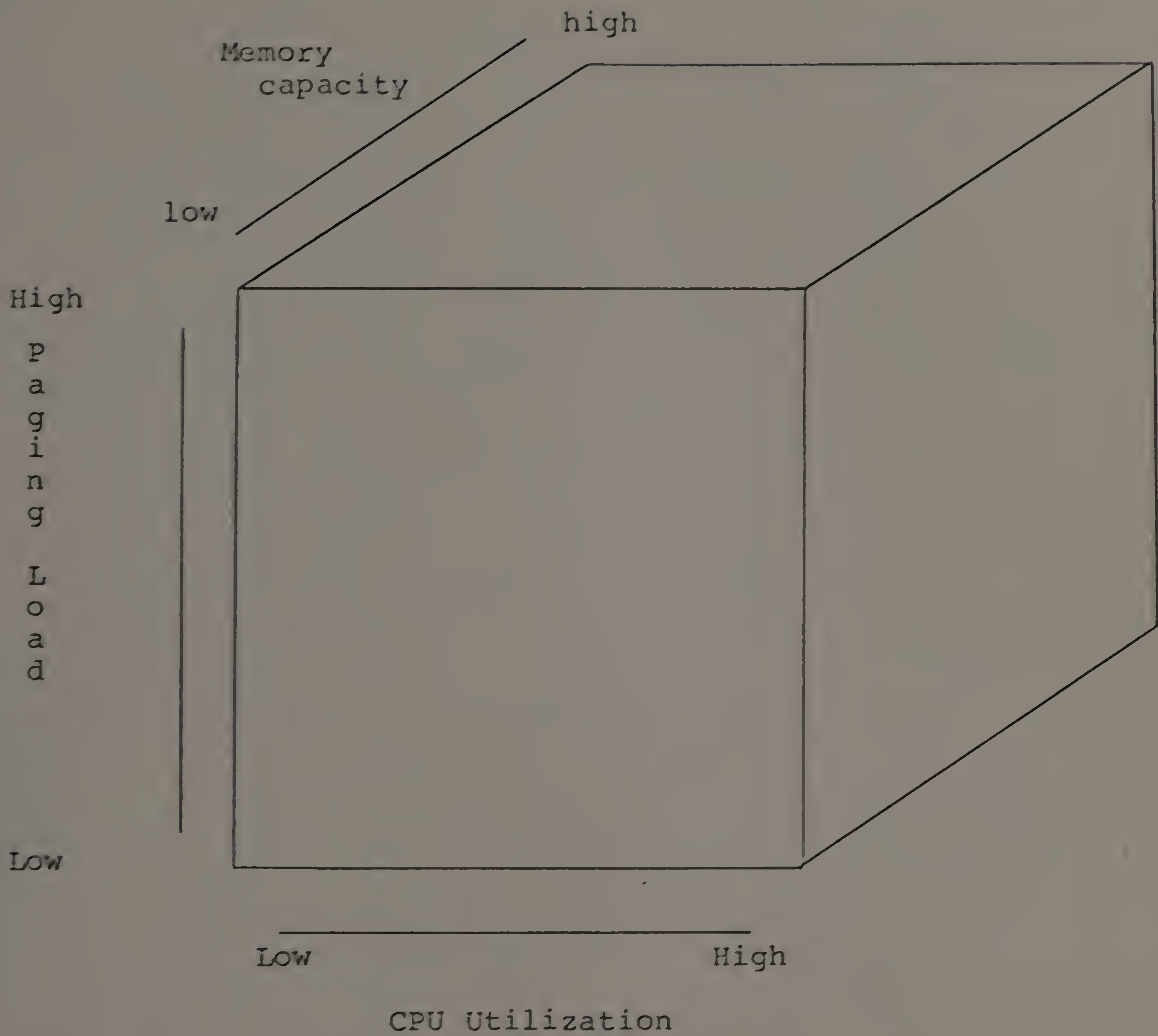


Figure 6: Independent Variables of the System Space.

base will affect CPU utilization, RAD utilization, and the amount of paging. The time of day, the day of the week, the time of the year, and general business conditions all affect the load placed upon a Product Information System by the customer base. As a result, any Product Information System will be dynamic in the sense that it will be able to be represented over time as a cluster of points in the system space. The exact specification of such a cluster will be an individual phenomenon to each Product Information System. That is, each Product Information System developed will have its own center to its cluster and a unique amount of dispersion around that cluster in system space.

Within the system space we will also consider the extreme points along each axis. In the design of computer systems frequently the extreme values of system variables are more critical to good system performance than centroids. An example of this can be seen by considering the fact that many time sharing system specifications stress peak load conditions. The reason for considering extremes (both lows and highs) in systems design is twofold. First, in a system experiencing a wide dispersion, the centroid may not represent the system's behavior at all. Secondly, points at the extremes of system space can produce unique or magnified system behavior. A system problem that exists at or nearby a centroid may not exist at the low end of one of the dimensions in system space. For example, under low CPU utilization, congestion in the CPU queues as a system problem may not exist. Under heavy system resource demands, the behavior of performance variables frequently becomes non-linear, even exponential in shape. At the extremes a small change in an independent variable may produce a magnified effect on the performance variable. An example of this is the sharp increase in system response time as a function of the increased number of users in a time sharing system (Hellerman, 1975, p. 139). Parachor curves, which show the total number of page interrupts a program encounters as a function of the amount of main memory available for holding pages, also demonstrate a high degree of sensitivity at the small memory size end of the curve (Madnick and Donovan, 1974, p. 163). After a certain point, the total number of interrupts rises sharply for a small decrease in memory size.*

---

*One should keep in mind that the running time of routines in a paging environment is not always a monotonically decreasing function of the size of main memory. Belady states and demonstrates that "With certain real-life programs the running time is reduced by decreasing the space in which the program runs." This counterintuitive behavior is due to the fact that page replacements made under FIFO replacement can cause cyclic patterns. These patterns can produce inefficiencies by increasing the page fault rate. To obviate this problem a LRU (least recently used) page replacement algorithm is recommended. (L.A. Belady, R.A. Nelson, and G.S. Shedler, 1969).

Besides primary independent variables a Product Information System contains a number of secondary independent variables. These variables are considered secondary, since one can set the primary independent variables by tuning these variables. For example, one can increase CPU utilization by increasing either the size of the QLA or the size of the editing routine, or the arrival rate of queries to the Product Information System. The RAD subsystem utilization can be increased by increasing the average number of page references per query or by changing the types of RAD devices or channel connections. Figure 7 lists the secondary independent variables most important to a Product Information System and lists representative values, ranges of values, and settings which will be used in this work. In Figure 7 the variables are grouped by subsystem. Also included are the variables applicable to the mainframe of a node computer. Within each subsystem the name of the variable, its representative values, ranges, and settings are given.

In some sense one may almost consider the primary independent variables as if they were dependent variables, since the setting of values to the secondary independent variables affects the primary independent variables. Yet, these primary variables are truly independent for our experimental purposes, since we are only using the

Figure 7: Secondary Independent Variables, Their Values, Ranges, and Settings

Secondary Independent Variables	Representative Values, Ranges, and Settings
1. Customer Base Subsystem	
a. arrival rate of calls	Peak conditions
b. no. of queries/transaction	1
c. no. of pages/query	mean = 10 - 50 (expon.)
2. Terminal Subsystem	
a. greet time	mean = 4 secs. (expon.)
b. think time	mean = 20 secs. (expon.)
c. typing time	mean = 10 secs. (expon.)
d. rep. responding time	mean = 20 secs.; s.d.=5 secs.
e. no. of CRTs	100
f. telecommunication time	1 sec.
3. DBMS subsystem	
a. QLA instructions	mean = 32,000 - 1,000,000 s.d. = 10,000 - 333,000
b. Editing instructions per page processed	mean = 5,000 - 40,000 s.d. = 1,700 - 13,000
4. Operating System	
a. Page directory	75
b. I/O instructions	1,000
c. CPU sched. instructions	(see text)
5. Mainframe	
a. no. of CPUs	1
b. CPU speed	1,000,000 instr./sec.
c. memory capacity	0 to 3 faults/routine
d. page size	4K
6. RAD subsystem	
a. no. of channels	1 - 2
b. device types	
- seek time	0.030 - 0.075 sec.
- rotation speed	0.010 - 0.025 sec.

Figure 7: continued

Secondary Independent Variables	Representative Values, Ranges, and Settings
- pages/track	1 - 3
- no. of cylinders	200 - 400
- RPS	on drum and fast disks only
c. device - channel link	approx. evenly balanced
d. distribution of data across devices	balanced
e. distribution of data across cylinders	binomial

secondary independent variables to set the environmental primary conditions for experimentation.

One hundred CRTs per node is reasonable as a modeling parameter. In terms of orders of magnitude it is difficult to see how ten CRTs per node would be economically justified, while one thousand CRTs per node would generate too much traffic for a medium-sized computer and would be too much for a single geographical area. We choose peak conditions for arrival rates at these CRTs such that a telephone call comes into the system at the termination of a previous call. If we consider that the data base may contain in the neighborhood of one-half million catalog items as do some of our large chain stores, this would necessitate four to ten present day secondary storage devices (random access devices) to contain this information. The exact number would depend on how much information resided in each node and how much resided in the common computer facility alone. This of course would be dynamic, changing in response to communication line charges, device costs, and the customer access frequency pattern.

As was pointed out in Chapter 2, most of the variables in the terminal subsystem have been shown in the literature to exhibit exponential or gamma distributions. We have used exponential distributions in the terminal subsystem of our model of a Product Information System, with the exception of the representative's responding time for which there is no evidence to contradict the assumption of a normal distribution. The mean values for the variables in the terminal subsystem were determined by role playing the activity of a representative, assuming the use of trained personnel. The number of instructions needed to run the query language analyzer and the editing routine on each query not only has a wide variation within any given set of queries, but, as has been pointed out in Chapter 2, there will be a wide range of sizes among the different QLAs and editing routines of different Product Information Systems. These routines will grow in size as the query language becomes more flexible and more desired output forms are incorporated into the system. Because QLAs have similarities with compilers and interpreters, a low end estimate of this range for QLAs is 32,000 bytes. The high end of the range is dictated more by mean response time than by the possible intrinsic complexity of the QLA. Designers of Query Language Analyzers will tend to desire to include more language features than can be handled in a timely fashion by the node computers. The limiting factor to the unlimited development of QLAs is and will be the CPU speed in the node computers. For a QLA executing one million instructions per query under one hundred CRTs, with a medium-sized computer being able to execute one million instructions per second, the system is well beyond saturation. CPU utilization reaches 1 and the response time becomes unbearable.

The page size is set at 4K, a size commonly used today in industry and government. The parameters for the device types are those for currently available drums and disks that are used in computer systems. The faster disks have rotational position sensing. The data constituting the data base is considered as being distributed across the secondary storage devices in such a way as to produce equal RAD device utilizations.* The data is placed on individual disk packs binomially with the most frequently accessed data at the center of the disk pack. This model does not include any SCAN policy for data retrieval, although it could easily be added to the model. These policies are not widely implemented at present, nor can one be assured that a Product Information System would use them.

---

*Buzen states that balancing the load on secondary storage devices may not give optimum throughput. Faster devices should be given higher utilizations, thus creating bottlenecks at the faster devices. Buzen points to Chen's Rule of Thumb for determining the optimum utilization levels. One adjusts the utilization levels so that the fraction of time each device is idle is inversely proportional to the square root of the speed of the device. (Jeffrey Buzen, 1976).

This refinement is not incorporated into our model for two reasons. First, it is not in use in today's systems and one cannot forecast when its incorporation into future systems will become widespread. Secondly, there are no known estimates of the overhead involved in continually adjusting the data base to produce this added throughput.

Hypotheses

The hypotheses on the relative behavior of the three disciplines that we present for experimentation are drawn from the system space of Figure 6 and the primary dependent performance variables: the system response time and the potential sales loss. We are interested in testing to see if significant differences in these performance variables can be shown to exist for the three CPU scheduling disciplines at selected points within the system space. Taken together the experiments form a sensitivity analysis for the three disciplines within the system space.

We formulate a set of null hypotheses: two hypotheses for each selected point in system space. The first null hypothesis (A) states no difference in system response time means among the three disciplines. The second null hypothesis (B) states no difference in the means of the potential sales loss among the three disciplines.

Each hypothesis, although stated as a single null hypothesis, is in reality three subhypotheses. The first subhypothesis states no difference in means between disciplines 1 and 2. The second subhypothesis states no difference in means between disciplines 1 and 3. The third subhypothesis states no difference in means between disciplines 2 and 3. Figure 8 illustrates the testing of these

<u>Legend</u>

Sign. = significant at 0.05 level
n.s. = not significant
D_i = ith CPU scheduling Discipline

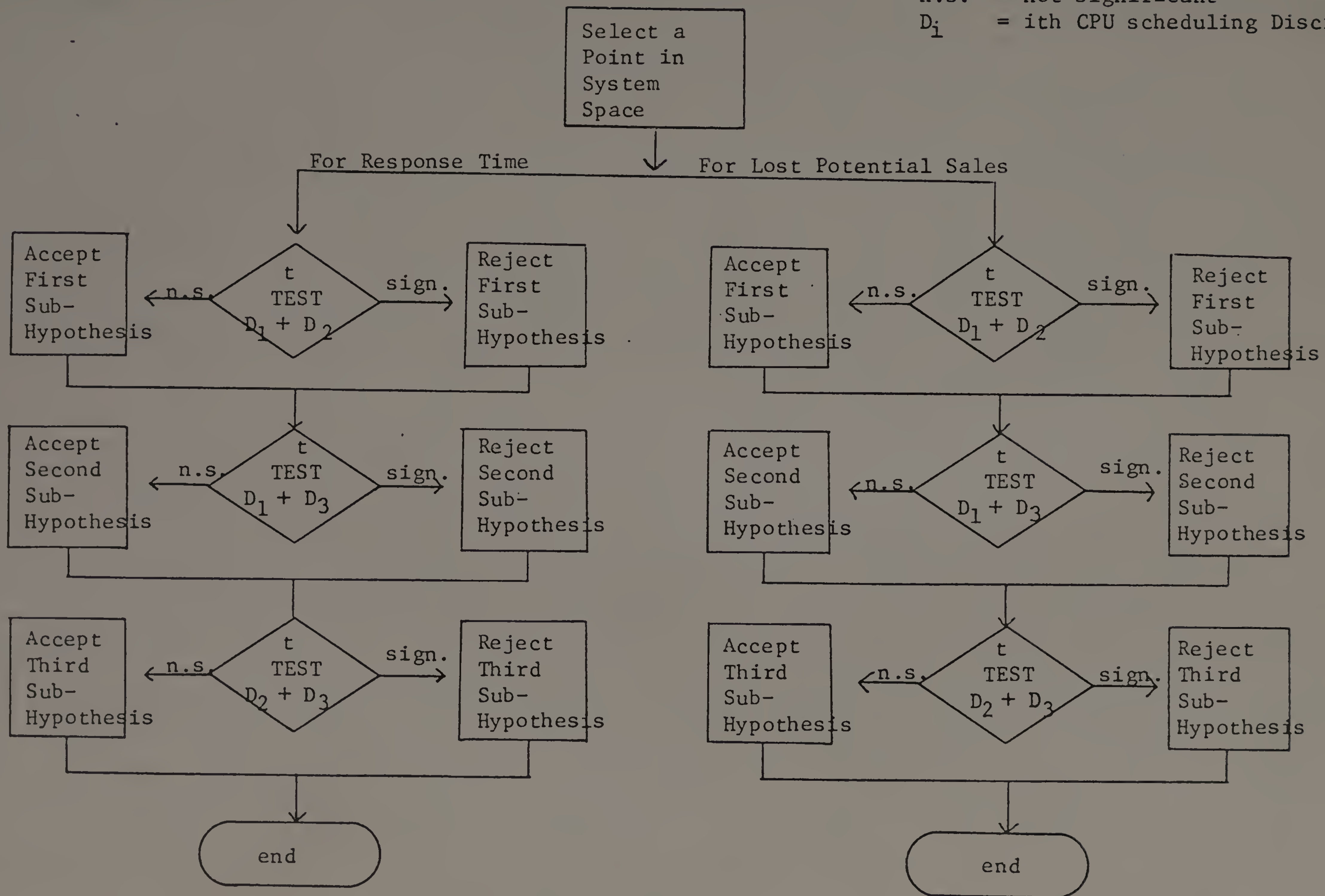


Figure 8: The Hypothesis Tree

hypotheses as a hypothesis tree. Figure 9 contains the chosen points in the system space. For clarity the main hypotheses are expressed in tabular rather than verbal form. For each hypothesis the performance measure is shown in column one. Each of the next three columns represents a value along one of the dimensions in the system space. Column 2, representing main memory capacity, shows one of two computer system configurations: sufficient main memory capacity to contain all needed routines (program page faults = 0) and a virtual storage environment in which an average of three program page faults occur for every virtual query language analyzer routine and virtual editing routine being executed.

The hypotheses are arranged in two main groups, classified first by main memory capacity. Within each group the hypotheses treat points in system space extending first along the CPU utilization dimension at low paging loads and then at high CPU utilization along the paging dimension. Lastly we consider points in the intermediate CPU utilization range under increasing paging loads.

Analytical Models of Cyclic Queueing Systems

There are basically three methods for performing computer performance evaluation on hypotheses. One can use analytical models or simulation models or one can set

HYPOTHESES	PROGRAM PAGE Faults/Routine (Memory Capacity)	CPU UTILIZATION DISCIPLINE 1		AVERAGE PAGE LOAD ON SLOWEST DEVICE*	
1A Response Time 1B Lost Potential Sales	0	Low	(0.16)	Low	(0.9)
2A Response Time 2B Lost Potential Sales	0	Moderate	(0.40)	Low	(6.8)
3A Response Time 3B Lost Potential Sales	0	Moderate	(0.62)	Low	(0.7)
4A Response Time 4B Lost Potential Sales	0	Moderate	(0.78)	Low	(0.9)
5A Response Time 5B Lost Potential Sales	0	High	(0.88)	Low	(1.0)
6A Response Time 6B Lost Potential Sales	0	Very High	(1.00)	Low	(0.3)
7A Response Time 7B Lost Potential Sales	0	High	(0.96)	Low	(6.7)
8A Response Time 8B Lost Potential Sales	0	High	(0.97)	Moderate	(14.3)
9A Response Time 9B Lost Potential Sales	0	High	(0.99)	High	(46.2)
10A Response Time 10B Lost Potential Sales	0	Moderate	(0.46)	High	(60.0)
11A Response Time 11B Lost Potential Sales	3	Low	(0.27)	Low	(1.4)
12A Response Time 12B Lost Potential Sales	3	Moderate	(0.61)	Low	(1.6)
13A Response Time 13B Lost Potential Sales	3	High	(0.88)	Low	(1.3)
14A Response Time 14B Lost Potential Sales	3	High	(0.94)	Moderate	(17.9)
15A Response Time 15B Lost Potential Sales	3	High	(0.92)	Moderately High	(27.6)
16A Response Time 16B Lost Potential Sales	3	Moderate	(0.80)	Moderately High	(27.1)
17A Response Time 17B Lost Potential Sales	3	Moderate	(0.32)	Moderately High	(31.9)
18A Response Time 18B Lost Potential Sales	3	Moderate	(0.41)	High	(98.0)
19A Response Time 19B Lost Potential Sales	3	Moderate	(0.60)	High	(185.3)

*Average queue length on slowest RAD devices.

Figure 9: Experimental Points in System Space

appropriate parameters on an existing system, if one exists, and then observe its behavior. Performance evaluation that treats development of future systems is usually restricted to the first two methods of evaluation. Engineers, designing new generations of circuits, rely heavily on the first two methods, especially before expensive prototypes are built. Analytical models are appropriate, provided the complexity of the model is not too great and the assumptions within the model can be accepted. When a system like a Product Information System can be modeled as a set of interconnecting queues, analytical models deserve attention.

Scheduling algorithms have been studied in computer systems using analytical models. Coffman (1967) reviewed single server queueing models, which can assist in the analysis of priority rules in multiprogramming systems. These include simple round-robin systems, multiple-level feedback models, which have different priorities at each level, and pre-emptive models which may or may not include priorities. Scheduling algorithms have also been studied using analytical models in multiprogramming systems in which the RAD subsystem and the CPU were multiple servers in a feedback system. Some of these studies include job scheduling, but are just as applicable to task scheduling. Adiri (1973) studied two scheduling environments: scheduling monoprogrammed jobs and multiprogrammed jobs. The following assumptions limit the generalizability of his model to Product Information Systems. The number of iterations of programs going from the CPU server to the I/O server is geometrically distributed, priority is FIFO, and pre-emption is not permitted. CPU and I/O processing have also been studied as cyclic queueing models. Boyd (1974) presents a 3-stage cyclic network queueing model of a batch processing multiprogramming system. Stage 1 represents the jobs blocked because they are waiting for permanent resources to be released by other jobs. Stage 2 represents CPU service and stage 3 I/O service. Jobs cycle from stage to stage until completion. This model, like many others, is restrictive because of the assumption that the number of entities (jobs) circulating in the system must remain constant. Boyd also considers several algorithms for scheduling jobs on processors. These job schedulers can be used in conjunction with task schedulers.

In analytic terms a node in a Product Information System can be viewed as a cyclic queueing model, with queries moving from customer through representatives, through the computer system and answers flowing back to the customer. The analytic study of cyclic queues was prominent as early as the 1950's (Koenigsberg), but has only recently been applied to computer system problems. Gordon and Newell (1967) developing the work done by Jackson

(1957, 1963) and others, have shown how a cyclic queue can be structured as a Markov process. Buzen (1973) extended this work by examining some computational aspects of the basic equilibrium distributions and presented algorithms for determining such equilibria under both constant and variable service times. Yet, in order to use presently developed Markov models in the study of Product Information Systems, the problem of the constant number of circulating entities, N, remains. Lewis and Shedler (1971), in modeling a multiprogramming computer system operating under demand paging as a cyclic queueing model, state "the assumption that N is a constant is an approximation that is justified by the common practice of operating such a system in a saturated mode." But this assumption is not justified for a Product Information System. The number of circulating entities in a Product Information System varies, first because the cyclic queue is not closed; customer traffic does vary. But more importantly, even assuming a peak load of customers, the number of page accesses circulating in the RAD subsystem will vary, being dependent upon the nature of the query.

The complexity of modeling a Product Information System analytically as a cyclic queue is increased when we consider that there are subcycles within the major cycle. These subcycles are formed by the program page faults. The modeling complexity is compounded by the fact that an accurate model must be able to reflect the overhead associated with scheduling, the pecularities of each discipline being studied, and the CPU's being fed by both a pre-emptive and a non-pre-emptive queue. Most cyclic queueing models, when applied to computer systems have assumed that all queues are processed in a FIFO manner. An exception to this is the Lewis and Shedler model which advanced the state of the art by considering both system overhead and pre-emptive scheduling. The overhead they consider includes the CPU processing needed to switch from one program to another, I/O handling, queue management, and the execution of page replacement algorithms. Yet, again N must be assumed to be constant, as well as the exponential nature of successive program execution intervals.

We now look at two other problems with the application of feedback or cyclic queueing models to Product Information Systems. The first is that in a Product Information System query processing produces bursts of page requests. As a result the arrival rate of page requests to the RAD subsystem does not follow a typical Poisson or other easily usable distribution. Secondly, the RAD subsystem service time is not independent of the size of the queues in front of the channel and devices. We have already alluded to the fact that under heavy I/O a disk device having Rotational Position Sensing will experience more wasted revolutions trying to reconnect with its channel. Delbrouck (1970) attempted to model as a feedback queueing system burst arrivals and queue dependent service time with only a single server. In his model requests for service occur in batches of varying size and are served bulkwise up to a specified maximum during the service cycle. The service time fluctuates in response to variations in the volume of demand. Delbrouck points out that, if the queue feeding this single processor is large, the mathematical analysis becomes intractable. He states

> . . . the results of such analysis (of processors) must be viewed with reserve. At worst it may be too crude an approximation; at best it may provide an independent means of monitoring a more sophisticated simulation of the system.

It is because of the modeling complexities mentioned above that we have chosen simulation as an appropriate technique for testing these hypotheses. By using simulation we are not forced into making the restrictive assumptions that would have to be made if analytical models were employed. In studying computer systems, simulation has been used for modeling entire computer centers, including the activity of people (Hutchinson, 1965), combined batch and time sharing systems (Norton, 1970; Mikelskas, 1973), batch systems alone (W. Martin, 1970), and time sharing systems (Fine and McIsaac, 1966; Nielsen, 1967; Blatny, et al, 1972). Simulation has been used for modeling computer jobstreams (Kaspar and Miller, 1974; Newkirk and Mullin, 1974) and operating systems (Hutchinson, 1973).

Choosing a Simulation Language for Modeling

Once simulation has been chosen as a modeling technique, a suitable language must be chosen to express the model. The most widely used simulation languages are SIMSCRIPT II.5 and GPSS. FORTRAN, although a general purpose language, is still used by many researchers. Other simulation languages are used to a lesser extent. GASP is a FORTRAN embedded simulation language. Simulation languages specially developed for the study of computer systems are beginning to evolve. One such language is OSSL (Dewan, Donaghey, and Wyatt, 1972). Another is ECSS II, Extendable Computer System Simulator (Kosy, 1973; Feingold and Chao, 1974). This simulation language is structured as a superset of SIMSCRIPT II.5. Unfortunately, ECSS II, which was developed by the RAND corporation for NASA, has not been released for general use to either universities or businesses.

SIMCRIPT II.5 was selected as the modeling language for Product Information System study for a variety of reasons. First, it is one of the most flexible languages for discrete event simulation. This pertains to both model development and output specification. SIMSCRIPT has been chosen over FORTRAN because FORTRAN has no built-in features to aid in modeling. A few of these desired features include event scheduling, queue handling, data gathering from the simulation, and statistical analysis of the results of the simulation. SIMSCRIPT was chosen over GPSS, since SIMSCRIPT programs are generally more readable and thereby more self-documenting. The argument that GPSS is better than SIMSCRIPT since GPSS allows one to model processes (sets of events), is no longer valid. Recent releases of SIMSCRIPT include this modeling ability.

Modeling a Product Information System with SIMSCRIPT

SIMSCRIPT models a system by viewing it as a group of permanent and temporary entities undergoing changes of state. A permanent entity will usually remain in existence for the duration of the simulation, while temporary entities are created and destroyed as the system changes over time. Each entity can have its own attributes (characteristics) and entities can belong to sets based on some commonality. Entities, attributes, and sets are in themselves static elements of a system. The attributes of the entities and the set memberships are changed by events. In SIMSCRIPT, events are routines which are executed at definite points in simulated time and, when executed, dynamically change the state of the system. An event then is a routine whose execution is time dependent. Events can be scheduled exogenously (from outside the model at selected simulation times) or endogenously (from within the model dependent upon state conditions therein). A simulation model in SIMSCRIPT moves through time by allowing these events to be able to schedule other events and even themselves immediately following or at some time in the future. SIMSCRIPT has a built-in scheduling mechanism which keeps track of which events are scheduled for what times. This scheduler also resolves conflicts when more than one event is scheduled for the same time.

In a model of a Product Information System the permanent entities might include CRTs, CPUs, and secondary storage channels and devices. It is important to note that, instead of creating a multitude of different entities, with the judicious use of a few, all of the elements of the system to be modeled can be preserved. For example, although the greet time and think time are actually characteristics of the firm's representatives manning the CRTs, one can specify these as attributes of the CRT, with no loss of modeling capability. This is one example of the fact that in developing models one need not have a slavish one-to-one correspondence between the elements in the model and the system being modeled as long as the behavior of the system being modeled is not distorted.

Attributes of the CRTs include the GREET.TIME, the THINK.TIME, the TYPING.TIME, a designation of whether the CRT is engaged or not (AVAILABILITY) and which query is assigned to each CRT (QUERY.ADDR).* The CPU entity has the following attributes. It has an execution SPEED, a STATUS indicating whether it is busy, the address of the routine presently executing (ROUT.ADDR), the time that the routine began executing (TIME.ON), and the address of a routine that may have been pre-empted (EV.ADDR). The first permanent entities in the secondary storage subsystem are the channels (RAD.CHANNEL) between the secondary storage devices and main memory. A channel's attribute (BUSY.2) keeps track of whether the channel is busy or free. The secondary storage devices (RAD) are permanent entities having the following attributes. The attribute ARM differentiates drums from disks and specifies for disks the current cylinder location of the access arm. An arbitrary value of -1 for ARM designates a drum. If ARM is a non-negative number, this attribute specifies at what cylinder the disk access arm is present. Each disk has a number of cylinders (NO.OF.CYL), an average seek time (AVE.SEEK), and a randomly calculated seek time for each access (SEEK.TIME).

---

*For clarity, entities, attributes, sets, and events used in the simulation model of a Product Information System as presented in Appendix 3 will be capitalized in the text.

Each device has a rotation speed (RAD.ROTATION.SPEED) and may have rotational position sensing (RPS). Each device is allocated to a LEVEL within the hierarchy of secondary storage devices and is allocated to a particular channel (CHANNEL.HOOK). Each device can contain a specified number of PAGES.PER.TRACK. The attribute RAD.BUSY maintains the status of each device as busy or free.

Temporary entities include queries (QUERY) and the DBMS and operating system routines (VIRTUAL.ROUTINE), which process the queries. In the model, because all routines are shared routines and thus have similar modeling properties, each is designated as VIRTUAL.ROUTINE. The attributes of each of the temporary entities QUERY are the following. Each QUERY has a unique query number (QUERY.NUM), a time when it came into existence (GEN.TIME), and a time marking the beginning of the system response time (CONVERSATION.END). Most of the information concerning a query is maintained in the attributes of the simulated routines processing the queries. As the state of the system changes affecting a query, these changes are captured in the attributes of the routines processing the queries. Information is also passed from routine to routine as these routines are called into play to process the simulated queries. Each VIRTUAL.ROUTINE, when created, is given a unique identification number (VR.NO), a specification of whether it belongs to the operating system or the data base management system (SUPERVISOR) and a designation of the type of routine (TYPE) as listed in Figure 1. The CRT.NUMBER is an attribute of the VIRTUAL.ROUTINE. This ties each virtual routine indirectly to a query through the CRT being serviced. Each virtual routine has a time at which it was created (CREATE.TM), a rank within queue B controlled by discipline 2 (LOSS.RANK) and a rank controlled by discipline 3 (BLOCK.RANK). Each routine has attributes specifying the total amount of CPU time needed to run (TOTAL. CPU.TIME.NEEDED), the amount of CPU time still needed at the current simulated time (CPU.TIME.TO.GO), the time till the next routine's program page fault (TIME.TILL.BLOCK), the accumulated number of times the routine has experienced a program page fault (TIMES.BLOCKED), and the number of times the virtual routine has been pre-empted (INTERRUPTS.PER.RUN). A virtual routine which indirectly processes a query by handling another virtual routine has an attribute, SERVICING. ROUTINE.ADDR, which acts as an address pointer to the routine being serviced. Attributes are also maintained for keeping track of which channel (RAD.CHAN) and device (RAD. DEVICE) is to be used when a routine experiences a program page fault. These attributes are also used for data page faults generated by the page directory lookup routine. The total number of pages generated by the page directory lookup routine is kept in the attribute TOTAL.PAGES.TO.BE.CREATED and the time at which these page requests enter the secondary storage subsystem is designated by the attribute SERV.START. If a device fails to reconnect to its channel because the channel is already busy servicing another device, this condition is kept in the attribute WANT.CHANNEL. In modeling query processing we focus on the routines actually doing the processing. We create these routines when needed, define their attributes, move them through the various queues in the model and capture their changed states. When RAD subsystem processing is needed for data, we model this as if the routine itself moved through the RAD subsystem and then rejoined queue B for more processing on the CPU.

Data references to secondary storage are modeled as temporary entities moving through secondary storage processing. Because of the similarity in which program page faults and data references are actually processed within a computer system, and instead of needlessly creating a new class of temporary entity, we use the structure of the VIRTUAL.ROUTINE entity. A data reference is differentiated from an actual virtual routine by coding the attribute TYPE with the arbitrary code value 20.

There are a number of sets or queues within the simulation model. TEL.QU is a telephone hold queue for queries before they enter the system. This queue can be used if one wishes to model the system under conditions of a saturated terminal subsystem. The pre-emptive CPU queue A is designated as CPU.QUEUE. There are three CPU DBMS queues in the model--one for each CPU scheduling algorithm. ONE.QU is associated with discipline 1. LOSS.QU is associated with disciplines 2 and 3. BLOCK.QU is associated with discipline 3. There is a queue associated with each channel (RAD.CHANNEL.QU) and with each secondary storage device (RAD.QU).

The simulation progresses through time by the events within the simulation model. The simulation model can be initialized to represent either a gradual increase of query traffic, or it can start with all CRTs active. The event TELEPHONE.CALL creates a query, defines its attributes and files the query into the TEL.QU. If the simulation is to begin with all CRTs active, then in the simulation routine INITIALIZATION the event TELEPHONE.CALL is scheduled to occur simultaneously for each CRT. If the start up conditions are to be more gradual, the event TELEPHONE.CALL reschedules itself at one second intervals until all CRTs have been activated. The event TELEPHONE.CALL also schedules the event ARRIVAL.AT.CRT.

The event ARRIVAL.AT.CRT removes the query from the telephone queue, tags as busy the CRT associated with the query, defines attributes for this CRT, and schedules the next event. This event, CRT.TRANS.READY then creates a

VIRTUAL.ROUTINE to handle the CRT interrupt. After the query has entered the main memory, the event CRT.TRANS. FINISHED creates a virtual image of the CPU scheduling discipline, defines its attributes, and sets this routine to run on the CPU immediately, or places it in queue A.

The event RUN.IT models the beginning of CPU activity. This event controls pre-emption. If a routine is pre-empted, the amount of CPU time used for that routine is updated. The event then schedules the event OFF.CPU. OFF.CPU is scheduled to occur at that time which is the minimum of CPU.TIME.TO.GO and TIME.TILL.BLOCK. This event also maintains the CPU attribute STATUS from which CPU utilization can be derived. The event OFF.CPU performs all the processing that a virtual routine would be expected to perform while on the CPU. In addition, this event models the different kinds of terminations on the CPU. This event models the activity of each type of virtual routine under the conditions of completion, being pre-empted or having encountered a program page fault. The event OFF.CPU also updates the time a virtual routine was on the CPU, changes the STATUS of the CPU to not busy, and selects the next routine to run on the CPU.

When program page faults and data references to secondary storage occur, events applicable to the RAD subsystem are processed. A request for a demanded page goes into a specified RAD.QU, unless it can be immediately handled. The designated channel and device must be simultaneously free if a page request is to be processed. If the target is a drum having rotational position sensing, a LATENCY.END event is scheduled. This is scheduled to occur at the end of one half revolution of the drum. If the target device is a disk with rotational position sensing, a LATENCY.END event is scheduled to occur at the end of a combined seek and one half revolution of the device. If the device is a disk without rotational position sensing, a SEEK.END event is scheduled. The scheduled time is based on a linear regression of plotted data known to characterize IBM disks (IBM, 1966). In the simulation model a SEEK.END event schedules a LATENCY.END event and this in turn schedules a READ.END event. This allows one to bootstrap the model from one event to another.

The event READ.END, signaling the end of the transmission of a page from secondary storage into main memory, changes the state of the channel and device to not busy and captures statistics on the service time distribution of the RAD subsystem. In the event READ.END, if the page entering main memory were a program page, then a virtual routine of the CPU scheduling discipline is created and filed into queue A. The CPU scheduling discipline, when run, will place the interrupted query processing routine back into queue B. When the last data page of a query enters main memory, event READ.END creates an editing virtual routine and a virtual routine of the CPU scheduling discipline. The latter is filed into queue A and, when run, places the editing routine in queue B. The event READ.END also selects the next page request for the free device and/or channel.

At the termination of the editing routine the event OFF.CPU creates a virtual routine (TYPE = 6) to handle the transmission of the answer back to the CRT. After this latter routine is run on the CPU, the event OFF.CPU schedules the arrival of the answer at the appropriate CRT, which arrival is the event ANSWER. The event ANSWER computes the system response time, posts this to a table of response times, determines if the transient period of the simulation has ended and determines if the modeled system has reached a steady state or whether the simulation should run for another 1000 queries. The transient period reflects the start up conditions of the simulation and so the response times for this period are dropped from later statistical analysis.

The event ANSWER schedules the last event (AN.END) for a query, which represents the end of the conversation between the customer and the firm's representative. Since we are assuming one query per customer transaction and peak customer traffic, the event AN.END then schedules the beginning of the next telephone call sequence. Thus the cycle is renewed.

The above description has given a synopsis of the model from the point of view of what each event performed.

In order to understand the dynamic nature of the model, Figures 10 and 11 are presented. These figures present the simulation model from the view of a query being processed. Further details are found in Appendix 3. Since events in an event-driven simulation are considered as occurring instantaneously in simulated time, time is measured as an interval between two simulated events. Thus, Figure 10 shows for a single query the events that are executed as well as what has occurred during the time interval between events. Figure 11 goes into more detail on the modeling of the processing of a single query. This figure lists each event needed to model the processing of a query and the simulation routines called upon within each event. Figure 12 lists the events within the RAD subsystem.

Having considered the simulation model of a Product Information System in terms of its entities, attributes, sets, and events, we now consider the necessary controls that must be built into the model to insure accurate results in the testing of our hypotheses. The first question to be answered is what is one's criterion of "accurate" results? Simulation models of computer systems can be divided into three broad classes. Class I represents models that have an exactly similar existing referent. Class III models do not have an already existing referent. Class II models have partial referents; some of the subsystems or some of the

Figure 10: Event Sequencing in a Product Information System Model.

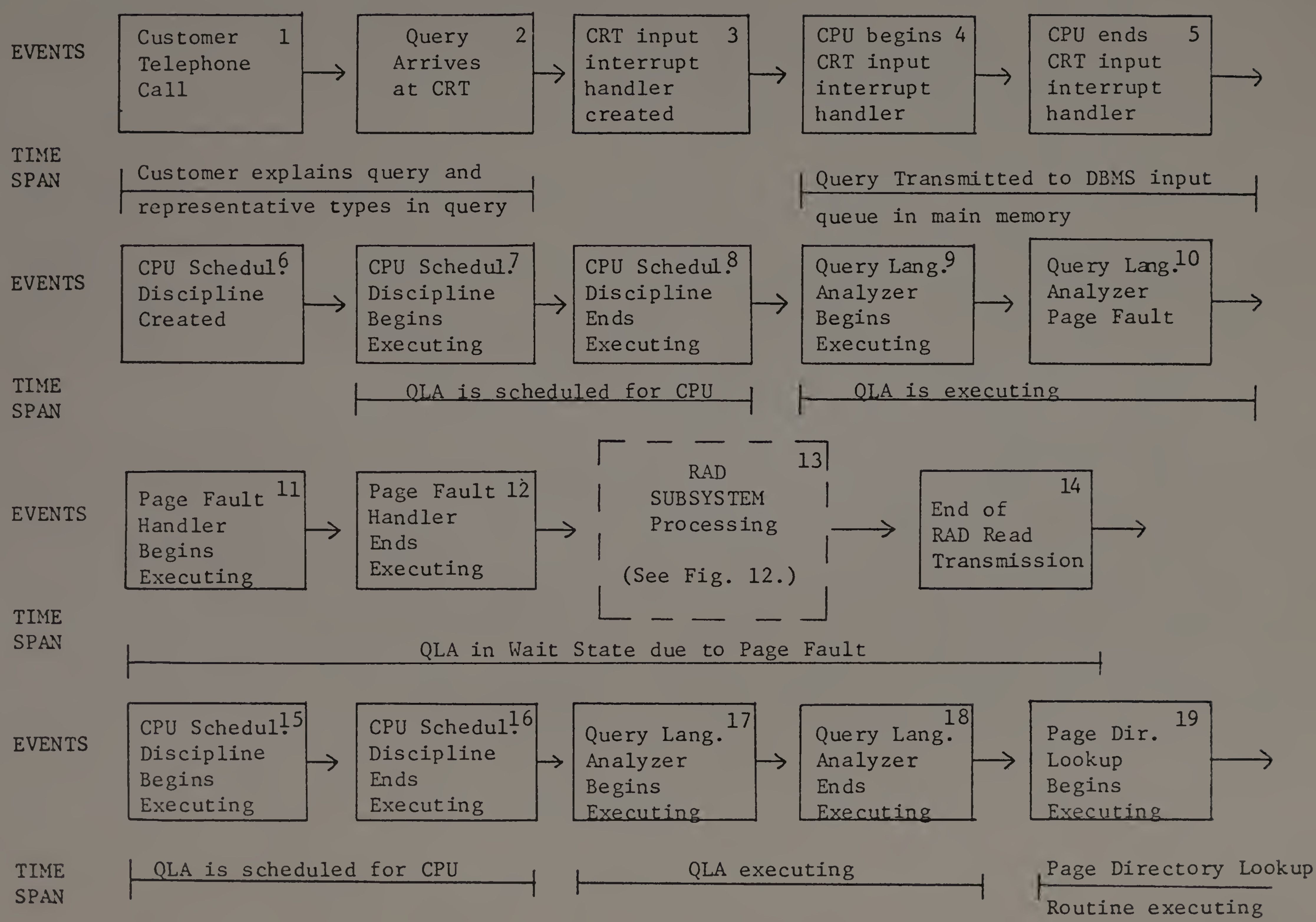


Figure 10: Event Sequencing in a Product Information System Model (CONT.)

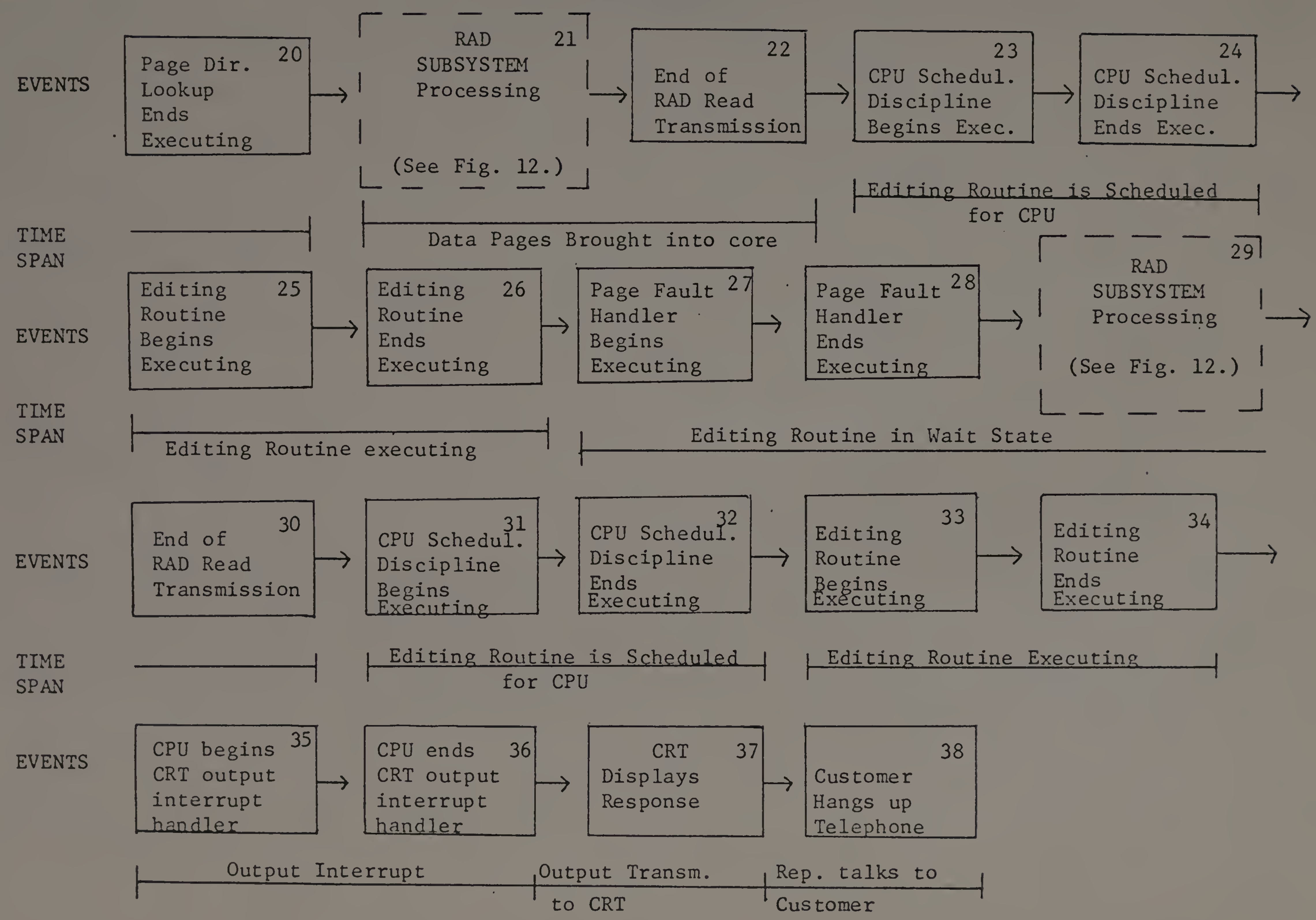


Figure 11: Event and Routine Sequencing for Query Processing in a Product Information System Model.

Sequence Number	Events & Routines	Event or Routine	Activity
1	TELEPHONE.CALL	Event	Customer places call to firm.
2	ARRIVAL.AT.CRT	Event	Firm's representative begins to service customer.
3	CRT.TRANS.READY	Event	
3a	FILL	Routine	Defines attributes for CRT input interrupt handler.
3b	CPU.CHECK	Routine	Schedules for CPU the execution of the CRT interrupt handler.
4	RUN.IT	Event	CRT input interrupt handler begins executing.
5	OFF.CPU	Event	CRT input interrupt handler ends executing.
5a	PIGGY.BACK	Routine	Schedules the end of CRT input transmission.
6	CRT.TRANS.FINISHED	Event	
6a	FILL	Routine	Defines attributes for CPU scheduling discipline.
6b	CPU.CHECK	Routine	Schedules CPU scheduling discipline.
7	RUN.IT	Event	CPU scheduling discipline begins executing.
8	OFF.CPU	Event	CPU scheduling discipline ends executing.
8a	PIGGY.BACK	Routine	
8aa	FILL	Routine	Defines attributes of QLA
8ab	INSERT.QUEUE	Routine	Puts QLA in CPU DBMS queue.
9	RUN.IT	Event	QLA begins executing on CPU.
10	OFF.CPU	Event	QLA incurs a program page fault.
10a	PIGGY.BACK	Routine	
10aa	LINK	Routine	Defines attributes for Page Fault Handler.
11	RUN.IT	Event	Page Fault Handler begins executing.
12	OFF.CPU	Event	Page Fault Handler ends executing.
12a	PIGGY.BACK	Routine	Determines program page address on RAD and initiates accessing.
13	(See Figure 12.)		
14	READ.END	Event	Program page of QLA enters memory.
14a	FILL	Routine	Defines attributes for CPU scheduling discipline.
14b	CPU.CHECK	Routine	CPU scheduling discipline is scheduled.
15	RUN.IT	Event	CPU scheduling discipline begins execution.
16	OFF.CPU	Event	CPU scheduling discipline ends execution.

Sequence Number	Events & Routines	Event or Routine	Activity
16a	PIGGY.BACK	Routine	
16aa	INSERT.QUEUE	Routine	QLA enters CPU DBMS queue.
17	RUN.IT	Event	QLA resumes executing on CPU.
18	OFF.CPU	Event	QLA ends executing on CPU.
18a	PIGGY.BACK	Routine	
18aa	FILL	Routine	Defines attributes of page directory lookup routine.
18ab	LINK	Routine	Defines attributes for CPU scheduling discipline.
19	RUN.IT	Event	CPU scheduling discipline begins execution.
20	OFF.CPU	Event	CPU scheduling discipline ends execution.
21	RUN.IT	Event	Page directory lookup routine begins execution.
22	OFF.CPU	Event	Page directory lookup routine ends execution.
22a	PIGGY.BACK	Routine	
23	(See Figure 12.)		
24	READ.END	Event	Last data page enters main memory.
24a	FILL	Routine	Defines attributes for CPU scheduling discipline.
24b	CPU.CHECK	Routine	CPU scheduling discipline is scheduled.
25	RUN.IT	Event	CPU scheduling discipline begins executing.
26	OFF.CPU	Event	CPU scheduling discipline ends executing.
26a	PIGGY.BACK	Routine	
26aa	INSERT.QUEUE	Routine	Editing routine is placed in CPU DBMS queue.
27	RUN.IT	Event	Editing routine begins executing.
28	OFF.CPU	Event	Editing routine ends executing.
28a	PIGGY.BACK	Routine	
28aa	LINK	Routine	Defines attributes for page fault handler.
29	RUN.IT	Event	Page Fault Handler begins execution.
30	OFF.CPU	Event	Page Fault Handler ends execution.
30a	PIGGY.BACK	Routine	Determines program page addresses on RAD and initiates accessing.
31	(See Figure 12.)		
32	READ.END	Event	Program page of editing routine enters main memory.
32a	FILL	Routine	Defines attributes for CPU scheduling discipline.
32b	CPU.CHECK	Routine	CPU scheduling discipline is scheduled.

Sequence Number	Events & Routines	Event or Routine	Activity
33	RUN.IT	Event	CPU scheduling discipline begins executing.
34	OFF.CPU	Event	CPU scheduling discipline ends executing.
34a	PIGGY.BACK	Routine	
34aa	INSERT.QUEUE	Routine	Editing routine is placed in CPU DBMS queue.
35	RUN.IT	Event	Editing routine resumes executing.
36	OFF.CPU	Event	Editing routine ends executing.
36a	PIGGY.BACK	Routine	
36aa	FILL	Routine	Defines attributes of CRT output interrupt handler.
37	RUN.IT	Event	CRT output interrupt handler begins executing.
38	OFF.CPU	Event	CRT output interrupt handler ends executing.
38a	PIGGY.BACK	Routine	
39	ANSWER	Event	CRT screen displays response.
40	AN.END	Event	Customer hangs up.

Sequence No.	Events & Routines	Event or Routine	Activity
1	SEEK.END	Event	RAD device arm is positioned over correct cylinder.
2	LATENCY.END	Event	End of latency period.
3	READ.END	Event	End of data transmission from RAD device.

Figure 12: Event Sequencing in the RAD Subsystem.

variables can be compared with existing referents. In the first class of models the results of a simulation run can be compared against an existing system, which has all the characteristics being modeled. This is called validation. The literature still rings with the philosophical dispute on whether a model of the first class must have a one-to-one structural similarity to its existing referent, or whether its predictive prowess is sufficient. If one restricts a model to a one-to-one structural similarity, then a death blow is dealt to modeling by analogy. But this would seem too restrictive.

Most computer performance evaluation problems demand solutions to problems of classes II and III. Frequently our computer performance evaluation questions involve either possible future changes to parts of an existing system (class II) or the development of completely new systems (class III). Examples of class II problems and models abound. How would the replacement of one operating system with another affect a given system? What will be the affect of replacing these tapes with those disks? With class II models one should obviously try to validate as many parts of the model against an existing referent.

Existing referents do not exist for the study of many future systems. Models of future systems are of necessity class III models. Again there are countless examples of simulation models of class III. The design of new shipping facilities, new traffic control systems, new train systems are some examples. The work of Jay Forrester is perhaps the most widely known in the area of modeling future systems. The criticism leveled against Forrester's work has been mostly on his selection of parameter rates which are based on historical and/or projected trends, not on whether his world dynamic models have an existing referent. The future is not contained wholly in the present and we must make decisions today which will influence the future.

With class III models, in order to begin on a firm foundation, one must extrapolate and project from the present to the future in terms of the selection of variables, the selection of parameters, and the relationships between the variables. Verification is assuring that these are plausible and that the behavior of the model as perceived by the user reflects the relationships between the variables (Kleijnen, 1974). Verification is part of the establishment of the accuracy of a model. The selection of an appropriate experimental design and statistical analysis are also needed to assure the accuracy of the model.

There is a difference between the accuracy of a simulation model and the usefulness of such a model. The former is a necessary condition of the latter. The usefulness of a class III model is increased as the probability that it will mirror a future system is increased and in proportion to the model's ability to represent a wider range of futurables.* It was with this in mind that the hypotheses generated in this work pertain to a system space as a range of futurables.

In class III simulation models, accuracy is attained by building controls into the model at each stage of its development and use. Verification involves the first two stages. The first stage is the design of the simulation model. In this stage one must insure the proper selection of dependent and independent variables, a representative set of values for the independent variables, and a correct representation of the interrelationships between these independent variables. In our simulation model of a Product Information System the dependent variables are commonly accepted measures of performance. The independent variables listed in Figure 7 are representative values, ranges or settings that would be expected in a well-designed Product Information System. The second stage is the implementation of the simulation model. In the implementation stage one must insure that the interrelationships proposed on the logical level are actualized when the model is coded into a computer language. In the Product Information

---

*A futurable is a variable, variable value or confluence of variables and values, (a system), whose future existence is possible or probable, based on present conditions.

System model modular development of the simulation routines enabled each major routine to be tested before being incorporated into the model. Secondly, a tracing facility was built into the simulation so that, when the model was being debugged, the model's performance could be monitored. This insured that the interfaces were correct when the simulation events and routines were put together into the single model.

The third stage of development is the design of experiments using the model. Each computer run of the simulation model actually performs three simulations: one for each of the three CPU scheduling disciplines. Each discipline is run under the same system conditions, including the same random number generator. This insures a controlled experimental environment for comparing the relative behavior of these disciplines, when measured by the dependent performance variables.

In the design of experiments using simulation models one is faced with the problem of determining an adequate sample size. This problem is common to all experimentation, but has some peculiarities in simulation modeling. In a simulation one must begin by determining the beginning of the steady state, assuming we are interested in the steady state, and in some simulations this may involve a long transient period (Eilon and Chowdhury, 1974). The second problem is to then determine the sample size after the transient period has been removed.

Conway (1963) suggests one method for detecting and deleting the transient period.

> As a rough guide I usually truncate a series of measurements until the first of the series is neither the maximum nor minimum of the remaining set. I do not do this for every run, but rather decide on a stabilization period by examining a few pilot runs and thereafter delete this same period from the results of each run.

Emshoff and Sisson (1970) point out that there are two ways of handling transient periods. The first is to have a run long enough that the transient period effects are diluted. This, however, causes its own problem. How long is long enough? The second method is to detect and delete. Emshoff and Sisson (1970) suggest that

> If the number of observations in which the output is greater than the average to a given point is about the number in which it is less, then steady-state conditions are likely to exist.

We employ the use of the Cox Stuart nonparametric trend test to detect the transient period. This test is used to detect a trend in response times as the customer load on the system is gradually increased. When a trend is no longer detected, the system is considered to be in the steady state. It takes up to 400 processed queries in some runs for a break in trend. In each simulation, the Cox Stuart test was applied at intervals of 100 queries until no trend was detected (See Appendix 3).

The starting conditions of a simulation run can influence the length of the transient period. A problem with starting a simulation under conditions of "empty and idle" is that it takes longer for the system to reach a steady state. But as Meier, Newell, and Pazer (1969) point out, it may not be easy to determine a priori what the loaded model should look like. We are willing to bear this longer transient period in order not to have the model experience unnecessary oscilations, which would make the detection of the transient period more difficult. Although we wish to study the model under peak conditions (all CRTs active), we start up the system gradually as described previously. By increasing the number of active CRTs gradually, the response times during the transient period will tend to increase as the system queues fill up.

Once the transient period has been detected, one can then begin to consider an appropriate sample size for the observations. To put this problem in the context of a simulation experiment, when does one stop the simulation? In modeling a Product Information System under each discipline the question becomes how many queries must be processed after the transient period. Meier, Newell, and Pazer (1969) list different methods of sampling. A number of techniques have been suggested in the literature for both the determination of sample size and the increase of statistical confidence intervals for a given sample size. Among the variance reduction techniques are sequential estimation (Moshman, 1958; Sasser, et al, 1970), the introduction of positive serial correlation between runs under different operating conditions and replication using negative correlation as in the use of antithetic variates (Emshoff and Sisson, 1970; Fishman, 1973). The following technique was employed to determine the stopping point for each simulation. At intervals of one thousand processed queries, all response times after the transient period were divided into two consecutive halves. The first half represented the sample. The second half represented the replication of the sample. The distribution of the sample and the distribution of the replication of the sample were compared using the Mann-Whitney U test at the 0.05 alpha level. Siegel (1956) states, " . . . the Mann-Whitney U test may be used to test whether two independent groups have been drawn from the same population." The use of this test is appropriate, since it obviates the problems that might occur if a parametric test were employed--the assumptions of normality and homogeneity of variance. It should be pointed out that this test is one of the most powerful of the nonparametric tests, and this test closely approximates the power of a parametric test when the number of observations, as we are using, is large. If the sample and the replication could not be shown to come from different populations, then we accept the run length of the simulation as being adequate as a representation of the steady state of the system. This varied between four thousand and seven thousand processed queries for most of the simulations. Therefore, the simulations were run for a minimum of seven thousand queries,and each discipline was checked with the Mann-Whitney U test at intervals of one thousand queries. The transient period ranged from one hundred to four hundred queries and these observations were discarded from the seven thousand observations.

With the transient period eliminated and a suitable representation of the steady state selected, the fourth stage, the statistical analysis of the results, could be initiated. As previously mentioned, for each hypothesis generated from the system space, the behavior of the CPU scheduling disciplines, as measured by the primary dependent variables, was then tested in pairs: disciplines 1 and 2; disciplines 1 and 3; and disciplines 2 and 3.* The samples

---

*Since we are treating three disciplines at each point in system space, one might suggest the use of the classical analysis of variance to test for differences between the disciplines. A number of reasons favored the use of the pairwise _t_ test. First, the two-sample _t_ test is more robust against heterogeneity of variance. Since discipline 2 is a variance reduction algorithm, an assumption of homegeneity of variance in observations of system response times between disciplines seems unwarranted. Bradley (1968, p.25) were tested for differences in population means by calculating the Z-score,

$$Z = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{s_{\bar{x}_1}^{\;2} + s_{\bar{x}_2}^{\;2} - 2(r)(s_{\bar{x}_1})(s_{\bar{x}_2})}}$$

This formula for the Z-score includes the commonly used correction for correlation between the two samples, since if the effects of the correlation between two experimental groups are not removed, the standard error of the mean is too inflated, producing the erroneous conclusion of no difference between the means (Kerlinger, 1964). We judge a significant Z-score (>1.96) would cause the rejection of the

---

comments ". . . although the two-tailed, two-sample _t_ test is a special case of the analysis-of-variance F test, the perfect robustness of the former against heterogeniety of variance when sample sizes are equal and infinite does not extend to the latter in the general case." A second reason for using the _t_ test instead of the analysis of variance is that experimental evidence has shown that the _t_ test is robust even in small samples against simultaneous non-normality and heterogeneity of variance, provided the number of observations in each sample are of equal size. Boneau (1960) states, "By invoking a few theorems of mathematical statistics it can be shown that if one samples from any two populations for which the Central Limit Theorem holds, . . . no matter what the variances may be, the use of equal sample sizes insures that the resulting distribution of t's will approach normality as a limit." Boneau presents experimental evidence and concludes that it would appear from the present results that the approach to normality is rather rapid, since samples of sizes of 15 are generally sufficient to undo most of the damage inflicted by violations of the assumptions.

null hypothesis and the alternate hypothesis that the two means were significantly different to be accepted.

Parametric tests performed on hypotheses are generally regarded as assuming normality, a common variance, and sampling independence. Kerlinger (1964, p. 258), however, points to a number of studies showing that statisticians have for too long unnecessarily stressed the importance of normality and homogeneity of variance.

> The evidence to date is that the importance of normality and homogeneity is overrated, a view shared by the author. Unless there is good evidence to believe that populations are rather seriously non-normal and that variances are heterogeneous, it is usually unwise to use a nonparametric statistical test in place of a parametric one.

Kerlinger points to Lindquist, who states that the probability statements resulting from $\underline{t}$ and F tests will be highly accurate even when the assumptions of normality and homogeneity are violated. Parametric tests have better discriminatory power in rejecting the null hypothesis, when this hypothesis is in fact false. Nonparametric tests are therefore more conservative.

The assumption of the independence of observations is important in the design of experiments. While the influence of dependent observations can be circumvented in some experiments by the use of paired observations, this is not applicable to our experimental situation. Autocorrelation in simulation experiments can present problems for the independence assumption on observations. Emshoff and Sisson (1970) present two methods generally used to deal with autocorrelated simulation results. We employ their second method, the blocking method, to remove the effects of autocorrelation from each simulation. The procedure is to divide the sample observations, after the transient period observations have been removed, into intervals longer than the interval of autocorrelation and use the mean of each of these "blocks" as a single independent observation. The mean and standard deviation of these independent individual observations can then be calculated in the ordinary manner.

How is the size of the blocks to be determined? To do this we use the serial correlation test suggested by Costis (1972) to test for independence in the samples. We calculate the correlation coefficient for a given lag (distance between observations.) The idea here is that if the data is serially correlated, this correlation will decrease as the distance between consecutive observations increases. We begin with a lag of 1. A correlation coefficient is calculated and tested under the null hypothesis by the use of a two-tailed t test. We continually increase the lag between the observations until the correlation coefficient is not significantly different from zero at the 0.05 level. This gives us independent observations for each CPU scheduling discipline. When two CPU scheduling disciplines are to be compared for either a difference in mean system response times or mean potential sales loss, an equal number of independent observations is selected under each discipline. The lag necessary to provide independent observations varied in the simulations up to a maximum of seventy observations. This provided the smallest sample size for independent observations, a sample size of ninety-six. Most sample sizes exceeded two hundred observations and some were over one thousand. These samples were then tested in pairs by a two-sample *t* test.

In summary, each simulation run was composed of three simulations, one for each of the three CPU scheduling disciplines. Each simulation *run* tested the three disciplines at one point in system space. Each simulation of a discipline in turn was composed of approximately seven thousand observations, from which one hundred to four hundred observations were discarded (the transient period). The basic observations were the system response times, the first primary dependent variable. From these observations the potential sales loss due to these response times was calculated. The potential sales loss represents the second primary dependent variable. These two variables are the performance variables according to which the CPU scheduling disciplines were tested for relative behavior.

Safeguards were built into each simulation to remove the transient period and to insure that the observations represented steady state conditions. The resulting observations of each simulation were then written out to magnetic tape. A second computer program then subjected these observations to the removal of autocorrelation thereby providing statistical independence for the observations. The three simulations in each run were tested in pairs for statistical differences in both performance variables by the calculation of Z-scores. The results of these experiments are presented and discussed in the next chapter, Chapter IV.

We conclude this chapter by presenting some information concerning the actual running of the experiments on a computer. Compilation time for the simulation model (program 1) takes approximately five minutes under SIMSCRIPT II.5, version D, running on an IBM 370/145 under VS2 release 1.6. The computer time needed to perform each simulation run ranged between two to six hours on a dedicated IBM 370/145. This time represents mostly CPU time, aside from the time needed to read in the table driven input and write all response times to tape. The compilation time for program 2 is one and one-half minutes. The maximum CPU time needed for the second program to process one point in the system space was six minutes. The amount of virtual storage needed to run the model and statistics ranged between 800K and 1,300K bytes.

CHAPTER IV

EXPERIMENTAL RESULTS AND CONCLUSIONS

In this chapter we set forth the experimental results of the computer simulations on the relative behavior of the three CPU scheduling disciplines at various points within the system space of a Product Information System. The behavior of these disciplines is measured by the performance variables: mean system response time and mean lost potential sales. As previously mentioned, the system space of a Product Information System is a three dimensional representation of the primary independent variables: CPU utilization, the paging load on the RAD subsystem, and main memory capacity. Each of these dimensions could *a priori* to experimentation influence the relative behavior of the three disciplines.

Increased CPU utilization tends to increase the queue length feeding the CPU. This in turn results in both additional processing overhead on the one hand and possibly better priority setting discrimination on the other hand for disciplines 2 and 3. The paging load, measured by the size of the queues feeding the RAD subsystem, tends to restrict the effectiveness of discipline 3 as this load increases.

For, as the paging load increases, the probability that the system will be found temporarily non-saturated decreases. If the RAD subsystem at a given point in time is saturated, then discipline 3's primary objective of increasing parallel operations between CPU processing and secondary storage processing becomes thwarted and the expended overhead on the CPU is not counterbalanced by time savings in the RAD subsystem.

As the size of the address space represented by the combined set of processing query routines grows larger than available main memory, average system response time and average potential sales loss are expected to increase. But how does this influence the relative behavior of the three scheduling disciplines? The more paging of program pages per query, the more overhead is incurred, but on the other side of the coin the priority setting process of disciplines 2 and 3 may be more effective.

Each of these three dimensions of the system space contains tradeoffs. The question we answer is how these disciplines react relative to each other at various points within the system space, the boundaries of most probable Product Information System behavior.

We consider in this study two planes cut through the system space. Each plane represents a different memory capacity. The first plane contains the Product Information Systems that have main memory sufficiency. All routines necessary for query processing are resident in main memory. The second plane contains a set of representative system conditions in which virtual storage processing will be employed because main memory is not large enough to have all these routine pages simultaneously resident in main memory. We have chosen for this second plane the main memory condition in which both the query language analyzer and the editing routine experience an average of three program paging faults per query processed. Other planes could have been chosen along the main memory capacity dimension, but as we shall demonstrate, this plane in conjunction with the first plane is adequate to demonstrate the following. First, the relative behavior of these three disciplines varies at different points within each plane. Secondly, when the two planes are compared, a virtual storage environment is shown to affect the relative ranking of the three disciplines.

Specifically, under main memory sufficiency (plane 1), when the paging load on the RAD subsystem is low, a statistically significant difference at the 0.05 alpha level is detected in the three disciplines at approximately 0.75 CPU utilization. Here discipline 3 is shown to be superior under both performance measures and discipline 1 is judged the worst. Within plane 1 at high CPU utilization (>0.9) the differentiation between the disciplines rapidly deteriorates as the paging load is increased. As the paging load is further increased, discipline 1 emerges as the best scheduling discipline. Within plane 1 under a moderate CPU utilization of approximately 0.5 and a moderate paging load discipline 2 becomes significantly inferior to disciplines 1 and 3. So within this one plane alone we see a significant difference in the relative ranking of the three disciplines.

While it was thought that discipline 2 might exhibit at some of the points in the system space simultaneously a greater mean system response time than discipline 1 but a lower mean potential sales loss, as discussed in Appendix 1, we found points at which discipline 2 was superior to discipline 1 in mean system response time. On closer inspection of the simulation results we found that discipline 2's supremacy over discipline 1 in mean system response time was attributable primarily to the fact that discipline 2 in giving higher priority to delayed query routines was inadvertently putting some of the smaller sized routines near the head of the DBMS CPU queue. This tended to decrease the average size of the queue at the same time that the discipline was attempting to reduce the variance in system response time. Another interesting result of the experiments is that discipline 2 does not always accomplish variance reduction. In fact, the opposite can be demonstrated under certain conditions. Table 9 shows an example where discipline 2 causes a significantly greater variance in system response time than discipline 1. The conclusion, of course, is that a CPU scheduling discipline, whose goal is to provide a more uniform response time for the customer base, may in reality end up providing not only a larger system response time but also one which has a greater variance.

When we compare the two planes, we find that, while a virtual storage environment increases mean system response time for all of the disciplines (as one might expect), discipline 3 is found to be superior over a wider range along both the dimensions of CPU utilization and paging load. The additional scheduling of routines per query does enhance discipline 3's effectiveness. Under this additional scheduling discipline 3 is shown to be significantly superior to discipline 1 at a lower CPU utilization and more robust against an increasing paging load at high CPU utilizations. At moderate CPU utilization (in the region of 0.5) in plane 1, discipline 2 is shown to be the most inferior discipline, while in the same region of plane 2, discipline 2 is significantly better than discipline 1. So the relative ranking of the three disciplines differs also as we move along the main memory capacity dimension.

We conclude from the experimental results that, within a data base processing environment of a Product Information System, the selection of a CPU scheduling discipline for query processing must extend beyond a FIFO discipline, since this discipline is shown to be superior only within a small area of the system space. Secondly, since significant differences in discipline performance exist within the system space, the appropriate selection of one of these disciplines for a particular Product Information System will depend on the centroid and the size of the disperion within the system space that is experienced over time by the system.

We now present the experimental results in detail in the following systematic manner. We will discuss in turn each plane of the system space. Within each plane our discussion will begin with low CPU utilization and low paging load. We will then move along the CPU utilization dimension. Then within the state of high CPU utilization we will consider points in the system space represented by increasing paging loads. This will give us extreme bounds on scheduling behavior. We will then consider the region of system space represented by moderate CPU utilization and higher paging loads.

The experimental results of the computer simulations are contained in the tables at the end of this chapter. Each table contains three sections. The first two sections, the General System Description and the Secondary Storage Configuration, present the independent variables under which the simulations on each of the three disciplines were run at a point in system space. CPU utilization in the table is affected by the combined number of instructions needed per processed query by the query language analyzer and the editing routines. The average number of editing instructions needed per data page can be determined by dividing Editor Instructions by Data Page Faults/Query. Main memory sufficiency is designated by "0" under the heading Program Page References/Routine. The paging load, measured by the Average Queue Length of the slowest devices, is determined by the number of Data Page Faults/Query and the Secondary Storage Configuration. The Secondary Storage Configuration section contains the secondary storage devices and the channels to which they are connected. The device types presented in the tables are designated as follows:

D = Drum having rotational position sensing (RPS).

F = Fast Disk (average seek time = 55 ms.
average rotation speed = 12 ms.
having rotational position sensing.)

S = Slow Disk (average seek time = 55 ms.
average rotation speed = 25 ms.
no rotational position sensing).

The section of each table entitled Experimental Results contains the mean and standard deviation of the two performance variables. These are followed by other system variables. The overall average paging rate is the average number of pages returned to main memory by the RAD subsystem when considered as a single server. The computer printouts included a utilization figure for each device. The RAD utilization of only the first two devices is shown in the tables so that the balanced state of the RAD subsystem is made manifest. These system state statistics are followed by the results of the _t_ tests performed on the performance variables.

In interpreting the results care should be taken to interpret the _relative_ value of the potential sales loss under the three disciplines. The absolute value of this performance measure is a function of the customer reaction to delays attributable to individual companies. For a given system response time this value is peculiar to each Product Information System. Also, while the standard deviation of the performance variables can meaningfully be compared within the three disciplines of each computer run, the standard deviations cannot always be compared between points in system space. The reason is that the size of the blocks of observations needed to provide statistical independence affects the standard deviation of the blocks. Yet, at any given point in system space each discipline is equally affected as to standard deviation, since the number of blocks has been made equal for each computer run. Each table also contains the results of the _t_ tests. Each table number matches a plotted point either in Figure 2 or Figure 3 and the number of each point corresponds to the hypothesis number listed in Figure 9 of Chapter III. Figures 2 and 3 of this chapter summarize graphically the results of all of the simulations.

Figure 2 represents plane 1 and Figure 3 represents plane 2. The plotted points in the figures are shown surrounded by a box on the top of which is the identification number of the point. Within each box in the top half is the ranking of the three disciplines by mean system response times.* In the lower half of each box is the ranking according to mean potential sales loss. For clarity we will introduce a convention to designate whether a significant difference was found between the pairs of the disciplines at each point under each of the performance variables. A line drawn either above or below the ranking will designate no significant difference between a pair of disciplines. For example, the ranking $\overline{3\ 2\ 1}$ for point 2 in Figure 2 states that the mean system response time of discipline 3 is the lowest, followed by that of discipline 2, followed by the highest response time of discipline 1. But the difference between these three is not significant at the 0.05 level. To convey the fact that there is only a significant difference between the first and last members of a ranking set,

---

*Ranking is expressed from left to right, from lowest to highest mean response time.

lines are drawn both above and below the ranking. For example, point 12 in Figure 3 shows $\overline{3\ \underline{1}}\underline{\ 2}$ for mean system response time. This states that a significant difference was found between disciplines 3 and 2, while a significant difference was found neither between disciplines 3 and 1, nor between disciplines 1 and 2.

Points in System Space

We now consider the points in system space at low paging load under increasing CPU utilization. These points are points 1 through 6 in Figure 2. Point 1 represents the low end of each of the three dimensions of system space. Here one can observe that the small size of the DBMS CPU queue length precludes any significant difference among the three disciplines. Since the DBMS CPU queue is so small, there is a small probability that at any given point in time this queue contains more than one virtual routine. As a result, disciplines 2 and 3 have little power of differentiation over discipline 1. We also see that a minimum level of overhead for both disciplines 2 and 3 is not enough to warrant the selection of discipline 1 over disciplines 2 and 3.

Points 2 and 3 in Figure 2 again show no significant difference in either the mean system response time or mean potential sales loss. At a CPU utilization of 0.6, the DBMS CPU queue is still too small to support a significant difference in the performance variables. Here the average DBMS CPU queue length is approximately one. It is interesting to note that while statistical significance cannot be demonstrated for points 1 through 3, the relative ranking order shows discipline 1 to be in last place for all three points.

When we consider points 4, 5, and 6, discipline 3 is significantly superior to disciplines 1 and 2 under both performance measures, except that at point 6 disciplines 1 and 2 are not significantly different for system response time. The rankings at points 4 and 5 show discipline 2 to be superior to discipline 1. Since variance reduction alone is not expected to render discipline 2 superior to discipline 1 when measured by mean system response time, the reason for this superiority must lie elsewhere. Table 5 shows the following: First, the overall average paging rate is higher for discipline 2 than discipline 1, and the DBMS CPU average queue length is smaller for discipline 2 than for discipline 1. This seemed to indicate that page requests were being generated at a faster rate by discipline 2 than by discipline 1. This could occur, if the page directory lookup routines were given higher priorities under discipline 2 and the queue size could be smaller if the smaller sized routines were favored when they were scheduled. We then looked more closely at the amount of time each of the three direct processing query routines remained in the DBMS CPU queue. Figure 1 shows these times for point 5.

	Average Time (secs.) in DBMS CPU queue for each discipline.		
	1	2	3
Query Language Analyzer	2.08	2.38	2.51
Page Directory Lookup Routine	1.80	1.10	0.20
Editing Routine	1.64	1.03	0.93
TOTAL	5.52	4.51	3.64

Figure 1: Average Time in DBMS QPU queue for direct processing query routines under each CPU scheduling discipline at point 5 in system space.

Figure 1 shows that, when we compare discipline 2 with discipline 1, the total waiting time in the queue for a query to be processed is smaller under discipline 2 than discipline 1. Secondly, the shorter routines, the page directory lookup routine and the editing routine, are favored by discipline 2, since the average waiting time is smaller for these routines under discipline 2 than discipline 1. When we compare discipline 3 with the other two disciplines we see a waiting time gradient. Discipline 3 in its priority setting favors the shortest routines, as would be expected. Why then should discipline 2 partially favor the smaller routines when its main objective is variance reduction? The answer lies in the fact that, when the DBMS CPU queue length is greater than 1, discipline 1, using FIFO scheduling, positions a routine at the end of the queue, while that same routine scheduled under discipline 2 can be scheduled at any locus in the queue. One may object by pointing out that this tends to favor each of the three types of routines equally. Let us point out, however, that without interference by a scheduling routine, the variance in the accumulated delays of the queries being processed increases as the queries move through the various stages of processing. Now the page directory lookup routines and the editing routines are processed after the QLA. Under discipline 2 some queries have greater accumulated delays after the execution of the QLA than before it. So, when the page directory lookup routines are about to be scheduled, some tend to be placed in the DBMS CPU queue closer to the head of the queue than the QLA routines. Discipline 2 favors some of the shorter and later executing routines of a query at the same time that it is attempting to reduce the variance in the system response time.

While the page directory lookup routine can always be expected to be smaller than either the QLA or the editing routines, we assume that in most Product Information Systems the QLA will be sophisticated enough so as to be at least approximately equal to and most probably larger than the editing routine. One does not anticipate the need for extensive editing for the display upon a CRT of answers to catalog type queries.

Table 6 illustrates the operation of a Product Information System under the conditions of a saturated CPU, where the CPU utilization is 1.0. Under these conditions an interesting phenomenon occurs which we term routine entrapment. While one might assume from a comparison of the mean system response times and the mean potential sales loss that discipline 3 is superior here, the standard deviation of the system response time indicates that discipline 3 may not be the best discipline to employ under CPU saturation. The extremely wide variance in the system response time of discipline 3 would certainly offend customers. This variance is so great that some queries are put into a state of "suspended animation." Some routines, processing queries, enter the DBMS CPU queue and lie dormant there. They enter the queue with a low priority only to have the cascading rush of other routines entering the queue placed in front of them. This has the effect of retarding the advance of some routines toward the head of the queue. These routines are therefore entrapped. To appreciate the extent of this entrapment, consider the following. Under discipline 2 the longest response time was 75 times greater than its mean response time. Under these system conditions, discipline 2 has certainly failed to reduce the variance in the system response time. Under discipline 3 the longest response time was 52 times greater than its mean response time. Therefore, under CPU saturation, unless an extremely large variance in response time is to be tolerated, the use of disciplines 2 and 3 are not recommended.

We now consider the points 7, 8, and 9. These points represent high CPU utilization, but not saturation, along a dimension of increasing paging load. Looking at point 7, one is immediately impressed by the rapidity at which discipline 3 fails to retain its supremacy. Points 7 and 8 show no significant difference between the three disciplines. When, however, we increase the paging load to point 9, discipline 1 emerges significantly superior to the other two disciplines in both performance measures. For discipline 3 this is understandable, since as the paging load is increased, the probability decreases that discipline 3 will find the RAD system unsaturated. For discipline 2 the giving of higher priorities to the smaller routines is counterbalanced by its delay at the RAD queues.

Lastly, within plane 1 we consider a point in the moderate CPU range under a high paging load. While one may consider this paging load moderate from the viewpoint of RAD utilization (0.4), it is high when we consider the probability of encountering empty RAD queues. There is an average of more than one query in each secondary storage device queue.

At point 10 we set the editing routine size substantially larger than the QLA in order to see how disciplines 2 and 3 perform under a most hostile sequence of routine processing. Here the bulk of the query processing comes at the end of the processing stages. Thus, there is the possibility that discipline 2 and discipline 3's objectives might be thwarted. Point 10 shows discipline 3 superior to discipline 2 and marginally but not significantly better than discipline 1.

In summing up plane 1, we recommend the use of discipline 3 except under high CPU utilization. At high CPU utilization one must take care that routine entrapment does not inordinately increase the variance in the performance variables. Also, at high CPU utilization with even slightly high paging loads, discipline 1 is to be preferred.

We now examine the experimental results for the points in plane 2 of the system space. These are summarized in Figure 3. Again we first consider points along an increasing CPU utilization dimension at low paging loads. Table 11, representing point 11, considers a low CPU utilization of approximately 25%. While no significant difference is detected here, a significant difference in the disciplines is manifest when we examine the region of CPU utilization slightly greater than 60%. Here at point 12, discipline 3 is significantly superior to discipline 1 for system response time. Increasing the CPU utilization still further to approximately 90% to point 13, discipline 3 is superior under both performance measures. As one moves along the CPU dimension at a low paging load, the relative merits of discipline 3 become apparent.

We now consider points at high CPU utilization with increasing paging load. These are the points 13, 14, and 15. At point 14, given a moderate paging load, discipline 3 is superior under both performance variables. The independent variables in point 15 are set similar to point 14 with the exception that again the editing routine is made larger than the QLA to see the effect on the system. The paging load for discipline 1 is greater under these conditions at point 15 than point 14. Yet, even at this higher paging load, discipline 3 is superior to discipline 1. Point 16 considers the system condition of moderate CPU utilization and moderately high paging load. Here both discipline 2 and discipline 3 are found to be significantly superior to discipline 1.

We now consider three points in plane 2 at moderate CPU utilization under heavier paging loads. Points 17 and 18 are again set up with a smaller QLA than the editing routine to see performance under harsh discriminatory conditions. At point 17 no significant difference was detected. Point 18 moves closer to the middle of the CPU utilization range under an even heavier paging load. Here the additional increase in CPU utilization more than offsets the increased paging load and favors discipline 2 and discipline 3 over discipline 1. Note that at point 17 the average DBMS CPU queue length for discipline 3 is 0.48, while at point 18 it is 1.6. Point 19 represents a moderate CPU utilization but a heavier paging load. Here the additional increase in paging load is great enough to prevent and the DBMS CPU queue is too small to effect a significant difference between the disciplines. Considering the points within plane 2, the data supports the conclusion that discipline 3 can be shown to be significantly superior to the other disciplines over a wider area of system space. The caution against using discipline 3 in a virtual storage environment at CPU saturation is still valid, although such a point is not plotted in Figure 3.

When we compare the behavior of discipline 3 in the two planes, we see that a virtual storage environment, if properly handled, can enhance the scheduling capability of a CPU scheduling discipline. To be specific, in a virtual storage environment discipline 3 has been shown to be superior to discipline 1 at a lower CPU utilization than under the conditions of main memory sufficiency (compare the rankings at points 3 and 13). Also under high CPU utilization discipline 3 is more robust against an increasing paging load (compare points 8 and 9 with points 14, 15, and 16).

In conclusion, the Data Base Administrator, when developing future Product Information Systems, must take cognizance of the effect of the CPU scheduling discipline which processes the routines handling queries to the data base. To say that such critical parts of the operating system like the CPU scheduling algorithm are "off limits" or outside the domain of the responsibility of the Data Base Administrator is nonsense. Such restrictions can only produce suboptimal system performance at best. We are not advocating that data base personnel become enmeshed in all aspects of operating systems, but that, as this study has demonstrated, a blind acceptance of a CPU scheduling algorithm to operate under all system conditions will prove to be a poor decision.

System response time and potential sales loss are both a combined function of not only the logical and physical storage and accessing of data, but also the scheduling of the routines which generate the requests for these accesses. These factors, influencing the performance variables, cannot be easily separated. Physical storage affects the accessing rate and the accessing rate is controlled in part by the CPU scheduling algorithm. Optimum storage allocation is in turn a function of the accessing rate. Thus we have a closed circle of interdependencies.

This study has shown that for a Product Information System discipline 3 provides the widest range of optimal behavior among the three disciplines studied, as measured by both mean system response time and mean lost potential sales, although it is not superior over the entire system space and even inferior to discipline 1 within some sectors of the system space. This study opens up avenues for future research into the areas where data base management systems and operating systems interact. Can some other CPU scheduling discipline provide a better response time or a smaller potential sales loss for a Product Information System or can it provide a wider range of superiority than discipline 3?

Looking to the future we know that CPU utilization and paging loads can be monitored by both system software monitors and stand-alone hardware monitors. We look forward to the day when the monitoring of such information is used in Product Information Systems as a basis for determining which CPU scheduling discipline should be operable at a given point in time at a given point in system space.

CPU UTILIZATION
Figure 2 Plane 1

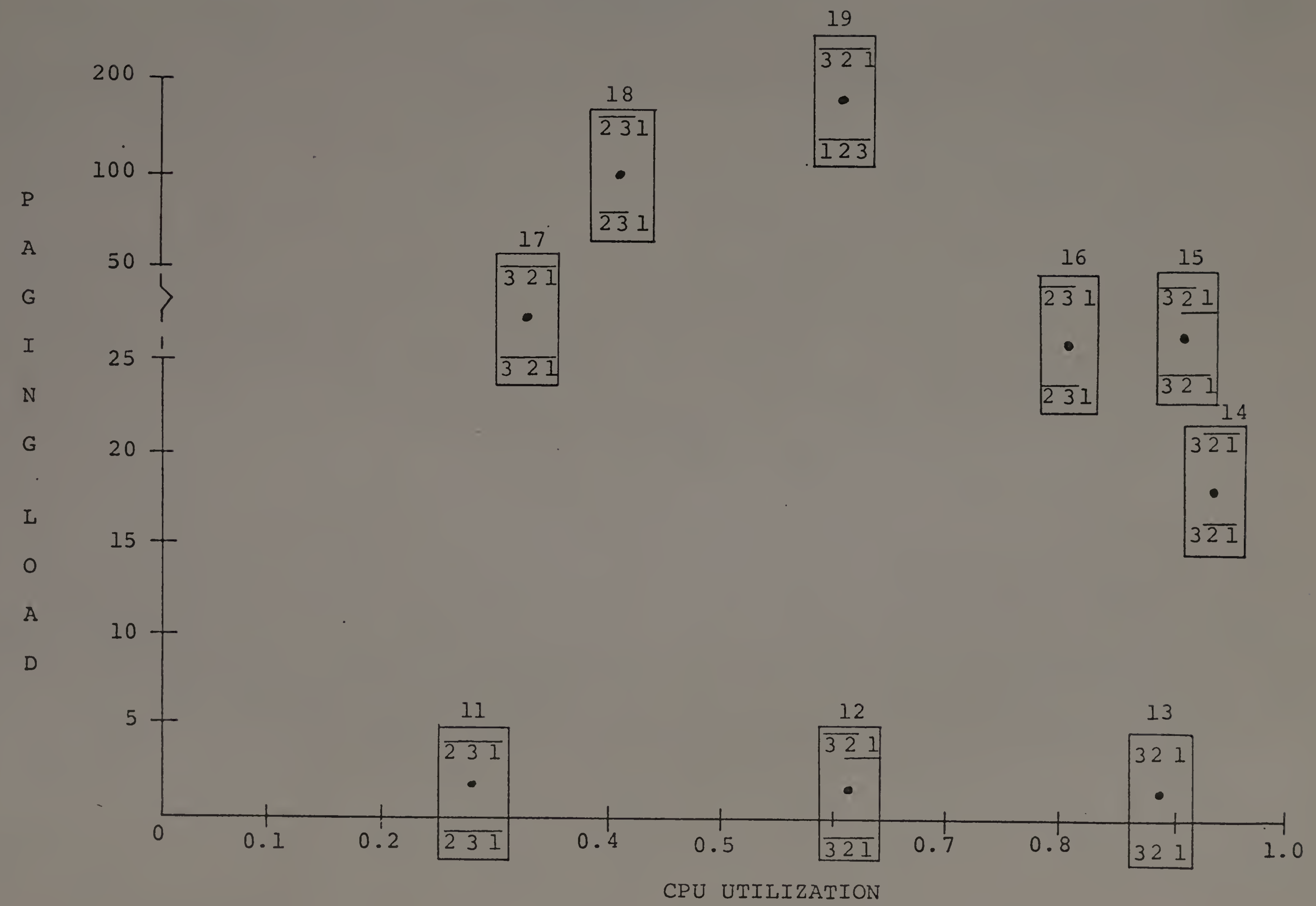


Figure 3 Plane 2

TABLE 1: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	32,000
	Editor Instructions	=	50,000
Program Page References/Routine		=	0
Data Page Faults/Query		=	10

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	12.62	12.60	12.60
S.D. of Response Time	4.52	4.46	4.55
Mean Pot. Sales Loss	0.156	0.156	0.157
S.D. of Loss	0.120	0.119	0.123
CPU Utilization	0.156	0.157	0.163
Ave DBMS CPU Queue Length	0.03	0.03	0.03
Overall Ave Paging Rate	18.31	18.58	18.65
Channel 1 Utilization	0.118	0.119	0.124
RAD 1 Utilization	0.076	0.076	0.079
RAD 2 Utilization	0.076	0.077	0.079
Ave Queue Length (slowest devices)	0.9	0.9	0.9

		Z-score	Signif. Level
Response Time	(Discip. 1 & 2)	0.09	n.s.
Response Time	(Discip. 1 & 3)	0.12	n.s.
Response Time	(Discip. 2 & 3)	0.03	n.s.
Lost Potential Sales	(Discip. 1 & 2)	0.09	n.s.
Lost Potential Sales	(Discip. 1 & 3)	0.15	n.s.
Lost Potential Sales	(Discip. 2 & 3)	0.23	n.s.

TABLE 2: SIMULATION DATA

General System Description:

DBMS Work Load: QLA Instructions = 100,000

Editor Instructions = 125,000

Program Page References/Routine = 0

Data Page Faults/Query = 25

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:

	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	14.20	14.09	14.06
S.D. of Response Time	1.85	1.60	1.69
Mean Pot. Sales Loss	0.172	0.168	0.167
S.D. of Loss	0.051	0.042	0.043
CPU Utilization	0.398	0.396	0.398
Ave DBMS CPU Queue Length	0.32	0.27	0.25
Overall Ave Paging Rate	44.3	43.2	43.7
Channel 1 Utilization	0.285	0.281	0.285
RAD 1 Utilization	0.191	0.189	0.191
RAD 2 Utilization	0.191	0.191	0.191
Ave Queue Length (slowest devices)	6.8	7.3	6.0

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	0.574	n.s.
Response time	(Discip. 1 & 3)	0.793	n.s.
Response time	(Discip. 2 & 3)	0.183	n.s.
Lost Potential Sales	(Discip. 1 & 2)	0.79	n.s.
Lost Potential Sales	(Discip. 1 & 3)	0.94	n.s.
Lost Potential Sales	(Discip. 2 & 3)	0.08	n.s.

TABLE 3: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	250,000
	Editor Instructions	=	100,000
Program Page References/Routine		=	0
Data Page Faults/Query		=	10

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	13.46	13.36	13.37
S.D. of Response Time	1.80	1.66	1.66
Mean Pot. Sales Loss	0.153	0.150	0.151
S.D. of Loss	0.045	0.040	0.040
CPU Utilization	0.617	0.638	0.627
Ave DBMS CPU Queue Length	1.06	0.99	0.81
Overall Ave Paging Rate	18.3	18.8	18.5
Channel 1 Utilization	0.115	0.121	0.119
RAD 1 Utilization	0.073	0.077	0.076
RAD 2 Utilization	0.073	0.077	0.076
Ave Queue Length (slowest devices)	0.7	1.0	0.8

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	0.60	n.s.
Response time	(Discip. 1 & 3)	0.50	n.s.
Response time	(Discip. 2 & 3)	0.10	n.s.
Lost Potential Sales	(Discip. 1 & 2)	0.74	n.s.
Lost Potential Sales	(Discip. 1 & 3)	0.62	n.s.
Lost Potential Sales	(Discip. 2 & 3)	0.10	n.s.

TABLE 4: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	375,000
	Editor Instructions	=	75,000
Program Page References/Routine		=	0
Data Page Faults/Query		=	10

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:

	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	14.72	14.32	13.89
S.D. of Response Time	4.68	4.75	4.66
Mean Pot. Sales Loss	0.208	0.198	0.187
S.D. of Loss	0.145	0.147	0.139
CPU Utilization	0.782	0.781	0.788
Ave DBMS CPU Queue Length	2.8	2.4	2.00
Overall Ave Paging Rate	17.93	17.92	18.25
Channel 1 Utilization	0.113	0.115	0.116
RAD 1 Utilization	0.073	0.073	0.074
RAD 2 Utilization	0.074	0.073	0.074
Ave Queue Length (slowest devices)	0.9	0.8	0.8

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	2.22	0.05
Response time	(Discip. 1 & 3)	4.63	0.01
Response time	(Discip. 2 & 3)	2.43	0.05
Lost Potential Sales	(Discip. 1 & 2)	1.76	n.s.
Lost Potential Sales	(Discip. 1 & 3)	3.83	0.01
Lost Potential Sales	(Discip. 2 & 3)	2.08	0.05

TABLE 5: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	500,000
	Editor Instructions	=	50,000
Program Page References/Routine		=	0
Data Page Faults/Query		=	10

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:

	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	18.24	17.31	16.17
S.D. of Response Time	4.02	2.53	2.33
Mean Pot. Sales Loss	0.311	0.269	0.231
S.D. of Loss	0.154	0.091	0.079
CPU Utilization	0.883	0.909	0.918
Ave DBMS CPU Queue Length	7.99	6.57	5.06
Overall Ave Paging Rate	16.78	17.17	17.55
Channel 1 Utilization	0.104	0.108	0.111
RAD 1 Utilization	0.067	0.068	0.070
RAD 2 Utilization	0.067	0.068	0.070
Ave Queue Length (slowest devices)	1.0	1.0	0.8

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	3.02	0.01
Response time	(Discip. 1 & 3)	7.21	0.01
Response time	(Discip. 2 & 3)	5.17	0.01
Lost Potential Sales	(Discip. 1 & 2)	3.57	0.01
Lost Potential Sales	(Discip. 1 & 3)	7.55	0.01
Lost Potential Sales	(Discip. 2 & 3)	5.07	0.01

TABLE 6: SIMULATION DATA

General System Description:

DBMS Work Load: QLA Instructions = 1,000,000

Editor Instructions = 400,000

Program Page References/Routine = 0

Data Page Faults/Query = 10

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:

	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	97.67	87.70	62.11
S.D. of Response Time	9.83	105.94	73.16
Mean Pot. Sales Loss	0.999	0.991	0.866
S.D. of Loss	0.00004	0.02535	0.15792
CPU Utilization	1.0	1.0	1.0
Ave DBMS CPU Queue Length	58.62	58.48	59.42
Overall Ave Paging Rate	7.5	7.4	7.5
Channel 1 Utilization	0.048	0.047	0.047
RAD 1 Utilization	0.030	0.030	0.029
RAD 2 Utilization	0.030	0.030	0.029
Ave Queue Length (slowest devices)	0.3	0.5	0.3

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	1.21	n.s.
Response time	(Discip. 1 & 3)	6.11	0.01
Response time	(Discip. 2 & 3)	2.47	0.05
Lost Potential Sales	(Discip. 1 & 2)	4.31	0.01
Lost Potential Sales	(Discip. 1 & 3)	10.82	0.01
Lost Potential Sales	(Discip. 2 & 3)	10.12	0.01

TABLE 7: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	500,000
	Editor Instructions	=	125,000
Program Page References/Routine		=	0
Data Page Faults/Query		=	25

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:

	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	21.56	22.08	21.96
S.D. of Response Time	4.26	4.36	4.30
Mean Pot. Sales Loss	0.438	0.455	0.449
S.D. of Loss	0.171	0.169	0.161
CPU Utilization	0.960	0.960	0.962
Ave DBMS CPU Queue Length	10.66	11.46	10.67
Overall Ave Paging Rate	38.56	38.81	39.32
Channel 1 Utilization	0.256	0.253	0.257
RAD 1 Utilization	0.170	0.167	0.169
RAD 2 Utilization	0.166	0.168	0.170
Ave Queue Length (slowest devices)	6.7	7.9	5.9

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	1.38	n.s.
Response time	(Discip. 1 & 3)	1.09	n.s.
Response time	(Discip. 2 & 3)	0.34	n.s.
Lost Potential Sales	(Discip. 1 & 2)	1.10	n.s.
Lost Potential Sales	(Discip. 1 & 3)	0.73	n.s.
Lost Potential Sales	(Discip. 2 & 3)	0.45	n.s.

TABLE 8: SIMULATION DATA

General System Description:

DBMS Work Load: QLA Instructions	=	500,000
Editor Instructions	=	150,000
Program Page References/Routine	=	0
Data Page Faults/Query	=	30

Secondary Storage Configuration:

Devices	1	2	3	4	5
Connected to Channel	1	1	1	2	2
Device Types	D	F	S	S	S

Experimental Results:	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	24.16	23.71	23.19
S.D. of Response Time	5.52	4.63	4.48
Mean Pot. Sales Loss	0.535	0.512	0.500
S.D. of Loss	0.202	0.178	0.174
CPU Utilization	0.973	0.966	0.979
Ave DBMS CPU Queue Length	13.97	13.59	13.77
Overall Ave Paging Rate	45.43	45.11	46.03
Channel 1 Utilization	0.257	0.247	0.254
RAD 1 Utilization	0.287	0.277	0.283
RAD 2 Utilization	0.287	0.277	0.284
Ave Queue Length (slowest devices)	14.3	13.0	13.6

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	0.79	n.s.
Response time	(Discip. 1 & 3)	1.68	n.s.
Response time	(Discip. 2 & 3)	1.13	n.s.
Lost Potential Sales	(Discip. 1 & 2)	0.76	n.s.
Lost Potential Sales	(Discip. 1 & 3)	1.65	n.s.
Lost Potential Sales	(Discip. 2 & 3)	1.08	n.s.

TABLE 9: SIMULATION DATA

General System Description:

DBMS Work Load: QLA Instructions = 750,000

Editor Instructions = 100,000

Program Page References/Routine = 0

Data Page Faults/Query = 20

Secondary Storage Configuration:

Devices	1	2	3	4
Connected to Channel	1	1	1	1
Device Types	F	S	S	S

Experimental Results:	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	42.44	48.73	47.57
S.D. of Response Time	7.42	9.52	9.21
Mean Pot. Sales Loss	0.944	0.970	0.967
S.D. of Loss	0.072	0.053	0.058
CPU Utilization	0.991	0.909	0.929
Ave DBMS CPU Queue Length	24.8	31.0	28.9
Overall Ave Paging Rate	23.2	22.2	22.5
Channel 1 Utilization	0.605	0.554	0.602
RAD 1 Utilization	0.542	0.483	0.529
RAD 2 Utilization	0.544	0.483	0.529
Ave Queue Length (slowest devices)	46.2	45.8	49.7

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	6.34	0.01
Response time	(Discip. 1 & 3)	5.37	0.01
Response time	(Discip. 2 & 3)	1.17	n.s.
Lost Potential Sales	(Discip. 1 & 2)	3.62	0.01
Lost Potential Sales	(Discip. 1 & 3)	3.11	0.01
Lost Potential Sales	(Discip. 2 & 3)	0.48	n.s.

TABLE 10: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	32,000
	Editor Instructions	=	250,000
Program Page References/Routine		=	0
Data Page Faults/Query		=	50

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	19.46	19.89	18.90
S.D. of Response Time	7.11	7.55	6.42
Mean Pot. Sales Loss	0.364	0.377	0.347
S.D. of Loss	0.236	0.250	0.223
CPU Utilization	0.457	0.467	0.452
Ave DBMS CPU Queue Length	0.71	0.68	0.52
Overall Ave Paging Rate	78.23	80.90	77.45
Channel 1 Utilization	0.397	0.562	0.526
RAD 1 Utilization	0.393	0.411	0.382
RAD 2 Utilization	0.396	0.411	0.383
Ave Queue Length (slowest devices)	60.0	64.6	45.3

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	1.19	n.s.
Response time	(Discip. 1 & 3)	1.56	n.s.
Response time	(Discip. 2 & 3)	2.68	0.01
Lost Potential Sales	(Discip. 1 & 2)	1.03	n.s.
Lost Potential Sales	(Discip. 1 & 3)	1.43	n.s.
Lost Potential Sales	(Discip. 2 & 3)	2.38	0.05

TABLE 11: SIMULATION DATA

General System Description:

DBMS Work Load: QLA Instructions = 100,000

Editor Instructions = 50,000

Program Page References/Routine = 3

Data Page Faults/Query = 10

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:

	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	13.25	13.14	13.17
S.D. of Response Time	10.07	10.22	10.34
Mean Pot. Sales Loss	0.209	0.206	0.206
S.D. of Loss	0.260	0.180	0.250
CPU Utilization	0.268	0.273	0.285
Ave DBMS CPU Queue Length	0.12	0.11	0.13
Overall Ave Paging Rate	28.75	28.53	28.99
Channel 1 Utilization	0.185	0.181	0.190
RAD 1 Utilization	0.119	0.118	0.123
RAD 2 Utilization	0.119	0.119	0.123
Ave Queue Length (slowest devices)	1.4	1.3	1.4

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	0.63	n.s.
Response time	(Discip. 1 & 3)	0.46	n.s.
Response time	(Discip. 2 & 3)	0.17	n.s.
Lost Potential Sales	(Discip. 1 & 2)	0.68	n.s.
Lost Potential Sales	(Discip. 1 & 3)	0.69	n.s.
Lost Potential Sales	(Discip. 2 & 3)	0.00	n.s.

TABLE 12: SIMULATION DATA

General System Description:

DBMS Work Load: QLA Instructions = 250,000

Editor Instructions = 100,000

Program Page References/Routine = 3

Data Page Faults/Query = 10

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	14.04	13.85	13.67
S.D. of Response Time	4.53	4.61	4.61
Mean Pot. Sales Loss	0.190	0.186	0.182
S.D. of Loss	0.135	0.138	0.135
CPU Utilization	0.614	0.610	0.606
Ave DBMS CPU Queue Length	1.3	1.0	0.8
Overall Ave Paging Rate	28.6	28.3	28.1
Channel 1 Utilization	0.185	0.180	0.177
RAD 1 Utilization	0.120	0.117	0.115
RAD 2 Utilization	0.120	0.116	0.115
Ave Queue Length (slowest devices)	1.6	1.3	1.3

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	1.08	n.s.
Response time	(Discip. 1 & 3)	2.09	0.05
Response time	(Discip. 2 & 3)	1.04	n.s.
Lost Potential Sales	(Discip. 1 & 2)	0.81	n.s.
Lost Potential Sales	(Discip. 1 & 3)	1.55	n.s.
Lost Potential Sales	(Discip. 2 & 3)	0.74	n.s.

TABLE 13: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	500,000
	Editor Instructions	=	50,000
Program Page References/Routine		=	3
Data Page Faults/Query		=	10

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	19.16	17.25	16.32
S.D. of Response Time	4.35	3.72	5.38
Mean Pot. Sales Loss	0.354	0.288	0.266
S.D. of Loss	0.110	0.130	0.160
CPU Utilization	0.883	0.911	0.909
Ave DBMS CPU Queue Length	8.3	6.1	4.4
Overall Ave Paging Rate	26.4	26.8	26.9
Channel 1 Utilization	0.167	0.172	0.170
RAD 1 Utilization	0.108	0.111	0.109
RAD 2 Utilization	0.108	0.111	0.110
Ave Queue Length (slowest devices)	1.3	1.4	1.2

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	5.75	0.01
Response time	(Discip. 1 & 3)	8.76	0.01
Response time	(Discip. 2 & 3)	4.68	0.01
Lost Potential Sales	(Discip. 1 & 2)	7.69	0.01
Lost Potential Sales	(Discip. 1 & 3)	10.71	0.01
Lost Potential Sales	(Discip. 2 & 3)	4.10	0.01

TABLE 14: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	500,000
	Editor Instructions	=	150,000
Program Page References/Routine		=	3
Data Page Faults/Query		=	30

Secondary Storage Configuration:

Devices	1	2	3	4	5
Connected to Channel	1	1	1	2	2
Device Types	D	F	S	S	S

Experimental Results:

	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	25.71	24.96	23.21
S.D. of Response Time	5.59	3.31	2.95
Mean Pot. Sales Loss	0.594	0.580	0.507
S.D. of Loss	0.198	0.136	0.125
CPU Utilization	0.944	0.970	0.977
Ave DBMS CPU Queue Length	14.3	13.1	12.2
Overall Ave Paging Rate	52.5	53.1	53.3
Channel 1 Utilization	0.288	0.293	0.294
RAD 1 Utilization	0.325	0.331	0.330
RAD 2 Utilization	0.325	0.331	0.330
Ave Queue Length (slowest devices)	17.9	17.9	15.6

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	1.24	n.s.
Response time	(Discip. 1 & 3)	4.32	0.01
Response time	(Discip. 2 & 3)	4.25	0.01
Lost Potential Sales	(Discip. 1 & 2)	0.64	n.s.
Lost Potential Sales	(Discip. 1 & 3)	4.00	0.01
Lost Potential Sales	(Discip. 2 & 3)	4.20	0.01

TABLE 15: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	150,000
	Editor Instructions	=	500,000
Program Page References/Routine		=	3
Data Page Faults/Query		=	30

Secondary Storage Configuration:

Devices	1	2	3	4	5
Connected to Channel	1	1	1	2	2
Device Types	D	F	S	S	S

Experimental Results:	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	26.35	25.23	25.07
S.D. of Response Time	5.53	5.01	3.95
Mean Pot. Sales Loss	0.616	0.580	0.579
S.D. of Loss	0.186	0.177	0.155
CPU Utilization	0.920	0.949	0.958
Ave DBMS CPU Queue Length	13.00	14.15	13.57
Overall Ave Paging Rate	51.66	52.27	53.07
Channel 1 Utilization	0.285	0.291	0.292
RAD 1 Utilization	0.323	0.330	0.328
RAD 2 Utilization	0.328	0.332	0.329
Ave Queue Length (slowest devices)	27.6	20.3	18.5

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	1.74	n.s.
Response time	(Discip. 1 & 3)	2.00	0.05
Response time	(Discip. 2 & 3)	0.26	n.s.
Lost Potential Sales	(Discip. 1 & 2)	1.65	n.s.
Lost Potential Sales	(Discip. 1 & 3)	1.58	n.s.
Lost Potential Sales	(Discip. 2 & 3)	0.01	n.s.

TABLE 16: SIMULATION DATA

General System Description:

DBMS Work Load: QLA Instructions = 500,000

Editor Instructions = 50,000

Program Page References/Routine = 3

Data Page Faults/Query = 40

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:

	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	23.91	22.27	22.28
S.D. of Response Time	4.26	3.13	3.20
Mean Pot. Sales Loss	0.533	0.466	0.467
S.D. of Loss	0.172	0.132	0.136
CPU Utilization	0.801	0.836	0.838
Ave DBMS CPU Queue Length	7.0	6.8	6.6
Overall Ave Paging Rate	67.4	68.8	69.37
Channel 1 Utilization	0.448	0.459	4.64
RAD 1 Utilization	0.316	0.321	0.324
RAD 2 Utilization	0.316	0.322	0.325
Ave Queue Length (slowest devices)	27.1	24.9	27.4

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	4.54	0.01
Response time	(Discip. 1 & 3)	4.18	0.01
Response time	(Discip. 2 & 3)	0.03	n.s.
Lost Potential Sales	(Discip. 1 & 2)	4.51	0.01
Lost Potential Sales	(Discip. 1 & 3)	4.12	0.01
Lost Potential Sales	(Discip. 2 & 3)	0.08	n.s.

TABLE 17: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	32,000
	Editor Instructions	=	150,000
Program Page References/Routine		=	3
Data Page Faults/Query		=	30

Secondary Storage Configuration:

Devices	1	2	3	4	5
Connected to Channel	1	1	1	2	2
Device Types	D	F	S	S	S

Experimental Results:	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	15.52	15.44	15.42
S.D.of Response Time	5.49	5.15	5.02
Mean Pot. Sales Loss	0.238	0.232	0.232
S.D. of Loss	0.178	0.165	0.161
CPU Utilization	0.317	0.320	0.322
Ave DBMS CPU Queue Length	0.58	0.51	0.48
Overall Ave Paging Rate	61.2	60.7	60.7
Channel 1 Utilization	0.340	0.337	0.340
RAD 1 Utilization	0.385	0.381	0.389
RAD 2 Utilization	0.385	0.382	0.389
Ave Queue Length (slowest devices)	31.9	27.8	29.7

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	0.38	n.s.
Response time	(Discip. 1 & 3)	0.48	n.s.
Response time	(Discip. 2 & 3)	0.10	n.s.
Lost Potential Sales	(Discip. 1 & 2)	0.78	n.s.
Lost Potential Sales	(Discip. 1 & 3)	0.87	n.s.
Lost Potential Sales	(Discip. 2 & 3)	0.08	n.s.

TABLE 18: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	32,000
	Editor Instructions	=	250,000
Program Page References/Routine		=	3
Data Base Faults/Query		=	50

Secondary Storage Configuration:

Devices	1	2	3	4	5	6	7	8	9	10
Connected to Channel	1	1	1	1	1	1	2	2	2	2
Device Types	D	F	F	F	S	S	S	S	S	S

Experimental Results:	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	26.55	24.86	25.90
S.D. of Response Time	6.88	5.45	6.09
Mean Pot. Sales Loss	0.612	0.560	0.594
S.D. of Loss	0.221	0.197	0.202
CPU Utilization	0.413	0.430	0.433
Ave DBMS CPU Queue Length	1.9	1.5	1.6
Overall Ave Paging Rate	80.7	82.2	82.2
Channel 1 Utilization	0.558	0.560	0.553
RAD 1 Utilization	0.400	0.406	0.404
RAD 2 Utilization	0.405	0.405	0.404
Ave Queue Length (slowest devices)	98.0	96.2	111.3

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	2.47	0.05
Response time	(Discip. 1 & 3)	0.96	n.s.
Response time	(Discip. 2 & 3)	1.79	n.s.
Lost Potential Sales	(Discip. 1 & 2)	2.17	0.05
Lost Potential Sales	(Discip. 1 & 3)	0.75	n.s.
Lost Potential Sales	(Discip. 2 & 3)	1.62	n.s.

TABLE 19: SIMULATION DATA

General System Description:

DBMS Work Load:	QLA Instructions	=	500,000
	Editor Instructions	=	100,000
Program Page References/Routine		=	3
Data Page Faults/Query		=	20

Secondary Storage Configuration:

Devices	1	2	3	4
Connected to Channel	1	1	1	1
Device Types	F	S	S	S

Experimental Results:	Discipline 1	Discipline 2	Discipline 3
Mean Response Time	55.08	55.07	54.75
S.D. of Response Time	12.17	11.72	11.62
Mean Pot. Sales Loss	0.981	0.982	0.984
S.D. of Loss	0.043	0.042	0.031
CPU Utilization	0.598	0.605	0.592
Ave DBMS CPU Queue Length	2.8	5.3	3.1
Overall Ave Paging Rate	26.4	26.4	26.4
Channel 1 Utilization	0.691	0.688	0.692
RAD 1 Utilization	0.645	0.642	0.646
RAD 2 Utilization	0.647	0.643	0.649
Ave Queue Length (slowest devices)	185.3	169.6	205.5

		Z-score	Signif. Level
Response time	(Discip. 1 & 2)	0.00	n.s.
Response time	(Discip. 1 & 3)	0.24	n.s.
Response time	(Discip. 2 & 3)	0.25	n.s.
Lost Potential Sales	(Discip. 1 & 2)	0.22	n.s.
Lost Potential Sales	(Discip. 1 & 3)	1.09	n.s.
Lost Potential Sales	(Discip. 2 & 3)	0.84	n.s.

LITERATURE CITED

LITERATURE CITED

Abate, Joseph; Dubner, H.; and Weinberg, S.B. 1968. Queueing analysis of the IBM 2314 disk storage facility. JACM. 15: 577-589.

Adiri, I.; Hofri, M.; and Yadin, M. 1973. A multiprogramming queue. JACM. 20: 589-603.

Altshuler, G., and Plagman, B. 1974. User system interface within the context of an integrated corporate data base. Proceedings of the AFIPS Conference. 43: 27-33.

Anacker, Wilhelm, and Wang, Chu Ping. 1967. Performance evaluation of computing systems with memory hierarchies. IEEE Transactions on Electronic Computers. EC-16: 764-773.

ANSI/X3/SPARC Study Group on Data Base Management Systems Interim Report. 1975. FDT--SIGMOD, The Special Interest Group on Management of Data. 7, 2: 12.

Arora, S.R., and Gallo, A. 1971. Optimal sizing, loading, and reloading in a multi-level memory hierarchy system. Proceedings of the Spring Joint Computer Conference. Pp. 337-343.

Arora, S.R., and Gallo, A. 1973. Optimization of static loading and sizing of multilevel memory systems. JACM. 20: 307-319.

Arora, S.R., and Jain, G.P. 1971. Drum queueing model. Proceedings of the Spring Joint Computer Conference. Pp. 319-324.

Asten, K.J. 1973. Data Communications for Business Information Systems. P. 268. New York: MacMillan Co.

Auerbach. 1973. Users Notebook. Sec. 715.4239.040, p. 1. Philadelphia: Auerbach Publishers, Inc.

Auerbach. 1974a. Data Processing Manual. Sec. 6-0201, p.4. Philadelphia: Auerbach Publishers, Inc.

Auerbach. 1974b. The synchronous data link control (SDLC). Auerbach Reporter. 2.

Bachman, Charles W. 1965. Software for random access processing. Datamation. (April.) Pp. 36-41.

Bachman, Charles W. 1969. Data base management: The keystone of applied system architecture in Critical Factors in Data Management. Edited by Fred Gruenberger. P. 33. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Bachman, Charles W. 1972. The evolution of storage structures. CACM. 15: 629.

Bacon, M.D., and Bull, G.M. 1973. Data Transmission. Pp. 75-76. New York: MacDonald/American Elsevier, Inc.

Bayer, R., and McCreight, E. 1972. Organization and maintenance of large ordered indexes. Acta Informatica. 1: 173-189.

Belady, L.A.; Nelson, R.A.; and Shedler, G.S. 1969. An anomaly in space-time characteristics of certain programs running in a paging machine. CACM. 12: 349-353.

Berztiss, A.T. 1975. Data Structures: Theory and Practice, 2d ed. New York: Academic Press.

Blatny, J.; Clark, S.R.; and Rourke, R.A. 1972. On the optimization of performance of time-sharing systems by simulation. CACM. 15: 411-420.

Blumenthal, Sherman C. 1969. Management Information Systems: A Framework for Planning and Development. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Boies, S.J. 1974. User behavior on an interactive computer system. IBM Systems Journal. 13: 3-18.

Boyd, D.L., and Epley, D.L. 1971. A simple model for multiple-resource allocation in operating systems. Proceedings of the Symposium on Computers and Automation. Pp. 277-292. Brooklyn, New York.

Brawn, B. 1970. SCRIPGEN: A Program to Generate TSS/360 User Session Scripts for System Evaluation. Technical Report 2808, T.J. Watson Research Center, IBM Corp.

Burge, William H., and Konheim, Alan G. 1971. An accessing model. JACM. 18: 400-404.

Buzen, Jeffrey. 1973. Computational algorithms for closed queueing networks with exponential servers. CACM. 16: 527-531.

Buzen, Jeffrey. 1976. 'Bottleneck principle' balances system load properly. Computerworld. April 26. P. S/6.

Cagan, Carl. 1973. Data Management Systems. P. 59. Los Angeles, California: Melville Publishing Co.

Canning, R.G. 1972. The data administrator's function. Data Base Analyzer. 10: 12-13.

Carbonell, Jaime, et al. 1968. On the psychological importance of time in a time sharing system. Human Factors. 10: 137.

Chen, P.P.S. 1973a. Optimal File Allocation. PhD. Thesis. Harvard University.

Chen, P.P.S. 1973b. Optimal file allocation in multi-level storage systems. Pp. 277-282. Proceedings of AFIPS Conference. New Jersey. AFIPS Press.

Chow, C.K. 1974. On optimization of storage hierarchies. IBM Journal of Research and Development. Pp. 194-203.

Chu, Wesley M. 1969. Optimal file allocation in a multiple computer system. IEEE Transactions on Computers. C-18: 885-889.

Ciampi, Peter L. 1972. Synthetic programs for real-time benchmark testing. Proceedings of the Summer Simulation Conference. Pp. 228-231.

CODASYL Data Base Task Group Report. April, 1971. New York: ACM.

CODASYL Systems Committee. 1971. Feature Analysis of Generalized Data Base Management Systems: Technical Report. New York: ACM.

Codd, E.F. 1970. A relational model of data for large shared data banks. CACM. 13: 377-387.

Coffman, E.G., Jr. 1967. Studying multiprogramming systems. Datamation. Pp. 47-54.

Coffman, E.G., Jr. 1969. Analysis of a drum input/output queue under scheduled operation in a paged computer system. JACM. 16: 73-89.

Coffman, E.G., Jr., and Eve, J. 1970. File structures using hashing functions. CACM. 13: 427-436.

Coffman, E.G., Jr., and Wood, R.C. 1966. Interarrival statistics for time sharing systems. CACM. 9:500-503.

Cohen, Leo. 1973. Data Base Management Systems: A Critical and Comparative Analysis. QED Information Sciences.

Computerworld. 1974. Two orders of magnitude: magnetic recording density seen rising. April 17. P. 37.

Computerworld. 1975a. College conferencing net to cost $2/hour per CPU. June 25. P. 27.

Computerworld. 1975b. DBMS. September 24. P. S/1.

Computerworld. 1975c. Shoppers order catalogue items directly from system. November 12. P. 29.

Considine, James P., and Weis, Allan H. 1969. Establishment and maintenance of a storage hierarchy for an on-line data base under TSS/360. Proceedings of the Fall Joint Computer Conference. Pp. 433-440.

Conti, C.J.; Gibson, D.H.; and Pitkowsky, S.H. 1968. Structural aspects of system/360 model 85 I--general organization. IBM Systems Journal. 7: 1-14.

Conway, R.W. 1963. Some tactical problems in digital simulation. Management Science. 10: 49.

Costis, Harry. 1972. Statistics for Business. Pp. 526-527. Columbus, Ohio: Charles E. Merrill Publ. Co.

Couger, J. Daniel, and McFadden, Fred R. 1975. Introduction to Computer Based Information Systems. P. 634. New York: John Wiley and Sons.

Date, C.J. 1975. An Introduction to Database Systems. Reading, Massachusetts: Addison-Wesley Publ. Co.

Delbrouck, L. 1970. A feedback queueing system with batch arrivals, bulk service, and queue dependent service time. JACM. 17: 314-323.

Denning, Peter J. 1967. Effects of scheduling on file memory operations. _Proceedings of the Spring Joint Computer Conference_. P. 11.

Denning, Peter J. 1968. A statistical model for console behavior in multiuser computers. _CACM_. 11: 605-612.

Denning, Peter J. 1972. A note on paging drum efficiency. _Computing Surveys_. 4: 2.

Dewan, Prem Bhushan; Donaghey, C.E.; and Wyatt, J.B. 1972. OSSL--A specialized language for simulating computer systems. _Spring Joint Computer Conference_. P. 799-814.

Dixon, W.J., and Massey, F.J., Jr. 1957. _Introduction to Statistical Analysis_. Pp. 124-127. New York: McGraw-Hill Book Co.

Dodd, George G. 1969. Elements of data management systems. _Computing Surveys_. 1: 117-133.

Donovan, J.J. 1972. _Systems Programming_. Pp. 389-392. New York: McGraw-Hill Book Co.

Downie, N.M., and Heath, R.W. 1965. _Basic Statistical Methods_. Pp. 135-136. New York: Harper & Row.

Eilon, Samuel, and Chowdhury, I.G. 1974. A note on steady-state results in queueing and job-shop scheduling. _Simulation_. Pp. 85-87.

Emshoff, James, and Sisson, Roger L. 1970. _Design and Use of Computer Simulation Models_. P. 192. New York: MacMillan Co.

Farmer, Vic. 1974a. Applications-oriented products dominate exhibits. _Computerworld_. February 27. P. 12.

Farmer, Vic. 1974b. Circuitry gains may have no effect. _Computerworld_. February 27. P. 25.

Farmer, Vic. 1974c. A look at 1985--Part IV--Moving head disk to remain best bet. _Computerworld_. March 27. P. 17.

Farmer, Vic. 1975. Imsai 108 intelligent disks ease construction of DBMS. _Computerworld_. August 13. P. 1.

Feingold, Robert S., and Chao, Yen W. 1974. Statistical instrumentation of ECSS models. Symposium on the Simulation of Computer Systems. Pp. 147-161. Gaithersburg, Maryland: National Bureau of Standards.

Fine, Gerald H., and McIsaac, Paul V. 1966. Simulation of a time-sharing system. Management Science. 12: B180-B194.

Fishman, George S. 1973. Concepts and Methods in Discrete Event Digital Simulation. P. 319. New York: John Wiley & Sons, Inc.

Flores, Ivan. 1970. Data Structure and Management. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Flynn, Michael J. 1972. Toward more efficient computer organizations. Proceedings of the Spring Joint Computer Conference. P. 1216.

Frank, H. 1969. Analysis and optimization of disk storage devices for time-sharing systems. JACM. 16: 602-620.

Frank, Ronald A. 1976a. Switched digital services provide savings to users. Computerworld. February 23. P. 27.

Frank, Ronald A. 1976b. Users told to expect few cost/ performance benefits. Computerworld. March 29. P. 25.

Fuchs, E., and Jackson, P.E. 1970. Estimates of distributions of random variables for certain computer communications traffic models. CACM. 13: 752-757.

Fuller, Samuel H. 1974. Minimal-total-processing time drum and disk scheduling disciplines. CACM. 17: 376-381.

Gecsei, J., and Lukes, J.A. 1974. A model for the evaluation of storage hierarchies. IBM Systems Journal. 13, 2: 163-178.

Gelenbe, Erol. 1973. The distribution of a program in primary and fast buffer storage. CACM. 16: 431-439.

Gepner, H.L. (ed.) 1975. Datapro 70. P. D09-200-003 ff. Deltran, New Jersey: Datapro Research Corp.

Ghosh, S.P. (n.d.) Some Results on Storage Space Requirement and Retrieval Time in Formatted File Organization. P. 23. New York: Rome Air Development Center.

Ghosh, S.P. 1972. File organization: The consecutive retrieval property. CACM. 15: 803.

Ghosh, S.P., and Lum, V.Y. 1975. Analysis of collisions when hashing by division. Information Systems. 1: 15-22.

Glans, Thomas B., et al. 1968. Management Systems. New York: Holt, Rinehart & Winston, Inc.

Goff, Norris S. 1974a. Standard benchmarks: prototype here? Computerworld. July 24. Pp. 1-2.

Goff, Norris S. 1974b. Generator solves test data problems. Computerworld. July 31. P. 9.

Gordon, William J., and Newell, Gordon F. 1967. Closed queueing systems with exponential servers. Operations Research. 15: 254-265.

Gotlieb, C.C., and MacEwen, G.H. 1973. Performance of movable-head disk storage devices. JACM. 20: 604-623.

Graef, Jed. 1970. The influence of cognitive states on time estimation and subjective time rate. Dissertation Abstracts International. 31 (2-B): 897.

Green, Paul E.; Robinson, P.J.; and Fitzroy, P.T. 1967. Experiments on the Value of Information in Simulated Marketing Environments. P. 34. Boston: Allyn and Bacon.

Greene, James. 1967. Operations Planning and Control. P. 75. Homewood, Illinois: Richard D. Irwin, Inc.

Greenbaum, H. 1968. A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System. M.S. Thesis, M.I.T.

Greenberger, Martin. 1966. The priority problem and computer time sharing. Management Science. 12: 890.

Grossman, David D., and Silverman, Harvey F. 1973. Placement of records on a secondary storage device to minimize access time. JACM. 20: 429-438.

Hare, Van Court, Jr. 1971. A special report on the SIGBDP forum 'The new data base task group report.' Data Base. 3, 3: 2-3.

Hatfield, D.J. 1972. Experiments on page size, program access patterns, and virtual memory performance. IBM Journal of Research and Development. 16: 58-66.

Hebert, John P. 1976. Users planning nets advised to 'Go for the minimum.' Computerworld. March 15. P. 29.

Hellerman, H., and Conroy, T.F. 1975. Computer System Performance. Pp. 92-100. New York: McGraw-Hill Book Co.

Hillegass, John R. 1966. Standardized benchmark problems measure computer performance. Computers and Automation. Pp. 16-19.

Hunt, Earl; Diehr, George; and Garnatz, David. 1971. Who are the users?--An analysis of computer use in a university computer center. Spring Joint Computer Conference. Pp. 231-238.

Hutchinson, G.K. 1965. A computer center simulation project. CACM. 8: 559-568.

Hutchinson, G.K. 1973. The use of micro-level simulation in the design of a computer supervisory system. Symposium on the Simulation of Computer Systems. Pp. 243-254. Gaithersburg, Maryland: National Bureau of Standards.

IBM. 1966. Introduction to IBM System/360 Direct Access Storage Devices and Organization Methods. P. 12. White Plains, New York.

Jackson, James R. 1957. Networks of waiting lines. Operations Research. 5: 518-521.

Jackson, James R. 1963. Jobshop-like queueing systems. Management Science. 10: 131-142.

Jackson, P.E., and Stubbs, C.D. 1969. A study of multi-access computer communication. Proceedings of the AFIPS Conference. 34: 491-504. Montvale, New Jersey: AFIPS Press.

Joslin, Edward O., and Aiken, John J. 1966. The validity of basing computer selection on benchmark results. Computers and Automation. Pp. 22-23.

Kaneko, T. 1974. Optimal task switching policy for a multi-level storage system. IBM Journal of Research and Development. Pp. 310-315.

Kaspar, Hans, and Miller, Donald S. 1974. Job and job-stream modeling in the TRW time-share system simulation. Symposium on the Simulation of Computer Systems. Pp. 95-121.

Katzan, Harry, Jr. 1971. Storage hierarchy systems. Proceedings of the Spring Joint Computer Conference. P. 334.

Kelly, Joseph F. 1970. Computerized Management Information Systems. P. 364. New York: MacMillan Co.

Kerlinger, Fred N. 1964. Foundations of Behavioral Research. Pp. 243-244. New York: Holt, Rinehart, and Winston, Inc.

Kimbleton, Stephen R., and Schneider, G.M. 1975. Computer communication networks: approaches, objectives, and performance considerations. Computing Surveys. 7: 138.

Kleijnen, Jack P.C. 1974. Monte Carlo simulation and its statistical design and analysis. Simuletter. 6: 25.

Knight, Kenneth E. 1966. Changes in computer performance. Datamation. P. 54.

Knuth, Donald E. 1975. The Art of Computer Programming. 2d ed. 1: 228-456. Reading, Mass.: Addison-Wesley Publ. Co.

Koenigsberg, Ernest. (n.d.) Cyclic queues. Operational Research Quarterly 9. 1: 22-35.

Kosy, Donald W. 1973. An interim empirical evaluation of ECSS for computer system simulation development. Symposium on the Simulation of Computer Systems. Pp. 79-90. Gaithersburg, Maryland: National Bureau of Standards.

Krinos, J.D. 1973. Interaction statistics from a database management system. Proceedings of the AFIPS Conference. Pp. 283-290. New Jersey: AFIPS Press.

Leavitt, Don. 1974. Outside supports aid data base management users. Computerworld. February 27. P. 16.

Lefkovitz, David. 1969. _File Structures for On-Line Systems_. Pp. 92-180. New York: Spartan Books.

Lefkovitz, David. 1974. _Data Management for On-Line Systems_. P. 122. Rochelle Park, New Jersey: Hayden Book Company, Inc.

Lewis, P.A.W., and Shedler, G.S. 1971. A cyclic-queue model of system overhead in multiprogrammed computer systems. _JACM_. 18: 201.

Liptay, J.S. 1968. Structural aspects of the system/360 model 85 II--the cache. _IBM Systems Journal_. 7: 15-21.

Lowe, Thomas C. 1968. The influence of data base characteristics and usage on direct access file organization. _JACM_. 15: 535-548.

Lucas, Henry C., Jr. 1973. _Computer Based Information Systems in Organizations_. P. 157. Chicago: Science Research Associates.

Lum, V.Y. 1973. General performance analysis of key-to-address transformation methods using an abstract file concept. _CACM_. 16: 603-612.

Lum, V.Y.; Yuen, P.S.T.; and Dodd, M. 1971. Key-to-address transform techniques: A fundamental performance study on large existing formatted files. _CACM_. 14: 228-239.

Lundell, E. Drake, Jr. 1973a. Programmer, installation manager aids featured. _Computerworld_. June 6. P.46.

Lundell, E. Drake, Jr. 1973b. Six FS design concepts stressed. _Computerworld_. October 10. P. 23.

Madnick, S.E. 1973. _Storage Hierarchy Systems_. P. 73ff. Cambridge, Mass.: M.I.T. Project MAC.

Madnick, S.E., and Donovan, J.J. 1974. _Operating Systems_. New York: McGraw-Hill Book Co.

Mahmoud, Samy, and Riordon, J.S. 1976. Optimal allocation of resources in distributed information networks. _ACM Transactions on Database Systems_. 1: 66-78.

Manocha, Trilok; Martin, William L.; and Stevens, Karl W. 1971. Performance evaluation of direct access storage devices with a fixed head per track. Proceedings of the Spring Computer Conference. Pp. 309-317.

Martin, E.W., Jr., and Perkins, William C. 1973. Computers and Information Systems: An Introduction. P. 360. Homewood, Illinois: Richard D. Irwin, Inc.

Martin, James. 1969. Telecommunications and the Computer. P. 350. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Martin, James. 1970. Teleprocessing Network Organization. P. 139. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Martin, James. 1976. Principles of Data-Base Management. P. 211. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Martin, Laurence D. 1969. A model for file structure determination for large on-line data files. International Federation of Information Processing Society. Occasional Publication No. 3, File Organization, Selected Papers from File 1968. Pp. 223-245. Amsterdam: Swets & Zeitlinger.

Martin, W.L. 1970. System analysis program: A simulation technique for computer system analysis. Proceedings of the Summer Simulation Conference. Pp. 98-103.

Maurer, W.D. 1968. An improved hash code for scatter storage. CACM. 11: 35.

Maurer, W.D., and Lewis, T.G. 1975. Hash table methods. Computing Surveys. 7: 5-19.

Maxwell, William L., and Severance, Dennis G. 1973. Comparison of alternatives for the representation of data items values in an information system. Data Base. 5: 121-136.

Meier, Robert C.; Newell, William R.; and Pazer, Harold L. Simulation in Business and Economics. P. 300. Englewood Cliffs, New Jersey: Prentice-Hall, Inc.

Mikelskas, Robert A. 1973. Simulation of computer software systems. Proceedings of the Summer Simulation Conference. Pp. 156-166.

Moreira, A.C.S.; Pinheiro, C.; D'Elia, L.F. 1974. Integrating data base management into operating systems. Proceedings of the AFIPS Conference. Pp. 57-62.

Morgan, Howard Lee. 1974. Optimal space allocation on disk storage devices. CACM. 17: 139-142.

Moshman, Jack. 1958. The application of sequential estimation to computer simulation and Monte Carlo procedures. JACM. 5: 343-352.

Naftaly, Stanley M.; Johnson, Bruce G.; and Cohen, Michael C. 1973. COBOL Support Packages. P. 81. New York: John Wiley and Sons.

Nahouraii, E. 1974. Direct-access device simulation. IBM Systems Journal. 13, 1: 19-31.

Nanamaker, J.F.; Swenson, D.E.; and Whinston, A.B. 1973. Specifications for the development of a generalized data base planning system. Proceedings of the AFIPS Conference. Pp. 259-270. New Jersey: AFIPS Press.

Newkirk, Ross, and Mullin, James K. 1974. Simulated demand in large-scale batch processing acquisitions: role and justification. Simuletter. 6: 52-59.

Newpeck, Fred F. 1973. Data Administration. Pp. 226-235. PhD. Dissertation. University of Massachusetts.

Nielson, Norman R. 1967. The simulation of time sharing systems. CACM. 10: 397-412.

Nievergelt, J. 1974. Binary search trees and file organization. Computing Surveys. 6: 195-207.

Nolan, R.L. 1974. Data base--an emerging organizational function. Proceedings of the AFIPS Conference. 43: 897-901.

Norton, Susanne R. 1970. A user reports on simulating computer systems. Proceedings of the Summer Simulation Conference. Pp. 104-112.

Oliver, P. 1974. An experiment in the use of synthetic programs for system benchmarking. Proceedings of AFIPS Conference. Pp. 431-438. New Jersey: AFIPS Press.

Oney, Walter C. 1975. Queueing analysis of the scan policy for moving-head disks. JACM. 22: 397-412.

Opler, A. 1967. Fourth generation software. Datamation. Pp. 22-24.

Organick, Elliott I. 1972. The Multics System. Cambridge, Massachusetts: M.I.T. Press.

Owen, Joel. 1967. A criterion for investing in information. Management Science. 14: B-715.

Pack, Charles D. 1973. The effects of multiplexing on a computer-communications system. CACM. 16: 161-168.

Prendergast, S. Lawrence. 1972. Selecting a data management system. Computer Decisions. Pp. 12-14.

Price, C.E. 1971. Table lookup techniques. Computing Surveys. 3: 61-64.

Radosevich, James D. 1976. Choice of correct JCL entries important to gain RPS benefit. Computerworld. March 15. P. 24.

Ramamoorthy, C.V., and Chandy, K.M. 1970. Optimization of memory hierarchies in multiprogrammed systems. JACM. 17: 426-445.

Resnick, M.H. 1974. Remote data collection case study--telephone order processing system (TOPS). Proceedings of the AFIPS Conference. Pp. 709-735.

Riesz, R.R., and Klemmer, E.T. 1963. Subjective evaluation of delay and echo suppressors in telephone communications. Bell System Technical Journal. 42: 2919-2941.

Salasin, John. 1973. Hierarchical storage in information retrieval. CACM. 16: 291-295.

Salton, Gerard. 1962. Manipulation of trees in information retrieval. CACM. 5: 103-114.

Salton, Gerard. 1972. Dynamic document processing. CACM. 15: 659.

Sasaki, Kyohei. 1969. Statistics for Modern Business Decision Making. P. 433. Belmont, California: Wadsworth Publ. Co.

Sasser, Earl, et al. 1970. The application of sequential sampling to simulation: an example inventory model. CACM. Pp. 287-296.

Scherr, A.L. 1965. An Analysis of Time-Shared Computer Systems. Cambridge, Massachusetts: M.I.T. Project MAC.

Scherr, A.L. 1966. Time-sharing measurement. Datamation. P. 26.

Schwartz, Barry. 1974. Waiting, exchange, and power: the distribution of time in social systems. American Journal of Sociology. 79: 841-870.

Schwetman, Herbert D., and DeLine, James R. 1969. An operational analysis of a remote console system. Proceedings of the Spring Joint Computer Conference. P. 260.

Senko, Michael E., et al. 1969. File Design Handbook. San Jose: IBM.

Senko, Michael E. 1975a. Information systems: records, relations, sets, entities, and things. Information Systems. 1: 3-13.

Senko, Michael E. 1975b. The DDL in the context of a multi-level structured description: DIAM II with FORAL. Data Base Description. Pp. 239-257. B.C.M. Douque and G.M. Nijssen, (eds.) Netherlands: North-Holland Publishing Co.

Severance, Dennis G. (n.d.) The evaluation of data structures in data base system design. Advanced Management Research International, Inc. C131-136.

Shedler, G.S. 1973. A queueing model of a multiprogrammed computer with a two-level storage system. CACM. 16: 3-10.

Shneiderman, Ben. 1973. Optimum data base reorganization points. CACM. 16: 362-365.

Shneiderman, Ben. 1974. Bibliography on data base structures. Data Base. 6: 3-9.

Siegel, Sidney. 1956. Nonparametric Statistics for the Behavioral Sciences. P. 116. New York: McGraw-Hill Book Co.

Siler, Kenneth F. 1976. A stochastic evaluation model for database organizations in data retrieval systems. CACM. 19: 84-95.

Smith, Charles St. John, Jr. 1976. Generalized communications software does exist. Computerworld. January 12. P. 23.

Snuggs, Mary E.; Popek, Gerald J.; and Peterson, Ronald J. 1974. Data base system objectives as design constraints. Data Base. 6: 11-20.

Sreenivasan, K., and Kleinman, A.J. 1974. On the construction of a representative synthetic workload. CACM. 17: 127.

Stimler, S. 1965. Real-Time Data Processing Systems. New York: McGraw-Hill Book Company.

Stone, Harold S. 1972. Introduction to Computer Organization and Data Structures. New York: McGraw-Hill Book Company.

Sussenguth, Edward H., Jr. 1963. Use of tree structures for processing files. CACM. 6: 272-279.

Tavitian, Aso. 1975. SMF data valid for billing but bad for measurement. Computerworld. December 17. P. 16.

Thananitayaudom, Tavorn. 1976. An IBM-developed nomographic technique for simplifying network design problems. Modern Data. 9: 29-34.

Trueman, Richard E. 1974. An Introduction to Quantitative Methods for Decision Making. Pp. 74-77. New York: Holt, Rinehart, and Winston, Inc.

Upton, Molly. 1976. Supercomputers have future, but micros seen factor. Computerworld. March 8. P. 40.

Voich, Dan; Mottice, H.J.; Shrode, W.A. 1975. Information Systems for Operations and Management. P. 57. Pelham Manor, New York: South-Western Publ. Co.

Ward, Patrick. 1974. DBMS users satisfied but transition can be rough. Computerworld. February 27. P. S/3.

Weingarten, Allen. 1966a. Analyzing a real-time system. Proceedings of the ACM. P. 179.

Weingarten, Allen. 1966b. The Eschenbach drum scheme. CACM. 9: 509-512.

Whitney, V.K.M. 1973. Fourth generation data management systems. Proceedings of the AFIPS Conference. 42: 239-244.

Wilhelm, Neil C. 1976. An anomaly in disk scheduling: a comparison of FCFS and SSTF seek scheduling using an empirical model for disk accesses. CACM. 19: 13-17.

Williams, John G. 1973. Asymmetric memory hierarchies. CACM. 16: 213-222.

Wilson, Thomas A., II. 1974. File management in management information systems. Decision Sciences. 5: 374-388.

Withington, Frederic G. 1969. The Real Computer: Its Influence, Uses, and Effects. P. 37. Reading, Mass.: Addison-Wesley Publ. Co.

Yue, P.C., and Wong, C.K. 1973. On the optimality of the probability ranking scheme in storage applications. JACM. 20: 624-633.

APPENDICES

APPENDIX 1

CPU DISCIPLINE 2 AND VARIATION REDUCTION OF SYSTEM RESPONSE TIME

The system response time has its effect on the firm in the form of lost potential sales. A customer, dissatisfied with the delay in receiving information, resolves not to use the information system again. This resolve may or may not be able to be changed at a later date. In a given customer population there is associated with each system response time a probability that the customer will no longer continue to use the Product Information System. This also represents lost potential sales. One expression of this probability function can be modeled as:

$$\text{(eq. 1)} \quad Y = \frac{a}{a + b^{-cx}}$$

where Y = the probability that a customer will no longer use the system,

x = the average system response time,

a, b, c = constants

This gives a typical sigmoid curve. For example, if we wish to construct an approximate fit between these different system response time levels and the associated probabilities of a lost customer,

System Response Time (secs.)	Probability of a Lost Customer
10	0.10
25	0.60
30	0.80
35	0.90

the following variables provide a good approximation in equation 1:

$$a = 0.015$$

$$b = 1.2$$

$$c = 1.0$$

This sigmoid curve can influence the behavior of discipline 2. Assuming a normal distribution of system response times, if the mean system response time is less than the inflection point on the curve, it is possible to experience simultaneously under discipline 2 a greater system response time than discipline 1, but a smaller average potential sales loss. All other things being equal, discipline 2 can have a greater average system response time than discipline 1 because of the additional overhead associated with discipline 2 in processing queries. It is possible for the average potential sales loss to be less in discipline 2 than discipline 1 due to the increasing slope of the curve below the inflection point. Figure 1 demonstrates this.

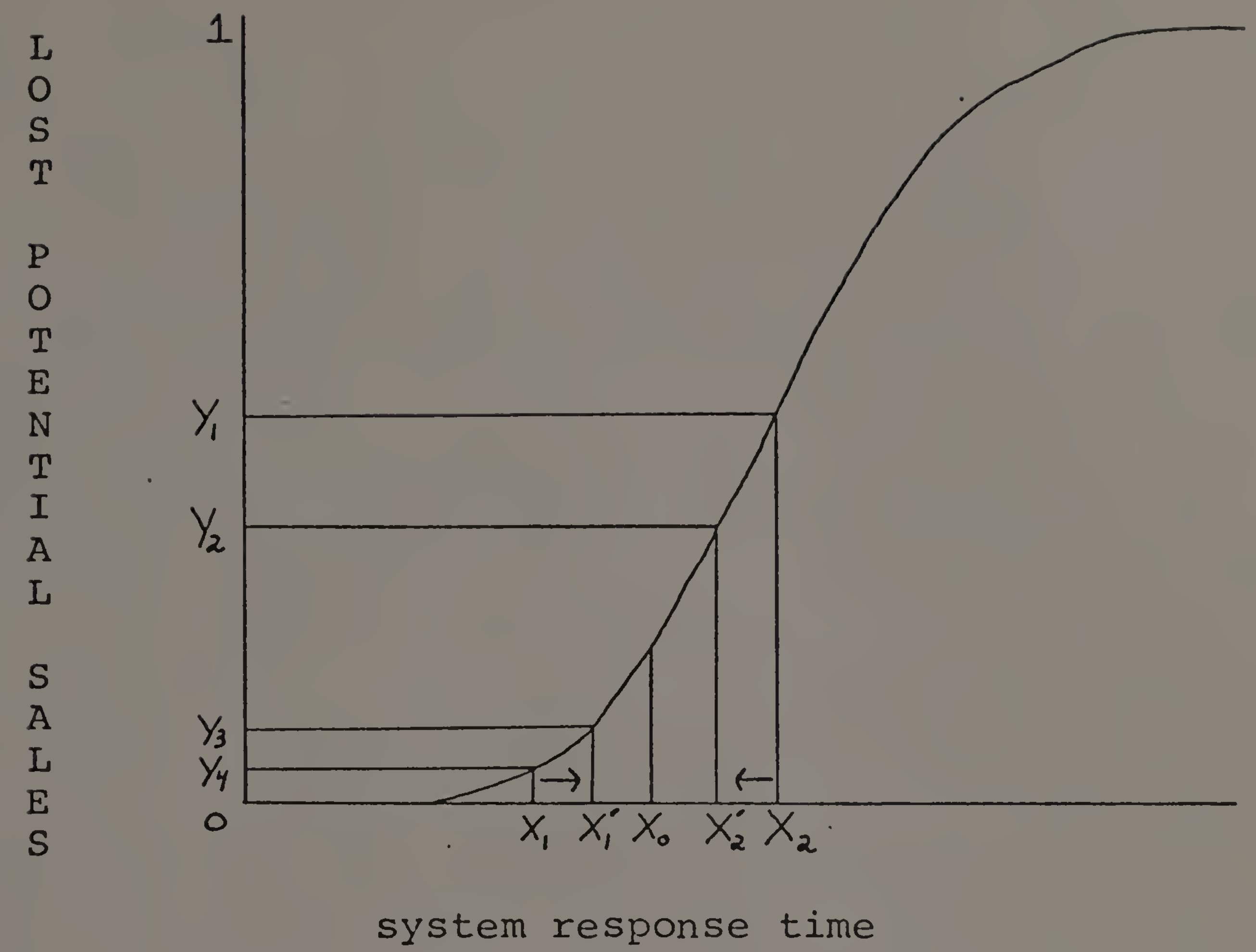


Figure 1: Lost Potential Sales Under Disciplines 1 & 2.

In Figure 1, X_0 is the mean system response time. X_1 and X_2 are the system response times associated with two queries under discipline 1. X_1' and X_2' are the response times under the same queries but a result of CPU scheduling by discipline 2. Since query 2 was detected by discipline 2 as being delayed in the system and query 1 ahead of schedule, discipline 2 can give query 2 a higher priority during processing. If we for a moment neglect discipline 2's additional overhead, this results in an increase in query 1's response time and a corresponding decrease in the response time for query 2. Judged by the performance measure of response time, we have an even tradeoff. The advantage in using discipline 2 would seem to be in a more uniform (smaller variance) system response time. However, when we measure the performance of the two according to lost potential sales, it becomes evident that here discipline 2 is providing a smaller lost potential sales and is in this example therefore superior under this performance measure. $Y_3 - Y_4 < Y_1 - Y_2$.

When we include a consideration of the additional overhead of discipline 2 in processing queries, the system response time is thereby increased. There is therefore a tradeoff on the lost potential sales axis between additional overhead and the possible reduction of variation, which is open to experimentation. The extent of any potential superiority of discipline 2 over discipline 1 would be dependent upon a number of factors: the values of a, b, and c, the average system response time, and the amount of variance reduction in the system response time. In addition, we will show in Chapter IV that discipline 2 has other effects on system response time beyond what is treated in this appendix.

When the system response time becomes greater than at the inflection point, the behavior of discipline 2 would be just the opposite. By reducing the variation discipline 2 would tend to aggravate an already poor system performance. Under a condition of a high average system response time the variation reduction of discipline 2 would increase the average potential sales loss beyond that of discipline 1.

We now show that the inflection point of this curve is located at a lost potential sales of 0.5.

Let $Y = f(X;a,b,c) = \dfrac{a}{a + b^{-Xc}} = 1 - \dfrac{1}{ab^{Xc} + 1}$

Change of variables: Let $M = b^{c}$

Then $Y = f(X;a,M) = 1 - \dfrac{1}{aM^{X} + 1}$

Now let $u = aM^{X}$, so $Y_u = 1 - \dfrac{1}{u + 1}$

$$\frac{dY}{du}(u) = \frac{1}{(u + 1)^2}$$

$$\text{so, } \frac{dY}{dX} = \frac{dY}{du} \cdot \frac{du}{dX} = \frac{1}{(u + 1)^2} \cdot u(\log M) = \frac{\log M}{u + 2 + \frac{1}{u}}$$

$$\frac{d}{du}(Y') = \frac{1 - u}{(u + 1)^3} \log M$$

$$Y'' = \frac{d}{dX}(Y') = \frac{d}{du}(Y') \cdot \frac{du}{dX} = \frac{u - u^2}{(u + 1)^3} (\log M)^2 = 0$$

if $u = 1$, $Y = 0.5$.

We conclude that under the scope of this discussion, discipline 2 can act as a two-edged sword, decreasing lost potential sales when the average system response time is less than at the inflection point and increasing the lost potential sales when the average system response time is higher. This discipline therefore tends to make a good Product Information System better and a poorly operating one worse.

APPENDIX 2

FILE SYSTEMS AND DATA BASE MANAGEMENT SYSTEMS

It is important to distinguish between a file system and a data base system and their respective management routines. Although a Product Information System could be implemented in either environment, a data base system would be more advantageous. Historically one could say a DBMS is an extension of the file system. Both systems manipulate data in the form of files. Yet, whereas a file management system is a set of programs written for specific file usage, a DBMS controls the structure, functioning, and maintenance of a data base as a repository of data irrespective of the users' programs which access the data base. A DBMS is not just a more complex file system. Both systems begin with a consideration of output and then input requirements, but from there the design strategies are antipodal.

In a file system one usually proceeds by then determining the processing specifications of the application programs. Lastly the files necessary to implement these specifications are structured along with appropriate accessing methods. In a file system the user's application programs are intimately tied to the files which they access. For example, an application program written in COBOL must have a one-to-one field correspondence in its data definitions of the files which are accessed.

In a Data Base System the design strategy centers around the data base, a set of integrated files. To use IBM Study Organization Plan (SOP) terminology (Glans, 1968), first the systems analyst delineates an activity or set of activities for a firm. Each activity is in turn broken down into a set of operations, each of which are then broken down into a set of processes. Once a decision has been made as to which activities, operations, and processes are to be automated, the Data Base Administrator or the Enterprise Administrator decides for all of the processes within the scope of the systems analysis study what elementary data items must be collected. He then attempts to organize these items into files as well as determine appropriate logical and physical relationships between these elementary data items. After the data base has been structured, then attention is paid to the application programs needed to process the data base.

By changing the focal point from the application programs to the data base we take the first step towards achieving data independence in which the structure of the data as seen by the application program need not be the same as the physical structure of the data base. Data independence and its associates, flexibility and data security, are provided at the expense of a complex DBMS to support such a data base.

As shall be brought out, data independence is not a luxury of a data base, but a necessity.

A file system in a business organization can be constructed as a segregated or an integrated file system. Figure 1 is an example of a segregated file system. Here each department of a firm is viewed as a closed node. Each department "owns" its own files, although physically all computer processing may be performed at a centralized facility. In the segregated file system each department begins with its own input, updates its own files, and generates its own output. In the accounting department of Figure 1, node 1, program A records sales order information in the accounts receivable (A/R) files, which files act as a subsidiary journal and ledger. Once goods have been shipped this entry can then be marked as a valid debit to the A/R ledger. Program B handles accounts payable (A/P). Vendor information and employee time card and salary information serve as the basis for the A/P subsidiary journal file and for the posting of credits to the A/P subsidiary ledger. Program C updates on a daily basis the general ledger with various debits and credits. Program D takes as input summary information from the A/R and A/P files and input on the period ending inventory levels to update the general ledger. The payroll program, E, inputs time cards and salary information and, besides generating employee payroll checks,

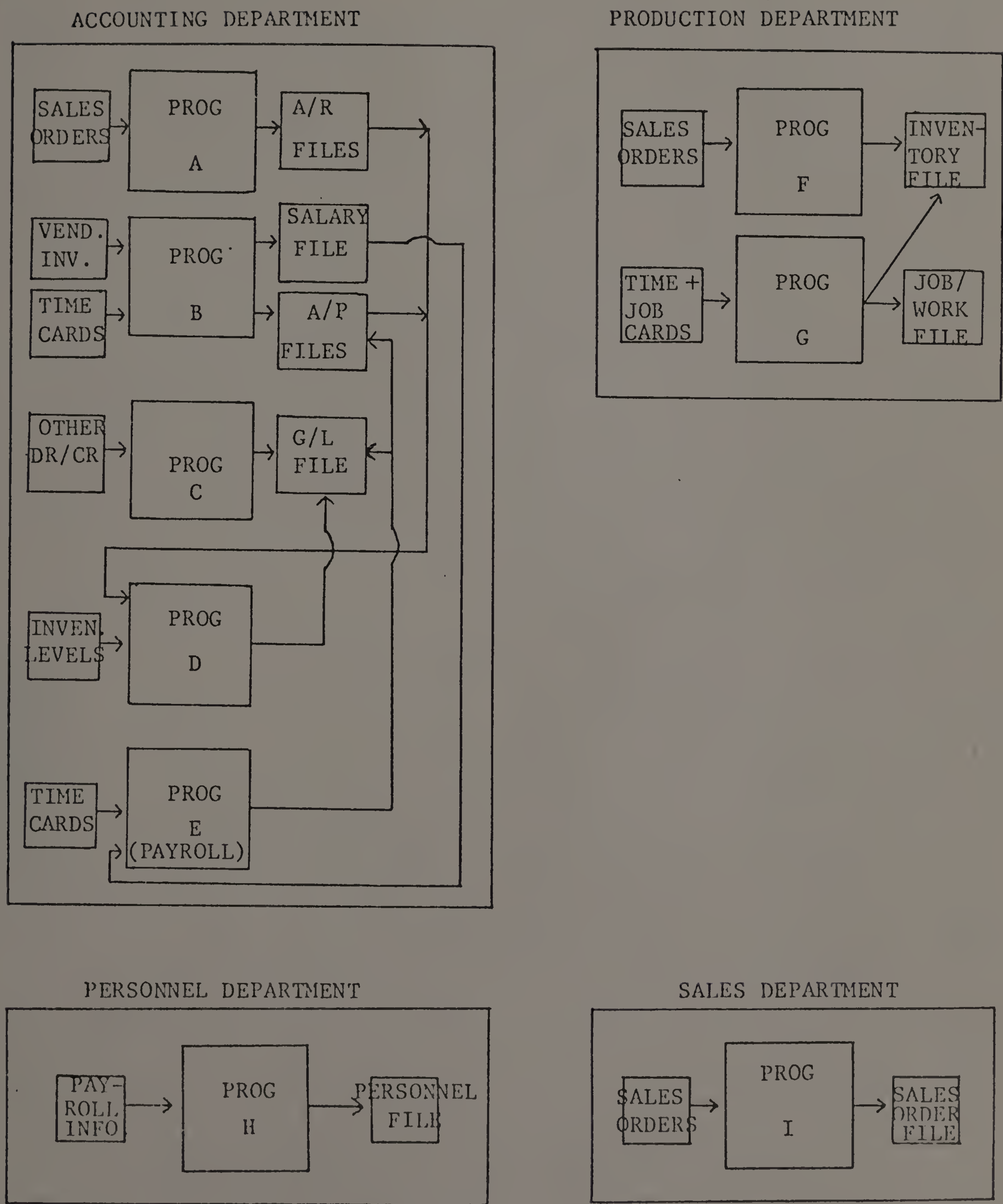


Figure 1: Segregated File System

debits the A/P file and creates a transaction in the general ledger (G/L) reflecting the taking of an expense.

In the production department, node 2, program F takes sales order information and updates the inventory file, reflecting that inventory items are to be allocated to various sales orders, although these items are considered still to exist as part of inventory. Program G accepts as input employee time cards, containing job and time completion information which can be used to update the job/work file and to adjust existing inventory levels.

In the personnel department, node 3, computer paper output from a payroll run is the original source document, which after having been punched is used by Program H to update the personnel file with payroll information. The sales department, node 4, processes sales orders to maintain a sales order file, which file can be used to produce various sales reports, including breakdowns by territory, by salesman as well as historical sales profiles for sales forecasts.

While this scheme is not expected to be complete in every detail, nevertheless, many companies today still have many of the characteristics of a segregated file system. Segregated file systems are inefficient for the following reasons. There is duplicate entry of the same input information within different departments or nodes. For example, in Figure 1 information from the sales order is extracted in nodes 1, 2, and 4. Time cards are inputted in runs for nodes 1 and 2. Much of this input is redundant. Secondly, a segregated file system suffers from needless intermediate output. For example, in Figure 1 at node 3, the personnel department, payroll information as output from the payroll run is used as input for updating the personnel file. If cooperation and file sharing were allowed between the accounting and personnel departments, the payroll run could directly update the personnel file. Redundant data items in the files themselves is the third source of the inefficiency of segregated file systems. Referring again to Figure 1, redundant data can exist between the job/work file and the personnel file.

An integrated file system solves some of these information system problems. There is usually a reduction in the number of necessary programs, input streams, and sometimes a reduction in the number of files. Figure 2 illustrates such a system. Sales orders are input to program A′, which updates the sales order file, enters information into the A/R files, and allocates inventory from the inventory file. Program B′ updates the A/P file, using data from vendor invoices, personnel time cards and salary information from the personnel file. The time cards are simultaneously used to update the job/work file and inventory levels in the inventory file.

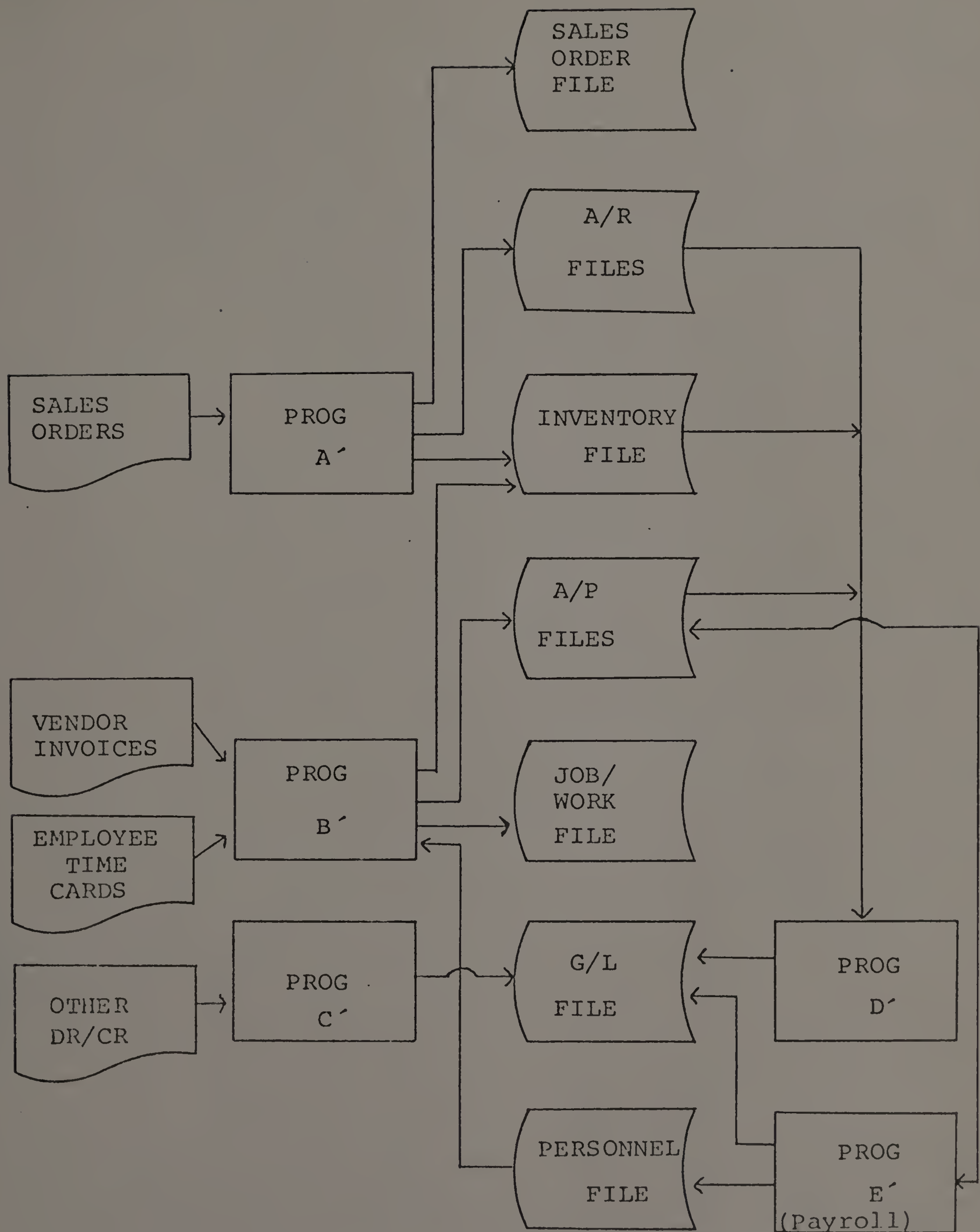


Figure 2: An Integrated File System.

Program D´fulfills the same functions as program D in Figure 1, except that program D´also updates the general ledger at the end of each period with summary information from the inventory file. The payroll program E´in Figure 2 is similar to program E in Figure 1, except that it also updates the personnel file directly.

In comparing Figures 1 and 2 we see that in Figure 2 there is a reduction of inputs and a reduction of programs. Yet, two problems still exist in an integrated file system: data item redundancy in the files and program dependency upon file structure. Also with the advent of the integrated file a new problem is born: maintaining corporate data security. A corporation must have control over the access and dissemination of its information on a need-to-know basis. Data base security has been defined as "protecting the database against deliberate destruction, modification, retrieval, or accidental exposure of information by an unauthorized user" (ANSI/X3/SPARC, 1975).

A program's dependency upon file structure can seriously impede the making of changes in both the logical and physical structure of an information system. In Figure 2 many of the programs listed are in actuality sets of programs. The payroll program may be 10 to 20 separate programs. If the physical structure of the general ledger were changed by the addition of a field to each record, then not only would the general ledger have to be reloaded, but changes would have to be made to each program within the program sets, C´ D´ and E´.

A Data Base system attempts to solve the problems mentioned above in file systems. In a Data Base system a department or node does not possess its own files, although it may be given authority to specify who has access rights to various files or data items within files. The first distinguishing characteristic of a data base is the application of the concept of a set to file structure. Figure 3 depicts the relationship between records in four files of a data base: the sales order file, the customer file, the job/work file, and the personnel file. Here each sales order is viewed as a set owner such that a variable number of jobs may belong to a sales order, representing the work needed to complete an order. Each sales order "owns" a customer member, the company from whom the sales order originated. The personnel file besides being a repository of historical facts on each employee can be structured so that each employee is an owner of a set of jobs on which he worked. While this figure represents the logical structure of the data base, the physical structure could be implemented with imbedded pointers as in Figure 4.

The sales order file in Figure 4 is shown as sequenced by order number. Each record has a pointer to the customer

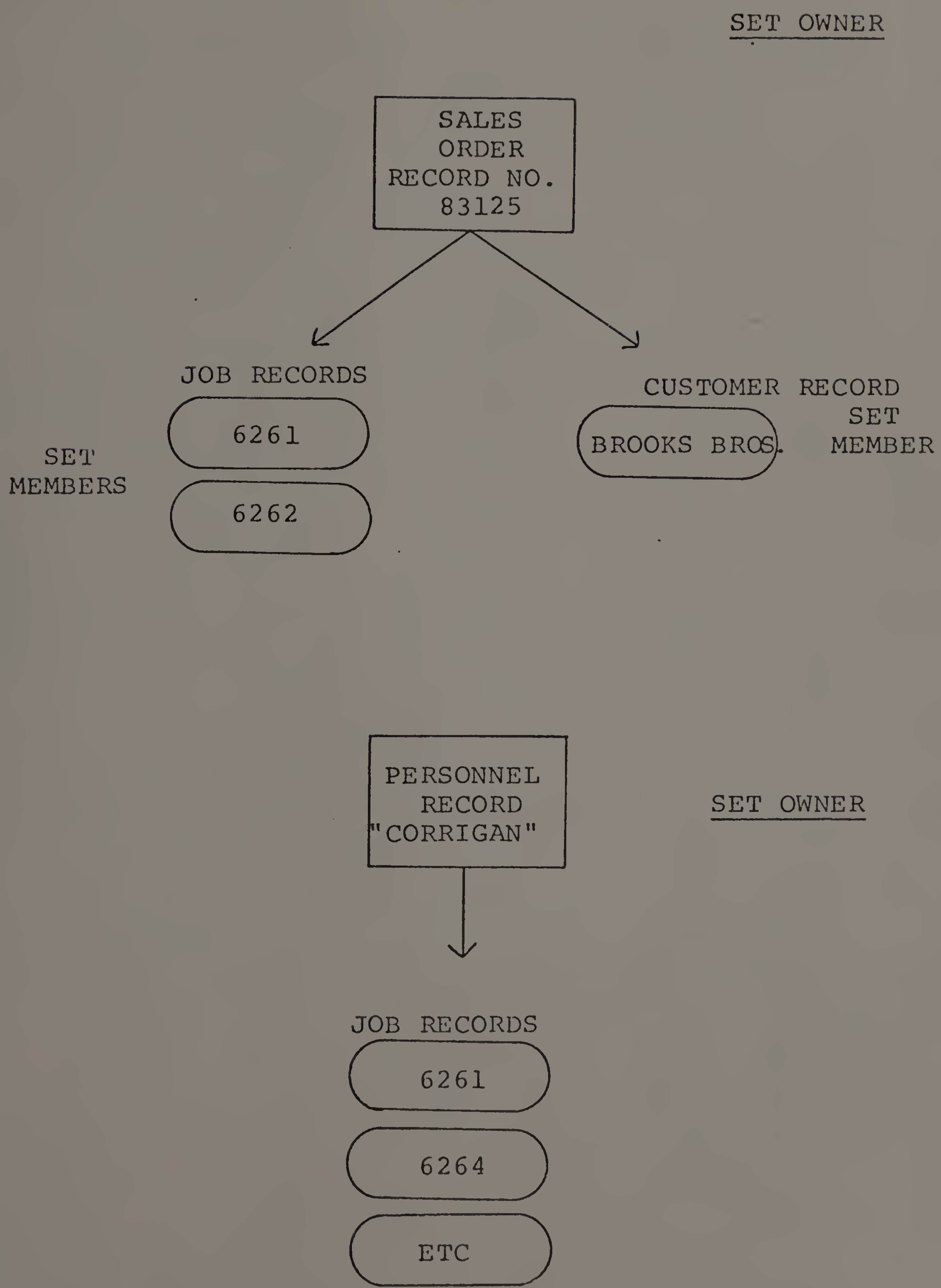


Figure 3: Owner/Member Relationships in a Data Base.

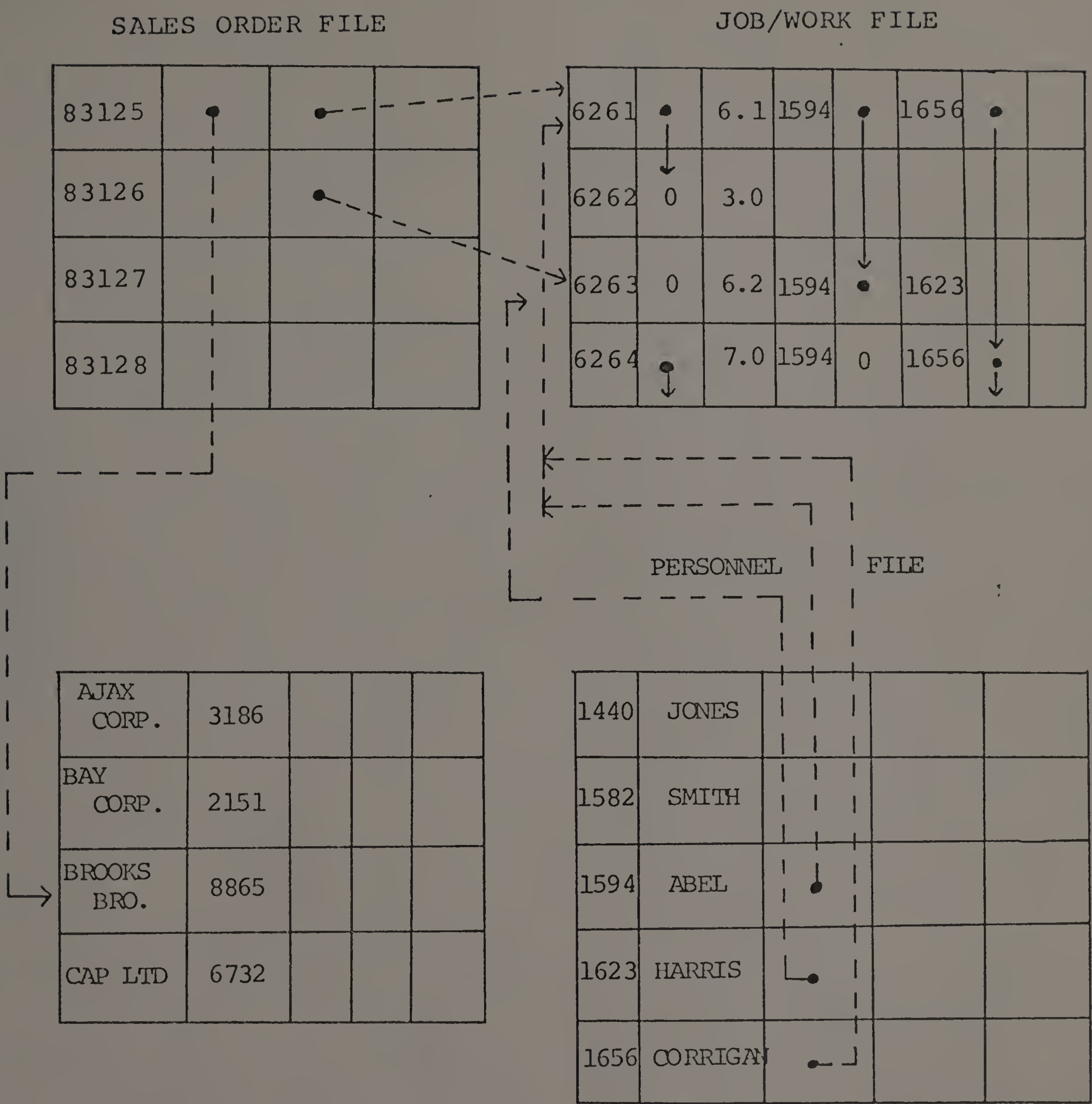


Figure 4: Relationship between Four Files in a Data Base.

file, which file could be ordered alphabetically. In addition, each sales order record as owner of a set of jobs has a job pointer, pointing to the first member of its set. The job/work file has a number of pointers. The left most pointer, the sales order pointer, links together the jobs belonging to a single sales order. In Figure 4 we assume more than one employee can work on a job. For each job entry there are a variable number of employee number-employee pointer pairs. In the job/work file the jobs each employee has worked on are linked together. The first member of each employee - job chain is referenced from the personnel file, since each job is considered a member of at least one employee record in the personnel file. In Figure 4 a pointer value of zero is by convention considered to end a chain. The relationship between the personnel file and the job/work file is an example of a network structure. In this network structure a given job/work record may have any number of immediate set owners in the personnel file.

In the data base example one can appreciate that a considerable amount of data item redundancy has been squeezed out, when compared with traditional file systems. Yet, in this example one could restructure these files in the data base so as to reduce redundancy even further. This, however, does not mean to imply that all redundancy is to be avoided in a data base system. The question of whether to allow a data item to be redundant can be resolved by considering the trade-off on the cost to store the item multiple times versus the cost of following pointers from one file to another to locate the desired data item.

In Figure 4 application programs that process the job/work file may need to reference the personnel file, at least to be able to update job pointers. This immediately gives rise to the security problem mentioned previously and forces as a consequence the construction of an interface between the user (application program) and the data base. This data base example demonstrates that such an interface is a necessity. Otherwise, it would be disastrous to give the complete file description of the personnel file to every application programmer who had to access the job/work file.

In a data base system a user program, which has authorization to update the job/work file, may be given a partial view of the personnel file. This user may be allowed to access the social security number, the employee name, and the job pointer from the personnel file as in Figure 5. In addition, this user may be restricted in his use of these data items. He may have a read only restriction on the first two data items. We now have at least two views of the data base, the DBA's and the user's view. The DBMS acts to reconcile this discrepancy.

Soc. Sec. #	Employee Name	Job Pointer

Figure 5: Subschema (External) View of Personnel File.

When this user program is run, the DBMS would trap all calls to the personnel file. After checking access rights to this file, the DBMS would map the user's view of the file to the actual structure (physical structure) of the file. The DBMS acts therefore to control access to the data base and allows an independence between the logical and physical structure of the data base as well as independence between these structures and the application programs or query routines which access the data base. From the user's point of view data independence insulates the user from the adverse effects of the evolution of the data environment. From the point of view of the firm, a DBMS provides data security. Other functions of a DBMS are presented in Chapter 2.

APPENDIX 3

SIMSCRIPT II.5 Model of a

Product Information System

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D

PREAMBLE
'' ****************************************************************************
'' *                                                                          *
'' *                PRODUCT INFORMATION SYSTEM                                *
'' *                PROGRAM 1                                                 *
'' *                SIMULATION AND SYSTEM STATE STATISTICS                    *
'' *                                                                          *
'' *                F. PAUL FUHS                                              *
'' *                                                                          *
'' ****************************************************************************
    NORMALLY MODE IS INTEGER
  DEFINE ELSE                                    TO MEAN REGARDLESS
  DEFINE ARRIVAL,SERVICE                         TO MEAN ARSEV
  DEFINE BALANCE.RAD                             TO MEAN BLR
  DEFINE BLOCKING.FACTOR                         TO MEAN BLF
  DEFINE BLOCKING,PERCENT                        TO MEAN BPC
  DEFINE BLOCK,QJ                                TO MEAN BQ
  DEFINE BUSY.2                                  TO MEAN BU2
  DEFINE CHANNEL,HOOK                            TO MEAN CHH
  DEFINE CPU.CHECK                               TO MEAN CUCK
  DEFINE CPU.QJ                                  TO MEAN CQ
  DEFINE CPU,TIME,TO,GO                          TO MEAN CTG
  DEFINE CRT.NUMBER                              TO MEAN CN
  DEFINE CRT.TRANS.READY                         TO MEAN CTR
  DEFINE GREET TIME                              TO MEAN GRT
  DEFINE LAST.PAGE                               TO MEAN LAPG
  DEFINE NO.OF.CYL                               TO MEAN NCL
  DEFINE NO.OF DEVICES                           TO MEAN ND
  DEFINE NO.OF.QUERIES                           TO MEAN NOQ
  DEFINE OPERATIONS                              TO MEAN OPN
  DEFINE OUTSIDE,TRANSIENT                       TO MEAN OTT
  DEFINE PAGE.ENTITY                             TO MEAN PAEN
  DEFINE QUERY                                   TO MEAN QRY
  DEFINE QUERY,ADDR                              TO MEAN QA
  DEFINE QUERY.NUM                               TO MEAN QNUM
  DEFINE RAD.BUSY                                TO MEAN RB
  DEFINE RAD.CHAN                                TO MEAN RC
  DEFINE RAD.CHANNEL                             TO MEAN RCHN
  DEFINE RAD.DEVICE                              TO MEAN RD
  DEFINE RAD,QJ                                  TO MEAN RQ
  DEFINE ROUT.ADD                                TO MEAN RA
  DEFINE ROUT.ADDR                               TO MEAN RA
  DEFINE SECOND.STAT                             TO MEAN SECD
  DEFINE SEEK.TIME                               TO MEAN SKD
  DEFINE SERV.START                              TO MEAN SES
  DEFINE THINK,TIME                              TO MEAN TT
  DEFINE TEL.QJ                                  TO MEAN TQ
  DEFINE TIME.ON                                 TO MEAN TON
  DEFINE TOTAL,CPU,TIME.NEEDED                   TO MEAN TCTN
  DEFINE TOTAL.PAGES.TO.BE.CREATED               TO MEAN TPTC
  DEFINE TRACE.MATRIX                            TO MEAN TMX
  DEFINE TYPING.TIME                             TO MEAN TYT
  DEFINE VIR.ADDR                                TO MEAN VA
```

LINE CACI SIMSCRIPT II.5 RELEASE 9D

```
    DEFINE VIR.ADDRESS                            TO MEAN VAD
    DEFINE VIRTUAL.ROUTINE                        TO MEAN VR
          THE SYSTEM OWNS AN OVFL.AREA
          PERMANENT ENTITIES
              EVERY CRT HAS AN AVAILABILITY, A QUERY.ADDR,
                                  A GREET.TIME, A THINK.TIME, A TYPING.TIME
                                  DEFINE GREET.TIME, THINK.TIME, TYPING.TIME
                                  AS REAL VARIABLES
              EVERY CPU HAS A SPEED AND A STATUS AND A ROUT.ADDR AND A TIME.ON
                    AND AN EV.ADDR AND OWNS A CPU.QU
                    DEFINE SPEED AND TIME ON AS REAL VARIABLES
              EVERY RAD HAS AN ARM, A RAD.BUSY, A LEVEL, A RAD.ROTATION.SPEED.
                     A NO.OF.CYL, A PAGES.PER.TRACK.
                    A CHANNEL.HOOK AND AN AVE.SEEK AND A RPS
                     AND A SEEK.TIME
                    AND OWNS A RAD.QU
                    '' ARM = -1 FOR A DRUM-LIKE DEVICE
                   DEFINE RAD.ROTATION.SPEED AND AVE.SEEK AS REAL VARIABLES
              DEFINE SEEK.TIME AS A REAL VARIABLE
              EVERY RAD.CHANNEL HAS A BUSY.2 AND OWNS A RAD.CHANNEL.QU
          TEMPORARY ENTITIES
              EVERY QUERY HAS A QUERY.NUM AND A GEN.TIME AND
                     A CONVERSATION,END AND
                     MAY BELONG TO A TEL.QU
                     DEFINE GEN.TIME AND CONVERSATION.END
                     AS REAL VARIABLES
              EVERY PAGE.ENTITY HAS A NUM.QUERY AND A PAGES AND MAY BELONG TO
                     A LAST.PAGE
              EVERY VIRTUAL,ROUTINE HAS A
                     VR.NO,
                   A SUPERVISOR,
                   A TYPE,
                   A CRT.NUMBER,
                   A SERVICING.ROUTINE.ADDR,
                   A CREATE,TM,
                   A TOTAL.CPU.TIME.NEEDED,
                   A CPU.TIME.TO.GO,
                   A TME,TILL.BLOCK,
                   A TIMES.BLOCKED,
                   A INTERRUPTS.PER.RUN,
                   A RAD,CHAN,
                   A RAD.DEVICE,
                   A WANT.CHANNEL,
                   A TOTAL.PAGES.TO.BE.CREATED,
                   A LOSS.RANK,
                   A BLOCK.RANK,
                   A SERV.START,
              AND MAY BELONG TO A ONE.QU, A LOSS.QU, A BLOCK.QU, A CPU.QU,
                     A RAD.QU,
                     A RAD.CHANNEL.QU
                     DEFINE CREATE.TM, TOTAL.CPU.TIME.NEEDED AND
                            CPU.TIME.TO.GO AND TME.TILL.BLOCK AND
                            SERV START AND
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D

                         LOSS.RANK, BLOCK.RANK AS REAL VARIABLES
          EVERY OVERFLOW HAS A VALUE AND MAY BELONG TO AN OVFL.AREA
                DEFINE VALUE AS A REAL VARIABLE
      EVENT NOTICES INCLUDE TELEPHONE.CALL
          EVERY ARRIVAL.AT.CRT HAS A CATHODE.TUBE
          EVERY CRT.TRANS.READY HAS A CRT.NUMB
          EVERY CRT.TRANS.FINISHED HAS A CATH.NO
          EVERY RUN.IT HAS A VIRT.ADDR
          EVERY OFF.CPU HAS A VIR.ADDR
          EVERY SEEK.END HAS A VIR.SEEK.ADDR
          EVERY LATENCY.END HAS A VIR.LAT.ADDR
          EVERY READ.END HAS A VIR.READ.ADDR
          EVERY ANSWER HAS A VIR.ADDRESS
          EVERY AN.END HAS A VADDR
      PRIORITY ORDER IS OFF.CPU, RUN.IT,
          CRT.TRANS.READY,
          CRT.TRANS.FINISHED,
          SEEK.END,
          LATENCY.END,
          READ.END,
          ANSWER,
          TELEPHONE.CALL,
          ARRIVAL.AT.CRT,
          AN.END
      DEFINE AA,AAA,AV, AV.RESP, BB, BBB, CCC, CUM, GREET.MEAN, HOLD, M3, M4,
      MEAN.RESPONSE,
      SD.RESP, SD.TALK, TALK.TIME, TEMP, THINK.MEAN, TOT.RESPONSE, TRANS.TIME,
      TYPING.MEAN, Z.CUT, Z.SCORE AS REAL VARIABLES
      DEFINE LINKER AS A REAL VARIABLE
      DEFINE DATA.FAULTS, BEG.FAULT, END.FAULT, INCR.FAULT AS REAL VARIABLES
      DEFINE NEXT.VR.NO AND LAST.ROUTINE AND QR
          AND NO.OF.QUERIES AND DISCIPLINE AND CHECK
          AND BEG.BLOCK AND BEG.LOAD AND BIG
          AND GOOD AND CHANGE.TO.GOOD AND OV.LOSS AND OV.BLOCK.LOSS
          AND OUTSIDE.LIMIT.QUERY
          AND RUN.INTERRUPTS AND RUN.BLOCKED
            AS INTEGER VARIABLES
      DEFINE SECOND.STAT AS AN INTEGER VARIABLE
      DEFINE ARR.TEST, BLOCKING.FACTOR, CEASE.RUN, COUNTER, COX.ON, CRT.CTR,
      DATA.ACCESS, DESIRE.PRINT, E, END.BLOCK, END.LOAD, INCR.BLOCK,
      INCR.LOAD, LIMIT.QUERY, LOAD.CHAN, LOWER.LIMIT, MAT.SIZE, PRINT.CP,
      PROG.ACCESS, SHARE, SHUNT, SHUT.OFF.TRACE, STOP.SHORT, WORK.LOAD
      AS INTEGER VARIABLES
      DEFINE BALANCE.RAD AS AN INTEGER VARIABLE
      DEFINE WANT.TAPE AS AN ALPHA VARIABLE
      DEFINE TEL.ARRIVAL AND HEAD AS ALPHA VARIABLES
      DEFINE ARRIVAL.SERVICE AS AN ALPHA VARIABLE
      DEFINE AC.PROB AS A 1-DIM REAL ARRAY
      DEFINE BLOCKING.PERCENT AS A 1-DIM REAL ARRAY
      DEFINE VAR.SERV AS A 1-DIM REAL ARRAY
      DEFINE MANY AS A 1-DIM REAL ARRAY
      DEFINE RATE AS A 1-DIM REAL ARRAY
      DEFINE C1 AND D2 AND FIR.RESPONSE AND SEC.RESPONSE AND RESPONSE
```

LINE CACI SIMSCRIPT II.5 RELEASE 8C

```
          AS A 1-DIM REAL ARRAY
     DEFINE TRANSIENT AS A 1-DIM INTEGER ARRAY
     DEFINE OUTSIDE.TRANSIENT AS A 1-DIM INTEGER ARRAY
     DEFINE EXPECTED AS A 2-DIM REAL ARRAY
     DEFINE ACCESS.TABLE AS A 2-DIM REAL ARRAY
     DEFINE DIST.ARRIVAL AS A 2-DIM REAL ARRAY
     DEFINE TRACE.MATRIX AS A 3-DIM REAL ARRAY
     DEFINE OPERATIONS AS A 3-DIM REAL ARRAY
     THE SYSTEM OWNS A LOSS.QU AND A BLOCK.QU AND A TEL.QU AND A ONE.QU
          AND A LAST.PAGE
     DEFINE MANN.WHITNEY AS A REAL FUNCTION
     DEFINE SEEK AS A REAL FUNCTION
     DEFINE FTEST AS A REAL FUNCTION
     DEFINE LOSS.FUNCTION AS A REAL FUNCTION
          DEFINE RAD.CHANNEL.QU AS A SET
          DEFINE RAD.QU AS A FIFO SET
          DEFINE TEL.QU AS A FIFO SET
          DEFINE ONE.QU AS A FIFO SET
          DEFINE CPU.QU AS A FIFO SET
     DEFINE LOSS.QU AS A SET RANKED BY HIGH LOSS.RANK
     DEFINE BLOCK.QU AS A SET RANKED BY LOW BLOCK.RANK
     DEFINE LAST.PAGE AS A SET
     DEFINE OVFL.AREA AS A SET RANKED BY LOW VALUE
     TALLY INT.MAX AS THE MAX, INT.AVE AS THE MEAN, INT.LOW AS THE MIN
          OF RUN.INTERRUPTS
     TALLY BLK,MAX AS THE MAX, BLK.AVE AS THE MEAN, BLK.LOW AS THE MIN
          OF RUN.BLOCKED
     TALLY SK.MAX AS THE MAX, SK.AVE AS THE MEAN, SK.LOW AS THE MIN OF
          SEEK.TIME
     ACCUMULATE RAD.MEAN AS THE MEAN OF RAD.BUSY
     ACCUMULATE CHAN.MEAN AS THE MEAN OF BUSY.2
     ACCUMULATE TEL.MAX AS THE MAX, TEL.AVE AS THE MEAN, TEL.LOW AS THE MIN
          OF N.TQ
     ACCUMULATE CPU.MAX AS THE MAX, CPU.AVE AS THE MEAN, CPU.LOW AS THE MIN
          OF N.CQ
     ACCUMULATE CPU.UTIL AS THE MEAN OF STATUS
     ACCUMULATE RAD.MAX AS THE MAX, RAD.AVE AS THE MEAN, RAD.LOW AS THE MIN
          OF N.RQ
     ACCUMULATE CHAN.MAX AS THE MAX, CHAN.AVE AS THE MEAN, CHAN.LOW AS
          THE MIN OF N.RAD.CHANNEL.QU
     ACCUMULATE ONE.MAX AS THE MAX, ONE.AVE AS THE MEAN, ONE.LOW AS THE MIN
          OF N.ONE.QU
     ACCUMULATE LOSS.MAX AS THE MAX, LOSS.AVE AS THE MEAN, LOSS LOW AS THE
          MIN OF N.LOSS.QU
     ACCUMULATE BL.MAX AS THE MAX, BL.AVE AS THE MEAN, BL.LOW AS THE MIN
          OF N.BQ
     ACCUMULATE PG.MAX AS THE MAX, PG.AVE AS THE MEAN, PG.LOW AS THE MIN
          OF N.LAPS
     DEFINE READ.PRINT AND TRACE AND SECOND.TRACE AS RELEASABLE ROUTINES
     DEFINE SPREAD AS A RELEASABLE ROUTINE
END
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D

MAIN
          DEFINE D AS A REAL VARIABLE
          CALL READ.PRINT RELEASE READ.PRINT
          RESERVE RESPONSE AS 100
          RESERVE EXPECTED AS LAST.ROUTINE BY 4
          LET MAT.SIZE = 40
          LET SHAPE = SHUT.OFF.TRACE +50
          RESERVE TRACE.MATRIX AS SHAPE BY MAT.SIZE BY 6
          RESERVE DIST.ARRIVAL AS 100 BY N.RAD *2
          RESERVE RATE AS N.RAD *2
          RESERVE VAR.SERV AS N.RAD
          RESERVE MANY AS N.RAD
          FOR BLOCKING.FACTOR = BEG.BLOCK TO END.BLOCK BY INCR.BLOCK
               FOR WORK.LOAD = BEG.LOAD TO END.LOAD BY INCR.LOAD
                     FOR DATA.FAULTS = BEG.FAULT TO END.FAULT BY INCR.FAULT
          DO
               SKIP 5 LINES PRINT 1 LINE WITH WORK.LOAD AND BLOCKING.FACTOR THUS
---------- NOW BEGINNING A SIMULATION WITH WORK.LOAD =      * AND BLOCKING     *
               SKIP 2 LINES
          LET E = 1
               LET DISCIPLINE = 1 CALL INITIALIZATION
          LET BETWEEN.V = 'TRACE'
START SIMULATION
          CALL STAT
               RESERVE FIR.RESPONSE AS NO.OF.QUERIES - TRANSIENT (1)
          FOR I=1 TO NO.OF.QUERIES - TRANSIENT (1)
               LET FIR.RESPONSE (I) = RESPONSE (I+ TRANSIENT (1))
          IF STOP.SHORT = 1
PRINT 1 LINE THUS
PREMATURELY ENDING SIMULATION
          STOP ELSE
          LET DISCIPLINE = 2 LET E=1
          CALL INITIALIZATION
          START SIMULATION
          CALL STAT
          RESERVE SEC.RESPONSE AS NO.OF.QUERIES - TRANSIENT (2)
          FOR I = 1 TO NO.OF.QUERIES - TRANSIENT (2)
               LET SEC.RESPONSE (I) = RESPONSE (I+ TRANSIENT (2))
          LET DISCIPLINE = 3  LET E = 1
          CALL INITIALIZATION
          START SIMULATION
          CALL STAT
          IF WANT.TAPE = "NO  " JUMP AHEAD ELSE
          CALL TAPE
          HERE
          IF SHUNT = 1 JUMP AHEAD ELSE
          CALL NORMAL
               HERE
          LET DISCIPLINE = 1
          RESERVE D1 AS DIM.F ( FIR.RESPONSE (*))
          FOR I=1 TO DIM.F (D1(*)) LET D1(I) = FIR.RESPONSE (I)
          CALL DISC.STAT
          FOR I=1 TO DIM.F(D1(*)) LET D1(I) = FIR.RESPONSE (I)
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                    06/28/76

           CALL AUTO.CORRELATION
           RELEASE D1
           LET DISCIPLINE = 2
           RESERVE D1 AS DIM.F ( SEC.RESPONSE (*))
           FOR I=1 TO DIM.F (D1(*)) LET D1(I) = SEC.RESPONSE (I)
           CALL DISC.STAT
           FOR I=1 TO DIM.F(D1(*))  LET C1(I) = SEC.RESPONSE (I)
           CALL AUTO.CORRELATION
           RELEASE C1
           LET DISCIPLINE = 3
           RESERVE C1 AS DIM.F ( RESPONSE (*)) - TRANSIENT (3)
           FOR I=1 TO DIM.F(D1(*)) LET D1(I) = RESPONSE (I+ TRANSIENT (3))
           CALL DISC.STAT
           FOR I=1 TO DIM.F(D1(*))  LET C1(I)= RESPONSE (I+ TRANSIENT (3))
           CALL AUTO.CORRELATION
           RELEASE C1
           RELEASE     FIR.RESPONSE, SEC.RESPONSE
           FOR KK= 1 TO 3 LET TRANSIENT (KK) = 0
           LOOP
           PRINT 1 LINE THUS
------- NORMAL END OF PROGRAM ---------
     END
```

LOCAL VARIABLES OF THIS ROUTINE

D	REAL	WORD	1	I	INTEGER	WORD	15
I.1	INTEGER	WORD	5	I.2	INTEGER	WORD	6
I.3	INTEGER	WORD	7	J.1	INTEGER	WORD	8
K.1	INTEGER	WORD	10	K.2	INTEGER	WORD	11
K.3	INTEGER	WORD	12	K.4	INTEGER	WORD	13
KK	INTEGER	WORD	16	L.10	INTEGER	WORD	14
L.8	INTEGER	WORD	2	N.1	INTEGER	WORD	9
R.1	DOUBLE	WORD	3				

```
ROUTINE READ.PRINT
          READ HEAD
          READ N.CRT CREATE EACH CRT
          FOR EACH CRT LET AVAILABILITY (CRT) = 1
          READ GREET.MEAN AND THINK.MEAN AND TYPING.MEAN
          READ HEAD
          READ N.CPU CREATE EACH CPU
          FOR EACH CPU READ SPEED (CPU)
          READ HEAD
          READ N.RAD CREATE EACH RAD
          FOR EACH RAD READ ARM ( RAD ), LEVEL (RAD), RAD.ROTATION.SPEED ( RAD ),
                            NO.OF.CYL ( RAD ) AND PAGES.PER.TRACK ( RAD ) AND
                            CHANNEL.HOOK ( RAD ) AND AVE.SEEK ( RAD ) AND
                            RPS ( RAD )
          READ HEAD AND N.RCHN CREATE EACH RAD.CHANNEL
          FOR EACH RAD COMPUTE BIG = MAX OF NO.OF.CYL
          ( RAD ) RESERVE ACCESS.TABLE AS N.RAD BY BIG + 1
          RESERVE AC.PROB AS N.RAD
          READ HEAD
          FOR L = 1 TO LEVEL ( N.RAD )
                DO READ TEMP
                   FOR EACH RAD WITH  LEVEL ( RAD ) = L COMPUTE X AS THE NUMBER
                      OF LEVEL ( RAD )
                   LET TEMP = TEMP / X
                     FOR EACH RAD WITH LEVEL (RAD)=L LET ACCESS.TABLE (RAD,1)=TEMP
               LOOP
          CALL SPREAD
          RELEASE SPREAD
          READ HEAD AND TRANS.TIME
          READ HEAD AND LAST.ROUTINE AND TALK.TIME AND SD.TALK
          READ HEAD AND SHUT,OFF,TRACE
          READ HEAD, AA, BB
          READ TEL.ARRIVAL
          READ HEAD, LOWER.LIMIT
          READ HEAD, LIMIT.QUERY
          IF LIMIT.QUERY > 0 LET OUTSIDE.LIMIT.QUERY = 1 ELSE
          RESERVE TRANSIENT AS 3
          RESERVE OUTSIDE.TRANSIENT AS 3
          READ HEAD AND OUTSIDE.TRANSIENT
          RESERVE BLOCKING.PERCENT AS 4
          READ HEAD AND BLOCKING.PERCENT
          RESERVE OPERATIONS AS 3 BY LAST.ROUTINE BY 2
          READ HEAD AND OPERATIONS
          '' SINCE SIMSCRIPT WILL NOT ALLOW A STANDARD DEVIATION OF ZERO
          '' WE DO THE FOLLOWING
          FOR I=1 TO 3 FOR J=1 TO LAST.ROUTINE
              DO
              IF OPERATIONS (I,J,2) = 0.0 LET OPERATIONS (I,J,2)=0.01 ELSE
              LOOP
          READ HEAD, BEG.FAULT, END.FAULT, INCR.FAULT, BEG.LOAD, END.LOAD,
              INCR.LOAD, BEG.BLOCK, END.BLOCK, INCR.BLOCK
          READ HEAD AND Z.OUT
          READ HEAD, AAA, BBB, CCC
```

```
            READ HEAD, LINKER
            READ HEAD, OV.LOSS, OV,BLOCK,LOSS
            ''THIS IS OVERHEAD FOR CALCULATIONS NEEDED TO DETERMINE RANK IN DBMS
                ''CPU QUEUE(S)
            READ HEAD AND BALANCE.RAD
            READ HEAD, LOAD.CHAN
            READ HEAD, STOP,SHORT
            READ HEAD, SHUNT
            READ HEAD, WANT.TAPE
            READ HEAD, ARRIVAL.SERVICE
         '' PRINT OUT INITIALIZATIONS '' START NEW PAGE
            PRINT 7 LINES WITH N.CRT AND GREET.MEAN AND THINK.MEAN AND TYPING.MEAN
                       AND TRANS.TIME AND TALK.TIME AND SD.TALK
               AND N.CPU AND SPEED (1) AND N.RAD THUS
THERE ARE   * CRTS WITH MEAN GREET TIME OF         *.*** SECS
                      MEAN THINK TIME OF        *.*** SECS
                      MEAN TYPING TIME OF       *.*** SECS
CRT TRANSMISSION TIME IS                       *.****** SECS
CRT REPRESENTATIVE ANSWER BACK TIME HAS MEAN OF   *.*** SECS    S.D.=   *.***SEC
THERE ARE   * CPUS WITH SPEED OF                      *.******** SECS
THERE ARE   * RADS
            SKIP 1 LINE
            FOR EACH RAD PRINT 4 LINES WITH RAD, ARM (RAD) AND LEVEL (RAD)
            AND RAD.ROTATION.SPEED ( RAD ) AND NO.OF.CYL ( RAD ) AND PAGES.PER.TRACK
                ( RAD ) AND CHANNEL.HOOK ( RAD ) AND AVE.SEEK ( RAD ) AND RPS (RAD)
                THUS
RAD NO  * HAS ARM POSITIONED AT CYLINDER    * ON LEVEL * WITH ROTATION SPEED
       *.******** SECS. THERE ARE   * CYLINDERS AND  * PAGES PER TRACK
        THE DEVICE IS CONNECTED TO CHANNEL * AND ITS AVE SEEK IS *.**** SECS
        WITH ROTATIONAL POSITIONING SENSING CODE OF *
            SKIP 2 LINES
            PRINT 3 LINES WITH LAST.ROUTINE AND LOWER.LIMIT AND LIMIT.QUERY THUS
THE LAST DBMS ROUTINE IS NUMBER            *
THE SIMULATION WILL CEASE AFTER      * QUERIES MINIMUM
THE SIMULATION WILL CEASE AFTER      * QUERIES MAXIMUM
            PRINT 1 LINE WITH SHUT.OFF.TRACE THUS
THE TRACE ROUTINE WILL SHUT OFF AFTER         * QUERIES
            PRINT 1 LINE WITH AA AND BB THUS
  SEEK TIME COEFFICIENTS ARE:   A=      *.*******  B=      *.********
            IF TEL.ARRIVAL = "FULL" PRINT 1 LINE THUS
THE SYSTEM IS AT PEAK LOAD --  ALL CRTS ARE OCCUPIED
      ELSE
            IF TEL.ARRIVAL = "SLOW" PRINT 1 LINE THUS
THE SYSTEM HAS A SLOW STEADY TELEPHONE START UP RATE
            ELSE
            PRINT 1 DOUBLE LINE WITH OUTSIDE.TRANSIENT (1), OUTSIDE.TRANSIENT (2)
                AND OUTSIDE.TRANSIENT (3) THUS
THE NUMBER OF QUERIES FOR THE TRANSIENT STATE IS      * FOR FIFO AND      * FOR LOSS AND         * FOR LOSS+BLOCK
            PRINT 1 LINE WITH BLOCKING.PERCENT (1)* 100  AND
                              BLOCKING.PERCENT (2)* 100  AND
                              BLOCKING.PERCENT (3)* 100  AND
                              BLOCKING.PERCENT (4)* 100 THUS
BLOCKING PERCENTAGES FOR DBMS ROUTINES ARE   *.*%   *.*%   *.*%   *.*%
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                              06/28/76

   87          PRINT 3 LINES WITH BEG.BLOCK, END.BLOCK, INCR.BLOCK, BEG.LOAD, END.LOAD,
   88              INCR.LOAD, BEG.FAULT, END.FAULT, INCR.FAULT
   89              THUS
BEGINNING BLOCKING FACTOR=    * ENDING BLOCKING FACTOR=  *  INCREMENT =    *
BEGINNING WORK LOAD NO.=      * ENDING WORK LOAD NO.=    *  INCREMENT =    *
BEGINNING PAGE FAULTS=     *.* ENDING PAGE FAULTS=     *.*  INCREMENT =  * *
   90          PRINT 1 LINE THUS
               AVERAGE AND STANDARD DEVIATION OF OPERATIONS PER ROUTINE
   91          FOR W = 1 TO 3
   92          FOR I = 1 TO LAST.ROUTINE PRINT 1 LINE WITH W, I, OPERATIONS (W,I,1),
   93              OPERATIONS (W, I, 2) THUS
FOR WORK LOAD   * AND ROUTINE  * THE AVE NO OF OPERATIONS=        *. S.D.=   *.**
   94          PRINT 1 LINE WITH AAA, BBB, CCC THUS
FOR LOSS FUNCTION:     A=      *.***     B=       *.***      C=       *.***
   95          PRINT 1 LINE WITH LINKER THUS
INSTRUCTION CONSTANT FOR CPU SCHEDULING ALGORITHM CHAINING =    *
   96          PRINT 1 DOUBLE LINE WITH OV.LOSS AND OV.BLOCK.LOSS THUS
OVERHEAD FOR CALCULATING THE LOSS FUNCTION =       * OPERATIONS  LOSS+BLOCK OVERHEAD =      *
   97          PRINT 1 LINE WITH BALANCE.RAD THUS
BALANCE RAD CODE =  *
   98          PRINT 1 LINE WITH LOAD.CHAN THUS
CODE ON LOAD.CHAN =  *
   99          PRINT 1 LINE WITH STOP.SHORT THUS
STOP.SHORT CODE =  *
  100          PRINT 1 LINE WITH SHUNT THUS
SHUNT CODE =  *
  101          PRINT 1 LINE WITH WANT.TAPE THUS
REQUEST FOR OUTPUT ONTO TAPE IS ****
  102          PRINT 1 LINE WITH ARRIVAL.SERVICE THUS
REQUEST FOR ARRIVAL AND SERVICE DISTRIBUTION IS ****
  103   RETURN END


               LOCAL VARIABLES OF THIS ROUTINE

  I            INTEGER   WORD 36      I.1          INTEGER   WORD  3
  I.2          INTEGER   WORD  4      I.3          INTEGER   WORD  5
  J            INTEGER   WORD 37      J.1          INTEGER   WORD  6
  K.1          INTEGER   WORD  8      K.2          INTEGER   WORD  9
  K.3          INTEGER   WORD 10      K.4          INTEGER   WORD 11
  L            INTEGER   WORD 19      L.13         DOUBLE    WORD 13
  L.14         DOUBLE    WORD 15      L.15         DOUBLE    WORD 17
  L.22         DOUBLE    WORD 21      L.23         DOUBLE    WORD 23
  L.27         INTEGER   WORD 26      L.28         INTEGER   WORD 27
  L.32         INTEGER   WORD 28      L.33         INTEGER   WORD 29
  L.37         INTEGER   WORD 30      L.38         INTEGER   WORD 31
  L.39         INTEGER   WORD 32      L.40         INTEGER   WORD 33
  L.41         INTEGER   WORD 34      L.42         INTEGER   WORD 35
  L.55         INTEGER   WORD 38      L.60         INTEGER   WORD 39
  N.1          INTEGER   WORD  7      R.1          DOUBLE    WORD  1
  W            INTEGER   WORD 40      X            INTEGER   WORD 25
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                          06/28/76

ROUTINE SPREAD
        NORMALLY MODE IS REAL
        DEFINE I AND J AND JJJ AS INTEGER VARIABLES
        DEFINE BIG AS AN INTEGER VARIABLE
        FOR EACH RAD COMPUTE BIG = MAX OF NO.OF.CYL (RAD)
        ''SPREAD EACH DEVICES ACCESS PROBABILITY ACROSS ITS CYLINDERS:BINOMIALLY
        FOR EACH RAD UNLESS NO.OF.CYL ( RAD ) = 0 DO
        LET L=6.0 / NO.OF.CYL ( RAD )
                '' THIS ASSUMES FURTHEST CYLINDER IS 3 S.D. FROM THE CENTER
                '' OF DEVICE
        '' SEE COSTIS, STATISTICS FOR BUSINESS, P. 289
        LET L2=(6.0 / NO.OF.CYL ( RAD )) / 2.0
        LET YY=1.0 / SQRT.F (2.0 * PI.C)
        LET POINT = -3.0
                FOR I = 1 TO BIG UNLESS I > NO.OF.CYL ( RAD )
            DO LET CO=0.5*(POINT - L2) **2
                LET Y1=YY*(1.0 / EXP.F (CO))
                LET CO=0.5*(POINT + L2)**2
                LET Y2= YY * (1.0 / EXP.F (CO))
                LET POINT = POINT + L
            LET ACCESS.TABLE ( RAD,I+1)=0.5*(Y1+Y2)*L*ACCESS.TABLE (RAD,1)
                    '' TRAPEZOIDAL RULE
        LOOP LOOP
        FOR EACH RAD UNLESS NO.OF.CYL ( RAD )=0
            DO FOR I = 2 TO BIG + 1
                    DO COMPUTE SUM.UP AS THE SUM OF ACCESS.TABLE ( RAD,I)
                    LOOP PRINT 1 LINE WITH RAD, ACCESS.TABLE ( RAD,1) AND SUM.UP
                            THUS
 PROBABILITY AMOUNT FOR RAD    * IS *.******  *.******
            LOOP
        FOR I=1 TO N,RAD LET AC.PROB (I) = ACCESS.TABLE (I,1)
        ''FORM A CUMULATIVE DISTRIBUTION
        LET TEMP = 0.0 LET CUM = 0.0
        FOR EACH RAD DO ADD ACCESS.TABLE (RAD,1) TO CUM
            FOR I=2 TO BIG + 1 WITH ACCESS.TABLE ( RAD,I) > 0.0 FIND JJJ=
            THE FIRST I
            IF NONE FOR I=2 TO BIG + 1 LET ACCESS.TABLE ( RAD,I)= CUM
                    ADD ACCESS.TABLE ( RAD,1) TO TEMP
                    JUMP AHEAD
                          ELSE FOR I=2 TO BIG + 1 UNLESS I > NO.OF.CYL (RAD) + 1
                    DO ADD TEMP TO ACCESS,TABLE ( RAD,I)
                          LET TEMP = ACCESS.TABLE (RAD,I)
            LOOP
                    LET ACCESS.TABLE (RAD, NO.OF.CYL (RAD) +1) = CUM
HERE
            LOOP
        LET TEMP = 0.0
        FOR EACH RAD DO ADD TEMP TO ACCESS.TABLE ( RAD,1)
            LET TEMP = ACCESS.TABLE ( RAD,1)
            LOOP
        FOR I = 1 TO N.RAD WITH NO.OF.CYL (I) NE 0
            DO LET SINGLE = ACCESS.TABLE (I, NO.OF.CYL (I)) -
                    ACCESS.TABLE (I, NO.OF.CYL (I) -1)
```

```
                    LET SLACK = ACCESS.TABLE (I, NO.OF.CYL (I) +1) -
                          ACCESS.TABLE (I, NO.OF.CYL (I)) - SINGLE
                    FOR J= NO.OF.CYL (I)/2 +1 TO NO.OF.CYL (I)
                          LET ACCESS.TABLE (I,J) = ACCESS.TABLE (I,J) + SLACK
            LOOP
        LET ACCESS.TABLE ( N.RAD, 1) = 1.0
        LET ACCESS.TABLE (N.RAD, NO.OF.CYL (N.RAD)+1) = 1.0
RETURN END
```

LOCAL VARIABLES OF THIS ROUTINE

BIG	INTEGER	WORD 4	I	INTEGER	WORD 1
I.1	INTEGER	WORD 29	I.2	INTEGER	WORD 30
I.3	INTEGER	WORD 31	J	INTEGER	WORD 2
J.1	INTEGER	WORD 32	JJJ	INTEGER	WORD 3
K.1	INTEGER	WORD 34	K.2	INTEGER	WORD 35
K.3	INTEGER	WORD 36	K.4	INTEGER	WORD 37
L	REAL	WORD 11	L.19	DOUBLE	WORD 19
L.20	DOUBLE	WORD 21	L.21	DOUBLE	WORD 23
L.4	DOUBLE	WORD 5	L.5	DOUBLE	WORD 7
L.6	DOUBLE	WORD 9	L2	REAL	WORD 12
N.1	INTEGER	WORD 33	POINT	REAL	WORD 14
SO	REAL	WORD 15	R.1	DOUBLE	WORD 27
SINGLE	REAL	WORD 38	SLACK	REAL	WORD 39
SUM.UP	REAL	WORD 25	YY	REAL	WORD 13
Y1	REAL	WORD 16	Y2	REAL	WORD 17

```
ROUTINE INITIALIZATION
        LET SEED.V(1)= 2116429302
        RELEASE RESPONSE RESERVE RESPONSE AS 100
        RELEASE DIST.ARRIVAL RESERVE DIST.ARRIVAL AS 100 BY N.RAD *2
        LET CR=0 LET NO.OF.QUERIES =0
        LET NEXT.VR.NO = 0
        LET SECOND.STAT = 0
        FOR I=1 TO N.RAD LET MANY (I) = 0
        IF DISCIPLINE = 1
        FOR I=1 TO LAST.ROUTINE FOR J= 1 TO 4 LET EXPECTED (I,J)= 0.0
            JUMP AHEAD ELSE
        IF DISCIPLINE NE 1 FOR I=1 TO LAST.ROUTINE UNLESS EXPECTED (I,3)=0.0
            DO LET EXPECTED (I,1) = EXPECTED (I,1) / EXPECTED (I,3)
               LET EXPECTED (I,3) = 1.0
               LET EXPECTED (I,2) = EXPECTED (I,2)/ EXPECTED (I,4)
               LET EXPECTED (I,4)=1.0
            LOOP
        ELSE HERE
        LET TOT.RESPONSE = 0.0
        LET MEAN.RESPONSE = 0.0
        LET BETWEEN.V =0
        LET TIME.V =0.0
        LET GOOD = 0
        LET ARR.TEST = 0
        LET PROG.ACCESS = 0  LET DATA.ACCESS = 0
        FOR EACH CRT LET AVAILABILITY (CRT) =1
        FOR EACH CPU DO LET STATUS (CPU) =0
            LET ROUT.ADDR (CPU) =0 LET EV.ADDR (CPU) =0
            LOOP
        'START' IF CPU.QU (1) IS EMPTY JUMP AHEAD ELSE
            REMOVE FIRST VR FROM CPU.QU (1) DESTROY VR GO TO START
        HERE
        FOR EACH RAD LET RAD.BUSY (RAD) =0
        FOR EACH RAD UNLESS ARM ( RAD ) = -1 LET ARM ( RAD )= NO.OF.CYL (RAD)/2
        FOR EACH RAD DO
        'H' IF RAD.QU (RAD) IS EMPTY CYCLE ELSE
            REMOVE FIRST VR FROM RAD.QU (RAD) DESTROY VR GO TO H
            LOOP
        FOR EACH RAD.CHANNEL LET BUSY.2 ( RAD.CHANNEL ) =0
        FOR EACH RAD.CHANNEL DO
        'J' IF RAD.CHANNEL.QU ( RAD.CHANNEL ) IS EMPTY CYCLE ELSE
            REMOVE FIRST VR FROM RAD.CHANNEL.QU ( RAD.CHANNEL )
            DESTROY VR GO TO J
            LOOP
        'K' IF TEL.QU IS EMPTY JUMP AHEAD ELSE
            REMOVE FIRST VR FROM TEL.QU DESTROY VR GO TO K
        HERE
        'L' IF ONE.QU IS EMPTY JUMP AHEAD ELSE
            REMOVE FIRST VR FROM ONE.QU DESTROY VR GO TO L
        HERE
        'M' IF LOSS.QU IS EMPTY JUMP AHEAD ELSE
            REMOVE FIRST VR FROM LOSS.QU IF BLOCK.QU IS EMPTY DESTROY VR GO TO M
                    ELSE REMOVE THIS VR FROM BLOCK.QU
```

```
                    DESTROY VR GO TO M
          HERE
          'N' IF BLOCK.CU IS EMPTY JUMP AHEAD ELSE
              REMOVE FIRST VR FROM BLOCK.CU DESTROY VR GO TO N
              HERE
          'P' IF LAST.PAGE IS EMPTY JUMP AHEAD ELSE
          REMOVE FIRST PAGE.ENTITY FROM LAST.PAGE DESTROY PAGE.ENTITY GO TO P HERE
          IF PRINT.CP = 1 LET SHUT.OFF.TRACE = 0 RELEASE TRACE.MATRIX ELSE
          RESET TOTALS OF RUN.INTERRUPTS AND RUN.BLOCKED AND N.TQ AND
              N.ONE.QU AND N.LOSS.QU AND N.BQ AND N.LAPG
          FOR EACH CPU RESET TOTALS OF N.CU (CPU)
          FOR EACH CPU RESET TOTALS OF STATUS (CPU)
          FOR EACH RAD RESET TOTALS OF N.RQ (RAD)
          FOR EACH RAD.CHANNEL RESET TOTALS OF N.RAD.CHANNEL.CU ( RAD.CHANNEL )
          FOR EACH RAD RESET TOTALS OF SEEK.TIME (RAD)
          FOR EACH RAD RESET TOTALS OF RAD.BUSY (RAD)
          FOR EACH RAD.CHANNEL RESET TOTALS OF BUSY.2 ( RAD.CHANNEL )
          START NEW PAGE
          PRINT 1 LINE WITH DISCIPLINE THUS
   ------ NEXT SIMULATION WITH DISCIPLINE  *    ------
          IF TEL.ARRIVAL = "FULL"
              FOR EACH CRT DO SCHEDULE A TELEPHONE.CALL NOW
                     SCHEDULE AN ARRIVAL.AT.CRT ( CRT ) NOW
                     LOOP ELSE
          IF TEL.ARRIVAL = "SLOW"
          LET CRT.CTR = 0
              FOR I=1 TO 4 DO ADD 1 TO CRT.CTR SCHEDULE A TELEPHONE.CALL NOW
                     SCHEDULE AN ARRIVAL.AT.CRT ( CRT.CTR) NOW
              LOOP ELSE
          LIST N.CO, N.RO, N.RAD.CHANNEL.QU, N.TQ, N.ONE.QU, N.LOSS.QU, N.BQ,
              N.LAPG
          RETURN END


            LOCAL VARIABLES OF THIS ROUTINE

I           INTEGER  WORD  1        I.1         INTEGER  WORD  5
I.2         INTEGER  WORD  6        I.3         INTEGER  WORD  7
J           INTEGER  WORD  2        J.1         INTEGER  WORD  8
K.1         INTEGER  WORD 10        K.2         INTEGER  WORD 11
K.3         INTEGER  WORD 12        K.4         INTEGER  WORD 13
N.1         INTEGER  WORD  9        R.1         DOUBLE   WORD  3
```

```
ROUTINE STAT
        NORMALLY MODE IS REAL
        START NEW PAGE
        IF SECOND.STAT = 0 PRINT 1 LINE THUS
 ------- STATISTICS UP TO LOWER LIMIT --------
        ELSE
        IF SECOND.STAT = 1 PRINT 1 LINE THUS
 X X X X X    STATISTICS FROM LOWER LIMIT TO END OF RUN     X X X X X
        ELSE
        LIST N.CQ, N.RQ, N.RAD.CHANNEL.QU, N.TQ, N.ONE.QU, N.LOSS.QU, N.BC,
            N.LAPG
        SKIP 3 LINES
        FOR I = TRANSIENT (DISCIPLINE) +1 TO NO.OF.QUERIES
            COMPUTE AV.RESP AS THE MEAN AND SD.RESP AS THE STD.DEV OF
                  RESPONSE (I)
        SKIP 2 LINES
        PRINT 2 LINES WITH AV.RESP AND SD.RESP THUS
 AVERAGE RESPONSE TIME IS                                    *.*** SECS
 STANDARD DEVIATION OF RESPONSE TIME IS                      *.*** SECS
        SKIP 2 LINES
        PRINT 1 LINE WITH QR AND NO.OF.QUERIES AND TIME.V THUS
OUT OF      * QUERIES      * WERE PROCESSED DURING            *.** SECS
        PRINT 1 LINE WITH TEL.MAX, TEL.AVE, TEL.LOW THUS
TELEPHONE QUEUE: MAX=       *  MEAN =     *.******  MIN =       *
        FOR EACH CPU PRINT 1 LINE WITH CPU.MAX (CPU), CPU.AVE (CPU),
            CPU.LOW (CPU), CPU THUS
CPU       QUEUE: MAX=       *  MEAN =     *.******* MIN =       *  FOR CPU    *
        FOR EACH CPU PRINT 1 LINE WITH CPU.UTIL, CPU THUS
CPU UTILIZATION:      *.******  FOR CPU  *
        PRINT 1 LINE WITH ONE.MAX, ONE.AVE, ONE.LOW THUS
ONE       QUEUE: MAX=       *  MEAN =     *.*******MIN =        *
        PRINT 1 LINE WITH LOSS.MAX, LOSS.AVE, LOSS.LOW THUS
LOSS      QUEUE: MAX=       *  MEAN =     *.******* MIN =       *
        PRINT 1 LINE WITH BL.MAX, BL.AVE, BL.LOW THUS
BLOCK     QUEUE: MAX=       *  MEAN =     *.******* MIN =       *
        FOR EACH RAD PRINT 1 LINE WITH RAD.MAX (RAD), RAD.AVE (RAD), RAD.LOW
            (RAD), RAD THUS
RAD       QUEUE: MAX=       *  MEAN =     *.******* MIN =       *  FOR RAD    *
        FOR EACH RAD PRINT 1 LINE WITH RAD.MEAN (RAD), RAD THUS
RAD UTILIZATION:    *.******   FOR DEVICE  *
        FOR EACH RAD PRINT 1 LINE WITH SK.MAX (RAD), SK.AVE (RAD), SK.LOW (RAD),
            RAD THUS
SEEK TIME    MAX=    *.**** MEAN=   *.**** MIN=    *.**** FOR RAD *
        FOR EACH RAD.CHANNEL PRINT 1 LINE WITH CHAN.MAX ( RAD.CHANNEL ),
            CHAN.AVE ( RAD.CHANNEL ), CHAN.LOW ( RAD.CHANNEL ), RAD.CHANNEL THUS
CHANNEL   QUEUE: MAX=       *  MEAN =     *.******* MIN =       *  FOR CHANNEL    *
        FOR EACH RAD.CHANNEL PRINT 1 LINE WITH CHAN.MEAN ( RAD.CHANNEL ),
            RAD.CHANNEL THUS
CHANNEL UTILIZATION:  *.******  FOR CHANNEL  *
        PRINT 1 LINE WITH PG.MAX, PG.AVE, PG.LOW THUS
PAGE      QUEUE: MAX=       *  MEAN =     *.******* MIN =       *
        PRINT 1 LINE WITH INT.MAX, INT.AVE, INT.LOW THUS
DBMS INTERRUPTS: MAX=       *  MEAN =     *.******* MIN =       *
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                          06/28/76

   36          PRINT 1 LINE WITH BLK.MAX, BLK.AVE, BLK.LOW THUS
DBMS BLOCKINGS : MAX=       *  MEAN =     *.******* MIN =       *
   37          PRINT 1 LINE WITH PROG.ACCESS AND DATA.ACCESS THUS
NUMBER OF PROGRAM PAGE-INS =         *  NUMBER OF DATA PAGE-INS =              *
   38          LIST MANY
   39          FOR I=1 TO N.RAD COMPUTE SUM = SUM OF MANY (I)
   40          FOR I= 1 TO N.RAD LET MANY (I) = MANY (I) / SUM
   41          LIST MANY
   42          SKIP 3 LINES
   43          PRINT 1 LINE THUS
                    MATRIX CALLED EXPECTED
   44          PRINT 1 LINE THUS
RT BEGINNING   END          NUMBER       NUMBER       AVE BEGIN  AVE END    TIME
   45          SKIP 1 LINE
   46          FOR I = 10 TO LAST.ROUTINE PRINT 1 LINE WITH I, EXPECTED (I,1),
                    EXPECTED (I,2), EXPECTED (I,3), EXPECTED (I,4),
   47               EXPECTED (I,1) / EXPECTED (I,3), EXPECTED (I,2) / EXPECTED (I,4),
   48               (EXPECTED (I,2) / EXPECTED (I,4)) - (EXPECTED (I,1)
   49                    / EXPECTED (I,3)) THUS
   50
 * *****.**    *****.**    *****.**    *****.**    ****.**    ****.**    ***.**
   51  RETURN END


               LOCAL VARIABLES OF THIS ROUTINE

   I           REAL      WORD 13       I.1          INTEGER   WORD  3
   I.2         INTEGER   WORD  4       I.3          INTEGER   WORD  5
   J.1         INTEGER   WORD  6       K.1          INTEGER   WORD  8
   K.2         INTEGER   WORD  9       K.3          INTEGER   WORD 10
   K.4         INTEGER   WORD 11       L.10         DOUBLE    WORD 21
   L.11        INTEGER   WORD 23       L.13         INTEGER   WORD 24
   L.2         INTEGER   WORD 12       L.40         DOUBLE    WORD 25
   L.41        DOUBLE    WORD 27       L.42         DOUBLE    WORD 29
   L.47        INTEGER   WORD 32       L.49         INTEGER   WORD 33
   L.7         DOUBLE    WORD 15       L.8          DOUBLE    WORD 17
   L.9         DOUBLE    WORD 19       N.1          INTEGER   WORD  7
   R.1         DOUBLE    WORD  1       SUM          REAL      WORD 31
```

LINE CACI SIMSCRIPT II.5 RELEASE 8D 06/28/76

```
EVENT TELEPHONE.CALL
        CREATE A QUERY
        LET QR = QR + 1
        LET QUERY.NUM ( QUERY ) = QR
        LET GEN.TIME ( QUERY ) = TIME.V
        FILE QUERY IN TEL.QU
        IF TEL.ARRIVAL = "FULL" OR CRT.CTR = N.CRT RETURN ELSE
        ADD 1 TO CRT.CTR
        SCHEDULE A TELEPHONE.CALL AT TIME.V + 1.0
        SCHEDULE AN ARRIVAL.AT.CRT (CRT.CTR) AT TIME.V + 1.0
 RETURN END
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                              06/28/76

 EVENT ARRIVAL.AT.CRT (X)  '' X IS THE CRT NUMBER
         IF TEL.QU IS EMPTY RETURN ELSE
         REMOVE FIRST QUERY FROM TEL.QU
         LET QUERY.ADDR (X) = QUERY
         LET AVAILABILITY (X) = 0
         'ONE'
         LET GREET.TIME (X) = EXPONENTIAL.F ( GREET.MEAN, E)
         IF GREET.TIME (X)<0.0 GO TO ONE ELSE
         'TWO'
         LET THINK.TIME (X) = EXPONENTIAL.F ( THINK.MEAN, E)
         IF THINK.TIME (X) < 0.0 GO TO TWO ELSE
         'THREE'
         LET TYPING.TIME (X) = EXPONENTIAL.F ( TYPING.MEAN, E)
         IF TYPING.TIME (X) < 0.0 GO TO THREE ELSE
         SCHEDULE A CRT.TRANS.READY (X) AT TIME.V + GREET.TIME (X) +
             THINK.TIME (X) + TYPING.TIME (X)
         LET CONVERSATION.END (QUERY) = TIME.V + GREET.TIME (X) + THINK.TIME (X)
 RETURN END

         LOCAL VARIABLES OF THIS ROUTINE

 X          INTEGER  WORD  1
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                06/28/76

      EVENT CRT.TRANS.READY (X)      ''X IS THE CRT NUMBER
           '' THIS ROUTINE WHEN RUN HANDLES THE CRT INTERRUPT
              CREATE A VIRTUAL.ROUTINE
                 LET SUPERVISOR (VR) = 1
                 LET TYPE (VR) = 1
                 CALL FILL (X)
              CALL CPU.CHECK (VR)
      RETURN END

              LOCAL VARIABLES OF THIS ROUTINE

 X          INTEGER  WORD  1
```

```
ROUTINE FILL (X)
        LET NEXT.VR.NO = NEXT.VR.NO +1
        LET VR.NO (VR)= NEXT.VR.NO
        LET CREATE.TM (VR) = TIME.V
        LET CRT.NUMBER (VR) = X
        'UP'
        IF TYPE (VR)= 1 OR TYPE (VR) = 6 OR TYPE (VR) = 10
        LET CPU.TIME.TO.GO(VR)=NORMAL.F( OPERATIONS ( WORK.LOAD, TYPE (VR),1),
             OPERATIONS ( WORK.LOAD, TYPE (VR),2),E) * SPEED (1)
        IF CPU.TIME.TO.GO (VR) < 0.0 GO TO UP ELSE
        IF TYPE (VR)< 10 LET TME.TILL.BLOCK (VR) = CPU.TIME.TC.GO (VR) ELSE
        IF TYPE (VR)= 10
        LET TME.TILL.BLOCK (VR) = CPU.TIME.TC.GO (VR) * BLOCKING.PERCENT
             ( BLOCKING.FACTOR )
        IF CHANGE.TO.GOOD = 1
        FOR I=10 TO LAST.ROUTINE FOR J=1 TO 2 UNLESS EXPECTED (I,4)=0.0
             DO LET EXPECTED (I,J) = EXPECTED (I,J)/ EXPECTED (I,J+2)
             LET EXPECTED (I,J+2) = 1.0
             LOOP
             LET CHANGE.TO.GOOD = 0 ELSE
        ADD TIME.V - CONVERSATION.END ( QUERY.ADDR (X)) TO EXPECTED ( 10,1)
        ADD 1.0 TO EXPECTED (10,2)
        ELSE
        JUMP AHEAD ELSE
        IF TYPE (VR)= 2 OR TYPE (VR) = 3 OR TYPE (VR) = 5
        IF DISCIPLINE = 1
             LET CPU.TIME.TO.GO (VR)= OPERATIONS ( WORK.LOAD, TYPE (VR), 1)
                    * SPEED (1) ELSE
        IF DISCIPLINE =2 LET CPU.TIME.TO.GO (VR) = ( OPERATIONS ( WORK.LOAD,
             TYPE (VR),1)+(OV.LOSS +(LINKER*N.LOSS.QU/2.0)))* SPEED (1) ELSE
        IF DISCIPLINE =3 LET CPU.TIME.TO.GO (VR)= ( OPERATIONS ( WORK.LOAD,
             TYPE (VR),1)+(OV.BLOCK.LOSS + (LINKER*N.LOSS.QU)))*SPEED(1)
                 ELSE
             LET TME.TILL.BLOCK (VR) = CPU.TIME.TO.GO (VR)
        JUMP AHEAD ELSE
        IF TYPE (VR) = 4
        LET CPU.TIME.TO.GO (VR)=
             OPERATIONS ( WORK.LOAD, TYPE (VR),1) * SPEED (1)
             LET TME.TILL.BLOCK (VR) = CPU.TIME.TO.GO (VR)
        JUMP AHEAD ELSE
        IF TYPE (VR) = 11
             LET TOTAL.PAGES.TO.BE.CREATED (VR) = 1 + TRUNC.F( EXPONENTIAL.F
                    ( DATA.FAULTS, 1 ))
        IF QUERY.NUM ( QUERY.ADDR (X))  < SHUT.OFF.TRACE
             SKIP 1 LINE PRINT 1 LINE WITH
             TOTAL.PAGES.TO.BE.CREATED (VR) AND QUERY.NUM ( QUERY.ADDR (X)) THUS
TOTAL PAGES TO BE CREATED IS      *   FOR QUERY NUMBER        *
             ELSE
        IF CHANGE.TO.GOOD = 1
        FOR I=10 TO LAST.ROUTINE FOR J=1 TO 2 UNLESS EXPECTED (I,4)=0.0
             DO LET EXPECTED (I,J) = EXPECTED (I,J)/ EXPECTED (I,J+2)
             LET EXPECTED (I,J+2) = 1.0
             LOOP
```

```
            LET CHANGE.TO.GOOD = 0 ELSE
        ADD TIME.V - CONVERSATION.END ( QUERY.ADDR (X)) TO EXPECTED ( 11,1)
        ADD 1.0 TO EXPECTED (11,3)
            '' CALCULATE TIME FOR DIRECTORY LOOK UP AND LINK TO RAD QUEUE
            LET CPU.TIME.TO.GO (VR)= (( OPERATIONS ( WORK.LOAD, TYPE (VP),1) *
            TOTAL.PAGES.TO.BE.CREATED )+ (  TOTAL.PAGES.TO.BE.CREATED *
            OPERATIONS  ( WORK.LOAD,4,1))) * SPEED (1)
            LET TME.TILL.BLOCK (VR)= CPU.TIME.TO.GO (VP)
            '' THIS ROUTINE IS TOO SHORT TO INCUR A PROGRAM PAGE BLOCK
            '' ALSO NO VARIANCE
        JUMP AHEAD ELSE
        'UPP'
        IF TYPE (VR) = 12
            LET CPU.TIME.TO.GO (VR) = NORMAL.F( OPERATIONS ( WORK.LOAD, 12, 1),
                     OPERATIONS ( WORK.LOAD, 12, 2), E) *  SPEED (1) *
                     TOTAL.PAGES.TO.BE.CREATED (VR)
        IF CPU.TIME.TO.GO (VR)< 0.0 GO TO UPP ELSE
            LET TME.TILL.BLOCK (VR) = CPU.TIME.TO.GO (VR) * BLOCKING.PERCENT
                     ( BLOCKING.FACTOR )
        IF CHANCE.TO.GOOD = 1
        FOR I=10 TO LAST.ROUTINE FOR J=1 TO 2 UNLESS EXPECTED (I,4)=0.0
            DO LET EXPECTED (I,J) = EXPECTED (I,J)/ EXPECTED (I,J+2)
            LET EXPECTED (I,J+2) = 1.0
            LOOP
            LET CHANGE.TO.GOOD = 0 ELSE
        ADD TIME.V - CONVERSATION.END ( QUERY.ADDR (X)) TO EXPECTED ( 12,1)
        ADD 1.0 TO EXPECTED (12,3)
        ELSE
    HERE LET TOTAL.CPU.TIME.NEEDED (VR) = CPU.TIME.TO.GO (VP)
RETURN END
```

LOCAL VARIABLES OF THIS ROUTINE

I	INTEGER	WORD	3	I.1	INTEGER	WORD	9
I.2	INTEGER	WORD	10	I.3	INTEGER	WORD	11
J	INTEGER	WORD	4	J.1	INTEGER	WORD	12
K.1	INTEGER	WORD	14	K.2	INTEGER	WORD	15
K.3	INTEGER	WORD	16	K.4	INTEGER	WORD	17
L.6	INTEGER	WORD	5	N.1	INTEGER	WORD	13
R.1	DOUBLE	WORD	7	X	INTEGER	WORD	1

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                              06/28/76

ROUTINE CPU.CHECK (ROUT)
        IF CPU.QU (1) IS NOT EMPTY FILE ROUT IN CPU.QU (1) RETURN
                                                         ELSE
        IF ROUT.ADDR (1) = 0 SCHEDULE A RUN.IT ( ROUT) NOW RETURN
           ELSE IF SUPERVISOR ( ROUT.ADDR (1)) = 1
                   FILE ROUT IN CPU.QU (1) RETURN ELSE
            SCHEDULE A RUN.IT ( ROUT) NOW
RETURN END

          LOCAL VARIABLES OF THIS ROUTINE

ROUT        INTEGER  WORD  1
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                  06/28/76

EVENT CRT.TRANS.FINISHED ( CATH )
          '' THIS EVENT IS SCHEDULED TO OCCUR AT THE EARLIEST WHEN THE QUERY
          '' HAS ARRIVED IN MAIN MEMORY
          '' WHEN THE ROUTINE CREATED HERE IS LATER RUN IT LINKS THE FIRST
          '' CBMS ROUTINE INTO THE APPROPRIATE QUEUE(S)
          CREATE A VIRTUAL.ROUTINE
               LET SUPERVISOR (VR) = 1
               LET TYPE (VR) = 2
               CALL FILL ( CATH )
               CALL CPU.CHECK (VR)
RETURN END

            LOCAL VARIABLES OF THIS ROUTINE

CATH        INTEGER  WORD  1
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                              06/26/76

EVENT RUN.IT ( VIRT )
        LET P = VIRT
        IF ROUT.ADD (1) = 0 GO TO OFF ELSE  '' A ROUTINE IS ALREADY ON CPU
        '' BEGINNING OF ERROR CHECKING
        IF SUPERVISOR (P) = 0 AND EV.ADDR (1) NE 0
           PRINT 2 LINES WITH VR.NO (P) AND VR.NO ( EV.ADDR (1)) THUS
 ---- ERROR IN RUN.IT EVENT      NEW DBMS ROUTINE NO =          *
      RUN ABORTED                OLD DBMS ROUTINE NO =          *
        STOP ELSE
        '' END OF ERROR CHECKING
        IF SUPERVISOR (P) = 0 IF DISCIPLINE =1 FILE P FIRST IN ONE.QU ELSE
            IF DISCIPLINE = 2 FILE P IN LOSS.QU ELSE
            IF DISCIPLINE = 3 FILE P IN LOSS.QU FILE P IN BLOCK.QU ELSE
            RETURN ELSE
        IF SUPERVISOR ( ROUT.ADDR (1)) =1
        FILE P IN CPU.QU (1) RETURN ELSE
        '' A DBMS ROUTINE IS TO BE INTERRUPTED
        ''TME.TILL.BLOCK IS RUN TIME NOT SIMULATION TIME
        LET TME.TILL.BLOCK ( ROUT.ADD (1))= TME.TILL.BLOCK ( ROUT.ADD (1)) -
                  (TIME.V - TIME.ON (1))
        IF TME.TILL.BLOCK ( ROUT.ADD (1)) < 0.000001
            LET TME.TILL.BLOCK ( ROUT.ADD (1)) = 0.0 ELSE
        LET CPU.TIME.TO.GO ( ROUT.ADD (1))= CPU.TIME.TO.GO ( ROUT.ADD (1)) -
                  (TIME.V - TIME.ON (1))
        IF CPU.TIME.TO.GO ( ROUT.ADD (1)) < 0.000001
            LET CPU.TIME.TO.GO ( ROUT.ADD (1)) = 0.0 ELSE
        LET INTERRUPTS.PER.RUN ( ROUT.ADD (1))= INTERRUPTS.PER.RUN
                  ( ROUT.ADD (1)) + 1
        IF EV.ADDR (1) NE 0 PRINT 1 LINE THUS
------- A ROUTINE IS ALREADY ON TEMPORARY HOLD
        ELSE LET EV.ADDR (1) = ROUT.ADDR (1)
        CANCEL OFF.CPU
        DESTROY OFF.CPU
        'OFF'
        SCHEDULE AN OFF.CPU (P) AT TIME.V + MIN.F (CPU.TIME.TO.GO (P),
                                                   TME.TILL.BLOCK (P))

        LET STATUS (1)   = 1 LET TIME.ON (1) = TIME.V
        LET ROUT.ADDR (1) = P
RETURN END


            LOCAL VARIABLES OF THIS ROUTINE

I.1      INTEGER  WORD  7     I.2      INTEGER  WORD  8
I.3      INTEGER  WORD  9     J.1      INTEGER  WORD 10
K.1      INTEGER  WORD 12     K.2      INTEGER  WORD 13
K.3      INTEGER  WORD 14     K.4      INTEGER  WORD 15
N.1      INTEGER  WORD 11     P        INTEGER  WORD  3
P.1      DOUBLE   WORD  5     VIRT     INTEGER  WORD  1
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 80                                        06/28/76

      EVENT OFF.CPU (P)
              '' UPDATE TIME ATTRIBUTES OF ROUTINE LEAVING CPU
              LET CPU.TIME.TO.GO (P) = CPU.TIME.TO.GO (P) - (TIME.V - TIME.ON (1))
              LET TME.TILL.BLOCK (P) = 0.0
              IF CPU.TIME.TO.GO (P) < 0.000001 LET CPU.TIME.TO.GO (P) = 0.0 ELSE
              CALL PIGGY.BACK (P)
              IF CPU.QU  (1)  IS NOT EMPTY
              REMOVE FIRST VIRTUAL.ROUTINE FROM CPU.QU (1)
              SCHEDULE A RUN.IT ( VIRTUAL.ROUTINE ) NOW GO TO OVER ELSE
              '' CPU QUEUE FOR SUPERVISOR ROUTINES IS EMPTY
              IF EV.ADDR (1) NE 0
                   SCHEDULE A RUN.IT ( EV.ADDR (1)) NOW
                   LET EV.ADDR (1) =0
                   GO TO OVER ELSE
              IF DISCIPLINE = 1 AND ONE.QU IS NOT EMPTY
                      REMOVE FIRST VIRTUAL.ROUTINE FROM ONE.QU
                      SCHEDULE A RUN.IT ( VR ) NOW GO TO OVER ELSE
              IF DISCIPLINE =2 AND LOSS.QU IS NOT EMPTY
                   REMOVE FIRST VR FROM LOSS.QU
                   SCHEDULE A RUN.IT (VR) NOW GO TO OVER ELSE
              IF DISCIPLINE =3 AND LOSS.QU IS NOT EMPTY
                   LET YES = 0
                   FOR EACH RAD IF RAD.QU (RAD) IS EMPTY LET YES=1 ELSE
                   IF YES =0 REMOVE FIRST VR FROM LOSS.QU
                            REMOVE THIS VR FROM BLOCK.QU
                            SCHEDULE A RUN.IT (VR) NOW GO TO OVER ELSE
                   REMOVE FIRST VR FROM BLOCK.QU
                   REMOVE THIS VR FROM LOSS.QU
                   SCHEDULE A RUN.IT (VR) NOW GO TO OVER ELSE
        'OVER' LET STATUS (1) = 0 LET ROUT.ALO (1) = 0
        RETURN END

              LOCAL VARIABLES OF THIS ROUTINE

P            INTEGER WORD  1       YES        INTEGER  WORD  3
```

```
ROUTINE PIGGY.BACK (P)
        DEFINE RAND AND TIME AS REAL VARIABLES
        DEFINE CAPTURE AS A DOUBLE VARIABLE
        IF TYPE (P) = 1  '' A CRT IS READY TO TRANSMIT
              LET Q = CRT.NUMBER (P)
              SCHEDULE A CRT.TRANS.FINISHED ( Q ) AT TIME.V + TRANS.TIME
              DESTROY THE VIRTUAL.ROUTINE CALLED P  RETURN ELSE
        IF TYPE (P) = 2  ''A CRT HAS FINISHED TRANSMITTING
            '' AND THE FIRST DBMS ROUTINE HAS BEEN CREATED AND LINKED INTO
            ''THE DBMS QUEUE(S)
              CREATE A VIRTUAL.ROUTINE
                   LET SUPERVISOR (VR) = 0
                   LET TYPE (VR) = 10
                   LET X = CRT.NUMBER (P)
                   CALL FILL (X)
                   CALL INSERT.QUEUE (VR)
              DESTROY THE VIRTUAL.ROUTINE CALLED P  RETURN ELSE
                   '' A LINK BACK TO DBMS QUEUE(S) OF ANY DBMS ROUTINE
                   '' INCLUDING A NEW EDIT ROUTINE
        IF TYPE (P) = 3 OR TYPE (P) = 5
            LET RAD.DEVICE (P) =0 LET RAD.CHAN (P) = 0
            LET Q=SERVICING.ROUTINE.ADDR (P)
            CALL INSERT.QUEUE (Q)
            DESTROY THE VR CALLED P     '' THE SUPERVISOR ROUTINE
            RETURN ELSE
        IF TYPE (P) = 4    '' A LINKING OF A DBMS ROUTINE TO A RAD.QUEUE IS NOW
            ''PERFORMED TO HANDLED THE BLOCK THAT HAD OCCURRED
            LET Q = SERVICING.ROUTINE.ADDR (P)
            LET TIMES.BLOCKED (Q) = TIMES.BLOCKED (Q) +1
        ''CALCULATE NEXT BLOCK TIME AMOUNT
        LET TME.TILL.BLOCK (Q) = TOTAL.CPU.TIME.NEEDED (Q) * BLOCKING.PERCENT
                   ( BLOCKING.FACTOR )
      HERE
            LET CHANN= CHANNEL.HOOK (1)
            LET RAD.DEVICE (Q) =1
            LET RAD.CHAN (Q) = CHANN
        '' SERVICE BEGINS WHEN VIRTUAL ROUTINE GOES TO RAD SUBSYSTEM
        IF GOOD = 1 AND ARR.TEST = 0 CALL ARRIV.SERV (1,1,TIME)
            LET SERV.START (Q) = TIME.V      ELSE
            ''TIME IS A DUMMY VARIABLE HERE
        IF RAD.BUSY (1) = 1 OR BUSY.2 (CHANN) = 1
        FILE Q IN RAD.QU (1) DESTROY THE VR CALLED P RETURN ELSE
        IF ARM ( RAD.DEVICE (Q))=-1
             SCHEDULE A LATENCY.END (Q) AT TIME.V+(0.5* RAD.ROTATION.SPEED (1))
             JUMP AHEAD ELSE
        IF ARM( RAD.DEVICE (Q)) NE -1 AND RPS ( RAD.DEVICE (Q))=1
             SCHEDULE A LATENCY.END (Q)
             AT TIME.V + SEEK ( RAD.DEVICE (Q)) + (0.5* RAD.ROTATION.SPEED
             ( RAD.DEVICE (Q)))  JUMP AHEAD ELSE
        IF ARM ( RAD.DEVICE (Q)) NE -1 AND RPS ( RAD.DEVICE (Q)) NE 1
             SCHEDULE  A SEEK.END (Q) AT TIME.V + SEEK ( RAD.DEVICE (Q)) ELSE
        HERE LET RAD.BUSY ( RAD.DEVICE (Q)) =1
             FILE Q IN RAD.CHANNEL.QU (CHANN)
```

```
          DESTROY THE VR CALLED P RETURN ELSE
      IF TYPE (P) = 6  '' BEGINNING OF ANSWER BEING SENT TO CRT
          LET C = SERVICING.ROUTINE.ADDR (P)
          SCHEDULE AN ANSWER (C) AT TIME.V + TRANS.TIME
          DESTROY THE VIRTUAL.ROUTINE CALLED P
          RETURN ELSE
      IF CPU.TIME.TO.GO (P) = 0.0 AND TYPE (P) = 10
          ADD TIME.V - CONVERSATION.END ( QUERY.ADDR ( CRT.NUMBER (P)))
                 TO EXPECTED ( 10,2)
          ADD 1 TO EXPECTED (10,4)
          CREATE A VIRTUAL.ROUTINE    '' THE NEXT DBMS ROUTINE
          LET SUPERVISOR (VR) = 0
          LET X = CRT.NUMBER (P)
          LET TYPE (VR) = TYPE (P) + 1
          CALL FILL (X)
          LET C = VR
      '' CREATE A VIRTUAL ROUTINE WHICH WHEN RUN WILL FILE THE DBMS ROUTINE
          '' INTO ITS APPROPRIATE DBMS CPU QUEUE(S)
          LET X = CRT.NUMBER (P)
          LET Y=5  CALL LINK (X,Y)
          LET SERVICING.ROUTINE.ADDR (VR) = 0
      FILE VR IN CPU.QU (1)
      DESTROY THE VR CALLED P RETURN ELSE
      IF CPU.TIME.TO.GO (P) =0.0 AND TYPE (P) = 11
          ADD TIME.V - CONVERSATION.END ( QUERY.ADDR ( CRT.NUMBER (P)))
                 TO EXPECTED ( 11,2)
          ADD 1 TO EXPECTED (11,4)
          ''WE ASSUME THAT WHEN TYPE 11 RAN IT USED ITS TIME TO NOT ONLY LOOK
          '' UP THE DIRECTORY BUT ALSO TO PUT THE DATA PAGES INTO THEIR
          ''RESPECTIVE RAD QUEUES
          '' WE NOW PUT THE DATA PAGE REQUESTS INTO THEIR RAD QUEUES
      CREATE A PAGE.ENTITY
          LET NUM.QUERY ( PAGE.ENTITY )= QUERY.NUM ( QUERY.ADDR ( CRT.NUMBER
                 (P)))
          LET PAGES ( PAGE.ENTITY ) = TOTAL.PAGES.TO.BE.CREATED (P)
          FILE PAGE.ENTITY IN LAST.PAGE
      IF BALANCE.RAD = 1 FOR EVERY RAD COMPUTE CAPTURE = MIN OF RAD.MEAN
          (RAD)
      FOR EVERY RAD WITH RAD.MEAN (RAD) = CAPTURE FIND DEVICE = THE FIRST RAD
      ELSE
          FOR K=1 TO TOTAL.PAGES.TO.BE.CREATED (P)
          DO
      CREATE A VIRTUAL.ROUTINE
          LET NEXT.VR.NO = NEXT.VR.NO + 1
          LET VR.NO (VR) = NEXT.VR.NO
          LET SUPERVISOR (VR) =0
          LET TYPE (VR) = 20
          LET CRT.NUMBER (VR) = CRT.NUMBER (P)
      LET TOTAL.PAGES.TO.BE.CREATED (VR) = TOTAL.PAGES.TO.BE.CREATED (P)
      IF BALANCE.RAD = 1 GO TO CONTINUE ELSE
            LET RAND = UNIFORM.F ( 0.0, 1.0, E)
      FOR EACH RAD WITH ACCESS.TABLE ( RAD,1) > RAND OR
          ACCESS.TABLE ( RAD,1) = RAND FIND DEVICE = THE FIRST RAD
```

CACI SIMSCRIPT II.5 RELEASE 8D 06/28/76

```
        IF NONE PRINT 1 LINE THUS
       ERROR ---- IN ROUTINE PIGGY.BACK   DEVICE CANT BE FOUND
        ELSE
        'CONTINUE'
        LET CHANN = CHANNEL.HOOK ( DEVICE )
        LET RAD.DEVICE (VR) = DEVICE LET RAD.CHAN (VR) = CHANN
        IF GOOD = 1 AND ARR.TEST = 0 CALL ARRIV.SERV ( DEVICE, 1, TIME)
            LET SERV.START (VR) = TIME.V      ELSE
            ''TIME IS A DUMMY VARIABLE HERE
        IF RAD.BUSY ( DEVICE ) = 1 OR BUSY.2 ( CHANN ) = 1
        FILE VR IN RAD.QU ( DEVICE ) CYCLE ELSE
        IF ARM ( DEVICE ) = -1 SCHEDULE A LATENCY.END (VR) AT TIME.V
            + (0.5 * RAD.ROTATION.SPEED (DEVICE ))  JUMP AHEAD ELSE
        IF ARM ( DEVICE ) NE -1 AND RPS ( DEVICE ) = 1 SCHEDULE A LATENCY.END
            (VR) AT TIME.V + SEEK (DEVICE) + (0.5 * RAD.ROTATION.SPEED (DEVICE))
            JUMP AHEAD ELSE
        IF ARM ( DEVICE ) NE -1 AND RPS ( DEVICE ) = 0 SCHEDULE A SEEK.END (VR)
            AT TIME.V + SEEK ( DEVICE ) ELSE
     HERE LET RAD.BUSY (DEVICE) = 1
        FILE VR IN RAD.CHANNEL.QU (CHANN)
            '' THE EDIT ROUTINE ONLY GETS CREATED AFTER THE LAST DATA
            '' PAGE HAS BEEN ACCESSED
            LOOP
        DESTROY A VR CALLED P RETURN ELSE
        IF CPU.TIME.TO.GO (P) = 0.0 AND TYPE (P) = 12
                '' END OF LAST DBMS ROUTINE
            ADD TIME.V - CONVERSATION.END ( QUERY.ADDR ( CRT.NUMBER (P)))
                  TO EXPECTED ( 12,2)
            ADD 1 TO EXPECTED (12,4)
            LET RUN.INTERRUPTS = INTERRUPTS.PER.RUN (P)
            LET RUN.BLOCKED = TIMES.BLOCKED (P)
              CREATE A VIRTUAL.ROUTINE  '' TC HANDLE INTERRUPT FOR ANSWER(QUERY)
                    LET SUPERVISOR (VR) = 1
                    LET TYPE (VR) = 6
                    LET X = CRT.NUMBER (P)
                    CALL FILL (X)
                    LET SERVICING.ROUTINE.ADDR (VR) = P
                    FILE VR IN CPU.QU (1)
                    RETURN ELSE
        IF CPU.TIME.TO.GO (P) > 0.0
        '' A BLOCK ON A DBMS ROUTINE OCCURRED ( EITHER A CLA OR EDITING ROUTINE)
        '' BLOCK OR A DATA PAGE-IN ROUTINE BLOCKED DUE TO EITHER A NEEDED
        '' CREATE A SUPERVISOR ROUTINE TO HANDLE THE BLOCK ITSELF
        IF TYPE (P) > 9 LET X = CRT.NUMBER (P)
            LET Y=4
        CALL LINK (X,Y)
        LET SERVICING.ROUTINE.ADDR (VR) = P
        FILE VR IN CPU.QU  (1)
        RETURN ELSE
        ELSE
RETURN END
```

LINE CACI SIMSCRIPT II.5 RELEASE 8D 06/28/76

LOCAL VARIABLES OF THIS ROUTINE

CAPTURE	DOUBLE	WORD	5	CHANN	INTEGER	WORD	9
DEVICE	INTEGER	WORD	17	I.1	INTEGER	WORD	21
I.2	INTEGER	WORD	22	I.3	INTEGER	WORD	23
J.1	INTEGER	WORD	24	K	INTEGER	WORD	18
K.1	INTEGER	WORD	26	K.2	INTEGER	WORD	27
K.3	INTEGER	WORD	28	K.4	INTEGER	WORD	29
L.4	DOUBLE	WORD	11	L.5	DOUBLE	WORD	13
L.6	DOUBLE	WORD	15	N.1	INTEGER	WORD	25
P	INTEGER	WORD	1	Q	INTEGER	WORD	7
R.1	DOUBLE	WORD	19	RAND	REAL	WORD	3
TIME	REAL	WORD	4	X	INTEGER	WORD	8
Y	INTEGER	WORD	10				

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                    06/28/76

  ROUTINE INSERT.QUEUE (K)
  '' CALCULATES THE PRIORITY OF THE VIRTUAL ROUTINE AND FILES IT INTO ITS
  '' APPROPRIATE QUEUE(S)
         DEFINE BEG, TERM, LINE.EXP, LINE.ACT, INDEPENDENT, EXPONENT
              AND THIS.LOSS, EXP AND INTER AND DENOM
              AS REAL VARIABLES
         IF DISCIPLINE = 1 FILE K IN ONE.QU RETURN ELSE
         IF DISCIPLINE = 2 OR DISCIPLINE = 3  '' CALCULATE PRIORITY
         LET BEG = EXPECTED ( TYPE(K),1) / EXPECTED (TYPE(K),3)
         LET TERM = EXPECTED (TYPE(K),2) / EXPECTED (TYPE(K),4)
         LET LINE.EXP = BEG +((( TOTAL.CPU.TIME.NEEDED (K) - CPU.TIME.TO.GO (K))
              / TOTAL.CPU.TIME.NEEDED (K)) * (TERM - BEG ))
         LET LINE.ACT = TIME.V - CONVERSATION.END ( QUERY.ADDR ( CRT.NUMBER (K)))
         LET INDEPENDENT = LINE.ACT - LINE.EXP + MEAN.RESPONSE
         LET EXPONENT = ABS.F ( -CCC * INDEPENDENT)
         '' TO PREVENT OVERFLOW
         IF EXPONENT > 170.0 LET EXPONENT = 170.0 ELSE
         LET INTER = BBB ** EXPONENT
         LET THIS.LOSS = AAA/(AAA+(1.0/INTER))
         LET EXP = ABS.F ( -CCC * MEAN.RESPONSE )
         IF EXP > 170.0 LET EXP = 170.0 ELSE
         LET DENOM = BBB ** EXP
         LET LOSS.RANK (K) = THIS.LOSS - (AAA/(AAA+(1.0/DENOM)))
         FILE K IN LOSS.QU
         ELSE
         IF DISCIPLINE = 3 LET BLOCK.RANK (K) = THE.TILL.BLOCK (K)
         FILE K IN BLOCK.QU ELSE
         RETURN END


              LOCAL VARIABLES OF THIS ROUTINE

 BEG          REAL      WORD  3      DENOM       REAL      WORD 12
 EXP          REAL      WORD 10      EXPONENT    REAL      WORD  8
 INDEPENDEN   REAL      WORD  7      INTER       REAL      WORD 11
 K            INTEGER   WORD  1      LINE.ACT    REAL      WORD  6
 LINE.EXP     REAL      WORD  5      TERM        REAL      WORD  4
 THIS.LOSS    REAL      WORD  9
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                   06/28/76

      ROUTINE LINK (X,Y)
      '' CREATES A SUPERVISOR ROUTINE WHICH WHEN RUN WILL BE USED FOR LINKING
      ''        1. NEW DBMS ROUTINE GENERATED FROM A PREVIOUS DBMS ROUTINE
      ''        2. A ROUTINE BACK FROM RAD TO DBMS QUEUE
      ''        3. A BLOCKED DBMS ROUTINE INTO A READ QUEUE
                CREATE A VIRTUAL.ROUTINE
                     LET SUPERVISOR (VR) =1
                     LET TYPE (VR) = Y
                     CALL FILL (X)    RETURN END

                LOCAL VARIABLES OF THIS ROUTINE

 X           INTEGER  WORD  1       Y           INTEGER  WORD  3
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                   06/28/76

EVENT SEEK.END (P)
          '' THIS EVENT OCCURS IN THE SIMULATION ONLY FOR DEVICES WITHOUT RPS.
          '' DEVICES WITH ACCESS ARMS WHICH ALSO HAVE RPS HAVE HAD A COMBINED
          '' SEEK AND LATENCY TIME CALCULATED.
               '' A CHANNEL MAY BE BUSY WITH ANOTHER ACCESSION
          IF BUSY.2 ( RAD.CHAN (P)) = 1 LET WANT.CHANNEL (P) = 1
          RETURN ELSE
          LET BUSY.2 ( RAD.CHAN (P) ) = 1
          SCHEDULE A     READ.END (P)    AT TIME.V + ( 0.5 * RAD.ROTATION.SPEED
                 ( RAD.DEVICE (P))) + ( RAD.ROTATION.SPEED ( RAD.DEVICE (P)) /
               PAGES.PER.TRACK ( RAD.DEVICE (P)))
RETURN END

           LOCAL VARIABLES OF THIS ROUTINE

P         INTEGER  WORD  1
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                06/28/76

  ROUTINE SEEK ( DEVICE )
          DEFINE Q AND PRO AS REAL VARIABLES
          '' FIND SEEK DISTANCE
          LET PRO=UNIFORM.F( ACCESS.TABLE (DEVICE,2), ACCESS.TABLE
               ( DEVICE,1),E)
          FOR I=2 TO NO.OF.CYL (DEVICE) +1 WITH ACCESS.TABLE (DEVICE,I)> PRO
               OR ACCESS.TABLE (DEVICE,I)= PRO
               FIND CYL = THE FIRST I
               LET CYL = CYL - 1
          LET DISTANCE = ABS.F ( ARM (DEVICE) - CYL )
          '' CALCULATE TIME TO SEEK
          IF DISTANCE = 0 LET SEEK.TIME (DEVICE) = 0.0 GO TO OVER ELSE
          LET Q = AA + BB * DISTANCE
          LET SEEK.TIME (DEVICE) = Q
          IF LEVEL (N.RAD) = 2
               IF LEVEL (DEVICE) = 1 LET SEEK.TIME (DEVICE)= 30.0/75.0 * Q ELSE
               IF LEVEL (DEVICE) = 2 LET SEEK.TIME (DEVICE)= Q ELSE
               JUMP AHEAD ELSE
          IF LEVEL (DEVICE) = 3 LET SEEK.TIME (DEVICE) = Q ELSE
          IF LEVEL (DEVICE) = 2 LET SEEK.TIME (DEVICE) = 30.0/75.0 * Q ELSE
          HERE
                       ''  TIME AS A FUNCTION OF SEEK DISTANCE
          LET ARM (DEVICE) = CYL
 'OVER' RETURN WITH SEEK.TIME (DEVICE)
          END

              LOCAL VARIABLES OF THIS ROUTINE

CYL          INTEGER   WORD  6       DEVICE      INTEGER   WORD  1
DISTANCE     INTEGER   WORD  7       I           INTEGER   WORD  5
PRO          REAL      WORD  4       Q           REAL      WORD  3
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8U                                    06/28/76

EVENT LATENCY.END (P)
          ''FOR DEVICES WITH RPS
          '' A CHANNEL MAY BE BUSY WITH ANOTHER ACCESSION
          IF BUSY.2 ( RAD.CHAN (P)) = 1 LET WANT.CHANNEL (P) = 2
          RETURN ELSE
          LET BUSY.2 ( RAD.CHAN (P) ) = 1
          SCHEDULE A      READ.END (P) AT TIME.V + ( RAD.ROTATION.SPEED
                 ( RAD.DEVICE (P)) / PAGES.PER.TRACK ( RAD.DEVICE (P)))
RETURN END

           LOCAL VARIABLES OF THIS ROUTINE

P       INTEGER  WORD  1
```

```
EVENT READ.END    (P)
        DEFINE TIME AS A REAL VARIABLE
        DEFINE HOLD.CHANNEL AS AN INTEGER VARIABLE
        LET HOLD.CHANNEL = RAD.CHAN (P)
        ADD 1 TO MANY ( RAD.DEVICE (P))
        LET RAD.BUSY ( RAD.DEVICE (P)) = 0
        LET BUSY.2   ( RAD.CHAN   (P)) = 0
        IF GOOD = 1 AND ARR.TEST = 0 AND SERV.START (P) NE 0.0
            LET TIME = TIME.V - SERV.START (P)
            CALL ARRIV.SERV ( RAD.DEVICE (P),2,TIME) ELSE
        IF TYPE (P) = 20  '' A DATA PAGE IN
            ADD 1 TO DATA.ACCESS
            FOR EACH PAGE.ENTITY IN LAST.PAGE
                    WITH NUM.QUERY ( PAGE,ENTITY )= QUERY,NUM ( QUERY.ADDR
                    ( CRT.NUMBER (P))) FIND THE FIRST CASE
            SUBTRACT 1 FROM PAGES ( PAGE.ENTITY )
            IF PAGES ( PAGE.ENTITY ) > 0
            REMOVE THIS P FROM RAD.CHANNEL.QU ( RAD.CHAN (P))
            DESTROY THE VIRTUAL.ROUTINE CALLED P GO TO BOTTOM
            ELSE
        REMOVE THIS PAGE.ENTITY FROM LAST.PAGE
        DESTROY THE PAGE.ENTITY
                    CREATE A VIRTUAL.ROUTINE     '' EDIT ROUTINE
                    LET SUPERVISOR (VR) = 0
                    LET TYPE (VR) = 12
        LET TOTAL.PAGES.TO.BE.CREATED (VR) = TOTAL.PAGES.TO.BE.CREATED (P)
                    LET X = CRT.NUMBER (P)
                    CALL FILL (X)
                    LET Q = VR
        ''
        ''              CREATE A SUPERVISOR ROUTINE TO LINK NEW EDIT ROUTINE
        ''              INTO APPROPRIATE QUEUE(S) AWAITING CPU SERVICE
        ''
        CREATE A VIRTUAL.ROUTINE
                    LET SUPERVISOR (VR) = 1
                    LET TYPE (VR) = 5
                    LET X= CRT.NUMBER (P)
                    CALL FILL (X) LET SERVICING.ROUTINE.ADDR (VR) = Q
                    CALL CPU.CHECK (VR)
                    REMOVE THIS P FROM RAD.CHANNEL.QU ( RAD.CHAN (P))
                    DESTROY VR CALLED P
                    GO TO BOTTOM
                    ELSE
        IF TYPE (P) NE 20  LET Q = P ELSE
        ADD 1 TO PROG.ACCESS
        '' A BLOCKED ROUTINE CAN NOW REJOIN THE DBMS CPU QUEUE(S)
        REMOVE THIS P FROM RAD.CHANNEL.QU ( RAD.CHAN (P))
            '' CREATE A SUPERVISOR ROUTINE TO LINK DBMS ROUTINE BACK INTO
            '' DBMS QUEUE(S)
        CREATE A VIRTUAL.ROUTINE LET SUPERVISOR (VR) = 1 LET TYPE (VR) = 3
        LET X= CRT.NUMBER (P)
        CALL FILL (X) LET SERVICING.ROUTINE.ADDR (VR) = P
        CALL CPU.CHECK (VR)
```

```
        'BOTTOM'
        ''HANDLE ACCESS REQUESTS FOR PENDING SEEK ENDS OR LATENCY ENDS FIRST
        IF RAD.CHANNEL.QU ( HOLD.CHANNEL) IS NOT EMPTY
            FOR EACH ROUT IN RAD.CHANNEL.QU (HOLD.CHANNEL)
                   WITH WANT.CHANNEL (ROUT) > 0 FIND ZZ = ROUT
                   IF NONE GO TO TRY.FURTHER ELSE
                   IF WANT.CHANNEL (ZZ)=1 LET BUSY.2 ( RAD.CHAN (ZZ))=1
                         SCHEDULE A READ.END (ZZ) AT TIME.V+(0.5*
                         RAD.ROTATION.SPEED ( RAD.DEVICE (ZZ))) + (
                         RAD.ROTATION.SPEED ( RAD.DEVICE (ZZ)) / PAGES.PER.TRACK
                   ( RAD.DEVICE (ZZ))) LET WANT.CHANNEL (ZZ)=0 GO TO DOWN ELSE
                   IF WANT.CHANNEL (ZZ)=2 LET BUSY.2 ( RAD.CHAN (ZZ)) = 1
                   SCHEDULE A READ.END (ZZ) AT TIME.V + ( RAD.ROTATION.SPEED
                   ( RAD.DEVICE (ZZ))/ PAGES.PER.TRACK ( RAD.DEVICE (ZZ)))
                   LET WANT.CHANNEL (ZZ)=0 GO TO DOWN ELSE
        'DOWN' RETURN
        ELSE
        'TRY.FURTHER'
        FOR EACH RAD WITH HOLD.CHANNEL = CHANNEL.HOOK (RAD)
            DO IF RAD.QU (RAD) IS NOT EMPTY AND RAD.BUSY (RAD) NE 1
                   REMOVE FIRST ROUT FROM RAD.QU (RAD)
                   IF ARM (RAD) = -1 SCHEDULE A LATENCY.END (ROUT)
                   AT TIME.V+(0.5* RAD.ROTATION.SPEED (RAD))
                   GO TO DROP ELSE
            IF ARM (RAD) NE -1 AND RPS (RAD) = 1
                   SCHEDULE A LATENCY.END (ROUT) AT TIME.V + SEEK (RAD)
                         + (0.5 * RAD.ROTATION.SPEED (RAD)) ELSE
                   IF ARM (RAD) NE -1 AND RPS (RAD) = 0
                   SCHEDULE A SEEK.END (ROUT) AT TIME.V + SEEK (RAD)
                         ELSE
        'DROP' LET RAD.BUSY (RAD) = 1
        FILE ROUT IN RAD.CHANNEL.QU ( HOLD.CHANNEL)
        ELSE LOOP
    RETURN END
```

LOCAL VARIABLES OF THIS ROUTINE

HOLD.CHANN	INTEGER	WORD	4	L.4	INTEGER	WORD	5
L.8	INTEGER	WORD	9	P	INTEGER	WORD	1
Q	INTEGER	WORD	7	ROUT	INTEGER	WORD	8
TIME	REAL	WORD	3	X	INTEGER	WORD	6
ZZ	INTEGER	WORD	10				

```
EVENT ANSWER (C)      '' A VIRTUAL ROUTINE ADDRESS
        DEFINE X, Y, M1, M2, TM AS REAL VARIABLES
        DEFINE M AS A 1-DIM REAL ARRAY
        IF CEASE.RUN = 1 DESTROY THE QUERY CALLED QUERY.ADDR ( CRT.NUMBER (Q))
            DESTROY THE VIRTUAL.ROUTINE CALLED C RETURN ELSE
        LET K = CRT.NUMBER (Q)
        IF DISCIPLINE > 1 AND NO.OF.QUERIES = 0 ADD 1 TO NO.OF.QUERIES
        FOR I =1 TO DIM.F ( FIR.RESPONSE (*)) COMPUTE RESPONSE (I) AS THE
            MEAN OF FIR.RESPONSE (I)
        LET TOT.RESPONSE = RESPONSE (1) ELSE
        ADD 1 TO NO.OF.QUERIES
        LET RESPONSE ( NO.OF.QUERIES ) = TIME.V - CONVERSATION.END
            ( QUERY.ADDR (K))
        ADD RESPONSE ( NO.OF.QUERIES ) TO TOT.RESPONSE
        'UP'    LET TM=NORMAL.F( TALK.TIME, SD.TALK,E)
        IF TM < 0.0 GO TO UP ELSE
        SCHEDULE AN AN.END (Q) AT TIME.V + TM
        IF BETWEEN.V = 0 JUMP AHEAD ELSE
        IF NO.OF.QUERIES > SHUT.OFF.TRACE AND DESIRE.PRINT = 0
        LET DESIRE.PRINT = 1 CALL SECOND.TRACE (DUM) LET BETWEEN.V = 0
        RELEASE TRACE.MATRIX
        ELSE
        HERE
        IF BETWEEN.V = 0 AND DISCIPLINE = 1 AND NO.OF.QUERIES = 2
            RELEASE TRACE, SECOND.TRACE ELSE
        '' WHEN LIMIT.QUERY = 0 THE SIMULATION WILL FIND ITS OWN STOPPING POINT
        IF QUERY.NUM ( QUERY.ADDR (K)) = LIMIT.QUERY
        IF GOOD = 0 PRINT 1 LINE THUS
      LAST QUERY BUT RUN IS STILL TRENDING
        ELSE
        GO TO DOWN ELSE
        IF NO.OF.QUERIES < 100 LET MEAN.RESPONSE = TOT.RESPONSE / NO.OF.QUERIES
            RETURN ELSE
        IF GOOD = 1 LET MEAN.RESPONSE =
            TOT.RESPONSE / ( NO.OF.QUERIES - TRANSIENT (DISCIPLINE))
        LET X= TRUNC.F ( NO.OF.QUERIES / 1000.0) LET Y= NO.OF.QUERIES / 1000.0
        IF X NE Y RETURN ELSE GO TO DOWN
        ELSE '' GOOD = 0
        LET X= TRUNC.F ( NO.OF.QUERIES  / 100.0 )
        LET Y= NO.OF.QUERIES / 100.0
        IF X NE Y
        LET MEAN.RESPONSE = TOT.RESPONSE / NO.OF.QUERIES
        RETURN
        ELSE
        CALL COX.STUART
        ''YOU CAN SET TRANSIENT PERIOD BEFORE SIMULATION IN MULTIPLES OF 100
        ''NEVERTHELESS YOU GET COX STUART TEST FOR EACH SERIES.
        IF NO.OF.QUERIES > OUTSIDE.TRANSIENT ( DISCIPLINE )
            LET TRANSIENT ( DISCIPLINE ) = OUTSIDE.TRANSIENT ( DISCIPLINE )
            IF COX.ON = 1
        PRINT  1 DOUBLE LINE WITH TRANSIENT ( DISCIPLINE ) AND DISCIPLINE THUS
FOR THIS SIMULATION THERE IS A TREND BUT THE TRANSIENT STATE *      * QUERIES   FOR DISCIPLINE  *
        JUMP AHEAD
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                          06/28/76
          ELSE ELSE IF COX.ON =1 GO TO MAKE.BIGGER ELSE  ''NO TREND
          SKIP 3 LINES PRINT 1 LINE WITH TRANSIENT (DISCIPLINE) AND
              DISCIPLINE THUS
FOR THIS SIMULATION THE TRANSIENT STATE =      * QUERIES FOR DISCIPLINE   *
      HERE
              '' TRANSIENT STATE SET OUTSIDE SIMULATION MUST BE IN MULTIPLES OF
              '' A HUNDRED
              '' RESET MEAN.RESPONSE
              FOR I=TRANSIENT ( DISCIPLINE ) - 19 TO TRANSIENT ( DISCIPLINE )
                     COMPUTE MEAN.RESPONSE AS THE MEAN OF RESPONSE (I)
                     LET TOT.RESPONSE = 0.0
          'RESET'
          RESET TOTALS OF RUN.INTERRUPTS AND RUN.BLOCKED AND N.TQ AND
              N.ONE.QU AND N.LOSS.QU AND N.PG AND N.LAPG
          FOR EACH CPU RESET TOTALS OF N.CQ (CPU)
          FOR EACH CPU RESET TOTALS OF STATUS (CPU)
          FOR EACH RAD RESET TOTALS OF N.RQ (RAD)
          FOR EACH RAD.CHANNEL RESET TOTALS OF N.RAD.CHANNEL.CU ( RAD.CHANNEL )
          FOR EACH RAD RESET TOTALS OF SEEK.TIME (RAD)
          FOR EACH RAD RESET TOTALS OF RAD.BUSY (RAD)
          FOR EACH RAD.CHANNEL RESET TOTALS OF BUSY.2 ( RAD.CHANNEL )
          IF SECOND.STAT = 1 GO TO RESET.RETURN ELSE
                     LET GOOD = 1 LET CHANGE.TO.GOOD = 1
          'MAKE.BIGGER'
          IF GOOD = 0 LET F = 100 JUMP AHEAD ELSE
          '' WE ARE PAST THE TRANSIENT PERIOD
          LET STACKS = TRUNC.F ( NO.OF.QUERIES / 1000.0) * 1000.0
          LET ONE.GREATER = STACKS + 1000.0
          LET F = ONE.GREATER - NO.OF.QUERIES
      HERE
          RESERVE H AS NO.OF.QUERIES
          FOR I=1 TO NO.OF.QUERIES LET H(I)= RESPONSE (I)
          RELEASE RESPONSE RESERVE RESPONSE AS NO.OF.QUERIES + F
          FOR I = 1 TO NO.OF.QUERIES LET RESPONSE (I) = H (I) RELEASE H
          RETURN
          'COWN'  ''TEST TO SEE IF THE SYSTEM WITH THE GIVEN DISCIPLINE IS IN
          ''EQUILIBRIUM USING THE MANN WHITNEY U TEST ON SAMPLE 1 AND SAMPLE 2
          ''THE TRANSIENT PERIOD IS FIRST REMOVED FROM CONSIDERATION THEN
          ''SAMPLE 1 BECOMES THE RUN ( THE FIRST HALF OF THE REMAINING RESPONSES)
          ''SAMPLE 2 BECOMES THE REPLICATION.
          LET LENGTH = TRUNC.F((NO.OF.QUERIES - TRANSIENT ( DISCIPLINE ))/2.0)
          RESERVE D1, D2 AS LENGTH
          FOR I= TRANSIENT (DISCIPLINE) +1 TO TRANSIENT (DISCIPLINE) + LENGTH
              LET D1(I - TRANSIENT (DISCIPLINE))= RESPONSE (I)
          FOR I = 1 TO LENGTH
              LET D2 (I)= RESPONSE ( TRANSIENT ( DISCIPLINE ) +I+LENGTH)
          LET Z.SCORE = MANN.WHITNEY (LENGTH, LENGTH) RELEASE D1, D2
          IF LOWER.LIMIT GE 2000 AND LOWER.LIMIT - NO.OF.QUERIES LE 1000
              AND SECOND.STAT = 0
                     CALL STAT
                     LET SECOND.STAT = 1
                     LET ARR.TEST = 0
                     FOR I= 1 TO 100 FOR J=1 TO N.RAD * 2 LET DIST.ARRIVAL (I,J)
```

LINE CACI SIMSCRIPT II.5 RELEASE 8D 06/28/76

```
                    = 0.0
                GO TO RESET ELSE
    'RESET.RETURN'
    IF LIMIT.QUERY = QUERY.NUM ( QUERY.ADDR (K)) GO TO DROP ELSE
    IF LIMIT.QUERY = 0 AND ABS.F (Z.SCORE) LE Z.CUT AND NO,OF,QUERIES >
                LOWER.LIMIT GO TO DROP ELSE
        RESERVE H AS NO.OF.QUERIES
        FOR I=1 TO NO.OF.QUERIES LET H(I) = RESPONSE (I)
        RELEASE RESPONSE RESERVE RESPONSE AS NO.OF.QUERIES + 1000
    FOR I=1 TO NO.OF.QUERIES LET RESPONSE (I) = H(I) RELEASE H JUMP AHEAD
  'DROP'
    LET CEASE.RUN = 1
    '' COMPRESS OUT EXTRA ZEROS
    RESERVE H AS NO.OF.QUERIES
    FOR I = 1 TO NO.OF.QUERIES LET H(I)= RESPONSE (I)
    RELEASE RESPONSE RESERVE RESPONSE AS NO.OF.QUERIES
    FOR I = 1 TO NO.OF.QUERIES LET RESPONSE (I) = H(I)
    RELEASE H
    HERE
    PRINT 1 DOUBLE LINE WITH NO.OF.QUERIES, TRANSIENT ( DISCIPLINE )
        AND DISCIPLINE, Z.SCORE AND QUERY.NUM ( QUERY.ADDR (K)) THUS
FOR    * TOTAL.QUERIES (      * TRANSIENT) OF DISCIPLINE    * Z= *.******    LAST QUERY NUMBER PROCESSED =      *
  RETURN END
```

LOCAL VARIABLES OF THIS ROUTINE

SUM	INTEGER	WORD 17	F	INTEGER	WORD 37
H	REAL	WORD 8	I	INTEGER	WORD 10
I.1	INTEGER	WORD 21	I.2	INTEGER	WORD 22
I.3	INTEGER	WORD 23	J	INTEGER	WORD 41
J.1	INTEGER	WORD 24	K	INTEGER	WORD 9
K.1	INTEGER	WORD 26	K.2	INTEGER	WORD 27
K.3	INTEGER	WORD 28	K.4	INTEGER	WORD 29
L.12	DOUBLE	WORD 31	L.13	DOUBLE	WORD 33
L.14	DOUBLE	WORD 35	L.4	DOUBLE	WORD 11
L.5	DOUBLE	WORD 13	L.6	DOUBLE	WORD 15
L.7	INTEGER	WORD 30	LENGTH	INTEGER	WORD 40
M1	REAL	WORD 5	M2	REAL	WORD 6
N.1	INTEGER	WORD 25	ONE.CREATE	INTEGER	WORD 39
Q	INTEGER	WORD 1	R.1	DOUBLE	WORD 19
STACKS	INTEGER	WORD 38	TH	REAL	WORD 7
X	REAL	WORD 3	Y	REAL	WORD 4

```
LINE  CACI SIMSCRIPT II.5  RELEASE 80                                06/28/76

      ROUTINE COX.STUART
              '' SEE BRADLEY, DISTRIBUTION FREE STATISTICAL TESTS P. 174 FF.
               ''TEST NEXT BLOCK OF 100 OBSERVATIONS FOR TREND.
              DEFINE BLOCK AS AN INTEGER VARIABLE
              DEFINE DIF AS A REAL VARIABLE
               LET BLOCK = 33
              FOR I=0 TO BLOCK - 1
              DO       LET DIF = RESPONSE ( NO.OF.QUERIES - I) - RESPONSE
                             ( NO.OF.QUERIES - I - (BLOCK * 2))
                  IF DIF > 0 ADD 1 TO POS ELSE
                  IF DIF IS NEGATIVE ADD 1 TO NEG ELSE
                  IF DIF = 0 ADD 1 TO TIE ELSE  ''TIES ARE THROWN OUT
                  LOOP
              LET S = MIN.F(NEG,POS)
              SKIP 1 LINE PRINT 1 LINE WITH POS+NEG+TIE, POS AND NEG AND TIE THUS
OUT OF       * OBSERVATIONS        * WERE POSITIVE     * WERE NEGATIVE      * TIES
               IF S GF 10 GO TO PRT ELSE
              PRINT 1 LINE WITH NO.OF.QUERIES THUS
WITH      * NO OF QUERIES THE COX STUART TEST IS SIGNIF. AT 0.05 LEVEL:  A TREND
              LET COX.ON = 1
              RETURN
              'PRT'
              PRINT 1 LINE THUS
COX.STUART TEST IS NOT SIGNIFICANT AT 0.05 LEVEL   NO DETECTABLE TREND
               '' EVEN IF NO TREND IS DETECTABLE WE DROP FIRST 100 OBSERVATIONS
               '' AS TRANSIENT
               LET TRANSIENT ( DISCIPLINE ) = NO.OF.QUERIES
              LET COX.ON = 0
              RETURN
              END
```

LOCAL VARIABLES OF THIS ROUTINE

BLOCK	INTEGER WORD 1	DIF	REAL	WORD 2	
I	INTEGER WORD 3	I.1	INTEGER	WORD 11	
I.2	INTEGER WORD 12	I.3	INTEGER	WORD 13	
J.1	INTEGER WORD 14	K.1	INTEGER	WORD 16	
K.2	INTEGER WORD 17	K.3	INTEGER	WORD 18	
K.4	INTEGER WORD 19	L.4	INTEGER	WORD 8	
N.1	INTEGER WORD 15	NEG	INTEGER	WORD 5	
POS	INTEGER WORD 4	R.1	DOUBLE	WORD 9	
S	INTEGER WORD 7	TIE	INTEGER	WORD 6	

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                    06/28/76

   EVENT AN.END (P)
          LET AVAILABILITY ( CRT.NUMBER (P)) = 1
          SCHEDULE A TELEPHONE.CALL NOW
          SCHEDULE AN ARRIVAL.AT.CRT ( CRT.NUMBER (P)) NOW
          DESTROY THE QUERY CALLED QUERY.ADDR ( CRT.NUMBER (P))
          DESTROY THE VIRTUAL.ROUTINE CALLED P
          IF CEASE.RUN = 1 CALL TERMINATE LET CEASE.RUN = 0 ELSE
          RETURN END

            LOCAL VARIABLES OF THIS ROUTINE

 P          INTEGER  WORD  1
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 80                                   06/28/76

ROUTINE NORMAL
          FOR I=1 TO DIM.F ( FIR.RESPONSE (*)) COMPUTE M3 = MEAN, S3 = STD OF
              FIR.RESPONSE (I)
          RESERVE D1 AS DIM.F ( FIR.RESPONSE (*)) RESERVE D2 AS DIM.F (D1(*))
          LET G = DIM.F (D1(*))
          FOR I= 1 TO DIM.F (D1(*)) DO LET D1(I) = FIR.RESPONSE (I)
              LET D2(I) = NORMAL.F (M3,S3,E)
              LOOP
          LET Z.SCORE = ABS.F ( MANN.WHITNEY ( DIM.F (D1(*)), DIM.F (D2(*))))
          PRINT 1 LINE WITH Z.SCORE THUS
FOR DISCIPLINE 1 THE Z.SCORE =  *.**** ON THE PROBABILITY THAT DIF. IS BY CHANCE
          IF Z.SCORE > Z.CUT  '' NOT NORMAL
          PRINT 1 LINE THUS
     RESPONSE TIMES ARE NOT NORMAL
          JUMP AHEAD ELSE
          PRINT 1 DOUBLE LINE THUS
     RESPONSE TIMES FOR THIS DISCIPLINE DO NOT SIGNIF. DIFFER FROM NORMAL DISTRIBUTION UNDER THE NULL HYPOTHESIS
          HERE
          '' TEST OF DISCIPLINE 1 FOR EXPONENTIAL DISTRIBUTION
          FOR I = 1 TO DIM.F(D1(*)) LET D2(I)= EXPONENTIAL.F (M3,E)
          LET Z.SCORE = ABS.F ( MANN.WHITNEY (G,G))
          PRINT 1 LINE WITH Z.SCORE THUS
FOR DISCIPLINE 1 Z.SCORE =    *.**** FOR EXPONENTIAL TEST
          LET DISCIPLINE  = 1
          START NEW PAGE CALL SORT.D1 (D1(*)) CALL HISTOGRAM
          RELEASE D1, D2
          '' TEST OF NORMALITY FOR DISCIPLINE 2  (RESPONSE TIMES)
          FOR I=1 TO DIM.F ( SEC.RESPONSE (*)) COMPUTE M3 = MEAN, S3 = STD OF
              SEC.RESPONSE (I)
          RESERVE D1 AS DIM.F ( SEC.RESPONSE (*)) RESERVE D2 AS DIM.F (D1(*))
          LET G = DIM.F (D1(*))
          FOR I= 1 TO DIM.F (D1(*)) DO LET D1(I) = SEC.RESPONSE (I)
              LET D2(I) = NORMAL.F (M3,S3,E)
              LOOP
          LET Z.SCORE = ABS.F ( MANN.WHITNEY ( DIM.F (D1(*)), DIM.F (D2(*))))
          PRINT 1 LINE WITH Z.SCORE THUS
FOR DISCIPLINE 2 THE Z.SCORE =  *.**** ON THE PROBABILITY THAT DIF. IS BY CHANCE
          SKIP 1 LINE
          IF Z.SCORE > Z.CUT  '' NOT NORMAL
          PRINT 1 LINE THUS
     RESPONSE TIMES ARE NOT NORMAL
          JUMP AHEAD ELSE
          PRINT 1 DOUBLE LINE THUS
     RESPONSE TIMES FOR THIS DISCIPLINE DO NOT SIGNIF. DIFFER FROM NORMAL DISTRIBUTION UNDER THE NULL HYPOTHESIS
          HERE
          '' TEST OF DISCIPLINE 2 FOR EXPONENTIAL DISTRIBUTION
          FOR I = 1 TO DIM.F(D1(*)) LET D2(I)= EXPONENTIAL.F (M3,E)
          LET Z.SCORE = ABS.F ( MANN.WHITNEY (G,G))
          PRINT 1 LINE WITH Z.SCORE THUS
FOR DISCIPLINE 2 Z.SCORE =    *.**** FOR EXPONENTIAL TEST
          LET DISCIPLINE = 2
          START NEW PAGE CALL SORT.D1 (D1(*)) CALL HISTOGRAM
          RELEASE D1, D2
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                06/28/76
            '' TEST OF NORMALITY FOR DISCIPLINE 3  (RESPONSE TIMES)
            FOR I = TRANSIENT (3)+1 TO DIM.F ( RESPONSE (*))
                COMPUTE M3 = MEAN, S3 = STD OF RESPONSE (I)
            RESERVE D1 AS DIM.F ( RESPONSE (*)) - TRANSIENT (3)
            LET G = DIM.F (D1(*))
            RESERVE D2 AS DIM.F (D1(*))
            FOR I = 1 TO DIM.F (D1(*))  DO
                LET D1(I) = RESPONSE ( I + TRANSIENT (3))
                LET D2(I) = NORMAL.F (M3,S3,E)
                LOOP
            LET Z.SCORE = ABS.F ( MANN.WHITNEY ( DIM.F (D1(*)), DIM.F (D2(*))))
            PRINT 1 LINE WITH Z.SCORE THUS
FOR DISCIPLINE 3 THE Z.SCORE =  *.**** ON THE PROBABILITY THAT DIF. IS BY CHANCE
            SKIP 1 LINE
            IF Z.SCORE > Z.CUT  '' NOT NORMAL
            PRINT 1 LINE THUS
      RESPONSE TIMES ARE NOT NORMAL
            JUMP AHEAD ELSE
            PRINT 1 DOUBLE LINE THUS
      RESPONSE TIMES FOR THIS DISCIPLINE DO NOT SIGNIF. DIFFER FROM NORMAL DISTRIBUTION UNDER THE NULL HYPOTHESIS
            HERE
            '' TEST OF DISCIPLINE 3 FOR EXPONENTIAL DISTRIBUTION
            FOR I = 1 TO DIM.F(D1(*)) LET D2(I)= EXPONENTIAL.F (M3,E)
            LET Z.SCORE = ABS.F ( MANN.WHITNEY (G,G))
            PRINT 1 LINE WITH Z.SCORE THUS
FOR DISCIPLINE 3 Z.SCORE =    *.**** FOR EXPONENTIAL TEST
            START NEW PAGE CALL SORT.D1 (D1(*)) CALL HISTOGRAM
            LET DISCIPLINE = 3
            RELEASE D1, D2
            '' TEST OF NORMALITY BETWEEN 2 KNOWN NORMAL DISTRIBUTIONS
                RESERVE D1 AS DIM.F ( FIR.RESPONSE (*))  RESERVE D2 AS DIM.F(D1(*))
            FOR I = 1 TO DIM.F ( FIR.RESPONSE (*)) COMPUTE M3 = MEAN, S3 = STD
                OF FIR.RESPONSE (I)
            FOR I= 1 TO DIM.F (D1(*)) DO LET D1(I) = NORMAL.F (M3,S3,E)
                               LET D2(I) = NORMAL.F (M3,S3,E)
                LOOP
            LET Z.SCORE = ABS.F ( MANN.WHITNEY ( DIM.F (D1(*)), DIM.F (D2(*))))
            PRINT 1 LINE WITH Z.SCORE THUS
FOR 2 NORMALS   THE Z.SCORE =  *.**** ON THE PROBABILITY THAT DIF. IS BY CHANCE
            IF Z.SCORE > Z.CUT  '' NOT NORMAL
            PRINT 1 LINE THUS
      RESPONSE TIMES ARE NOT NORMAL  FOR TWO NORMAL DISTRIBUTIONS
            JUMP AHEAD ELSE
            PRINT 1 DOUBLE LINE THUS
RESPONSE TIMES FOR NORMALS DO NOT SIGNIFICANTLY DIFFER FROM NORMAL DISTRIBUTION UNDER THE NULL HYPOTHESIS
            HERE
            RELEASE D1, D2
      RETURN END
```

LOCAL VARIABLES OF THIS ROUTINE

G	INTEGER	WORD 12	I	INTEGER	WORD 1
I.1	INTEGER	WORD 15	I.2	INTEGER	WORD 16
I.3	INTEGER	WORD 17	J.1	INTEGER	WORD 18
K.1	INTEGER	WORD 20	K.2	INTEGER	WORD 21
K.3	INTEGER	WORD 22	K.4	INTEGER	WORD 23
L.17	DOUBLE	WORD 25	L.18	DOUBLE	WORD 27
L.19	DOUBLE	WORD 29	L.20	DOUBLE	WORD 31
L.24	INTEGER	WORD 33	L.32	DOUBLE	WORD 35
L.33	DOUBLE	WORD 37	L.34	DOUBLE	WORD 39
L.35	DOUBLE	WORD 41	L.39	INTEGER	WORD 43
L.4	DOUBLE	WORD 3	L.47	DOUBLE	WORD 45
L.48	DOUBLE	WORD 47	L.49	DOUBLE	WORD 49
L.5	DOUBLE	WORD 5	L.50	DOUBLE	WORD 51
L.6	DOUBLE	WORD 7	L.7	DOUBLE	WORD 9
N.1	INTEGER	WORD 19	R.1	DOUBLE	WORD 13
S3	INTEGER	WORD 11			

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                        06/28/76

ROUTINE AUTO.CORRELATION
        NORMALLY MODE IS REAL
        DEFINE OBSERV.BLOCKS AS A 1-DIM ARRAY
        DEFINE I,J,K,N, INDEX AS INTEGER VARIABLES
        SKIP 1 LINE
        LET N=DIM.F (D1(*))
FOR I=1 TO N WITH D1(I) = 0.0 FIND THE FIRST CASE IF FOUND PRINT 1 LINE WITH I
THUS
D1 HAS A ZERO AT INDEX     *
ELSE
        FOR K=1 TO TRUNC.F (N/20.0)
            DO
        LET FRONT = 1.0 / (N-K)
        LET NN=N LET SQ.FRONT = 1.0/((NN-K)**2)
        FOR J=1 TO N-K COMPUTE CROSS = SUM OF D1(J) * D1(J+K)
        FOR J=1 TO N-K COMPUTE FIRST=SUM OF D1(J)
        FOR J=1 TO N-K COMPUTE SECOND = SUM OF D1(J+K)
        FOR J=1 TO N-K COMPUTE FIR.SQ=SUM OF D1(J)**2
        FOR J=1 TO N-K COMPUTE SEC.SQ=SUM OF D1(J+K) **2
        LET NUM=(FRONT * CROSS) -(SQ.FRONT*FIRST*SECOND)
        LET DENOM=SQRT.F( (FRONT * FIR.SQ)-(SQ.FRONT * (FIRST **2))) *
             SQRT.F( (FRONT * SEC.SQ) - ( SQ.FRONT * (SECOND **2)))
        LET R = NUM / DENOM
        IF K = 1 LET LOW = R LET INDEX = 1 JUMP  AHEAD ELSE
        IF R<LOW LET LOW = R LET INDEX = K ELSE
      HERE
        LET Z= ABS.F ( RTEST (R,N))
        PRINT 1 LINE WITH DISCIPLINE, K, R, Z THUS
FOR DISCIPLINE   * UNDER A LAG OF    . * FOR AN R OF *.****  Z=    *.****
        IF Z < 1.96 OR Z=1.96 PRINT 1 LINE THUS
  NO SIGNIFICANT AUTOCORRELATION
        GO TO BLOCKS ELSE
            LOOP
        PRINT 1 LINE THUS
------ WE WILL CHOOSE THE LAG WITH THE LOWEST AUTOCORRELATION ------
        LET K = INDEX
      'BLOCKS'
        RESERVE OBSERV.BLOCKS AS TRUNC.F  (N/K)
        FOR I=0 TO TRUNC.F(N/K) -1
            DO FOR J=I*K+1 TO I*K+K WITH D1(J) NE 0.0 COMPUTE OBSERV.BLOCKS
            (I+1)= MEAN OF D1(J)
            LOOP
        FOR I=1 TO TRUNC.F(N/K) COMPUTE VARN AS THE VAR, M1 AS THE MEAN OF
            OBSERV.BLOCKS (I)
        PRINT 1 LINE WITH M1, SQRT.F (VARN), VARN THUS
MEAN OF BLOCKS =    *.*** ST. DEV =     *.** VAR =      *.******
        SKIP 1 LINE
        RELEASE OBSERV.BLOCKS
RETURN END
```

LINE CACI SIMSCRIPT II.5 RELEASE 8D 06/28/76

LOCAL VARIABLES OF THIS ROUTINE

CROSS	REAL	WORD 29	DENOM	REAL	WORD 63
FIR.SQ	REAL	WORD 53	FIRST	REAL	WORD 37
FRONT	REAL	WORD 20	I	INTEGER	WORD 2
I.1	INTEGER	WORD 11	I.2	INTEGER	WORD 12
I.3	INTEGER	WORD 13	INDEX	INTEGER	WORD 6
J	INTEGER	WORD 3	J.1	INTEGER	WORD 14
K	INTEGER	WORD 4	K.1	INTEGER	WORD 16
K.2	INTEGER	WORD 17	K.3	INTEGER	WORD 18
K.4	INTEGER	WORD 19	L.1	INTEGER	WORD 7
L.12	DOUBLE	WORD 23	L.13	DOUBLE	WORD 25
L.14	DOUBLE	WORD 27	L.18	DOUBLE	WORD 31
L.19	DOUBLE	WORD 33	L.20	DOUBLE	WORD 35
L.24	DOUBLE	WORD 39	L.25	DOUBLE	WORD 41
L.26	DOUBLE	WORD 43	L.30	DOUBLE	WORD 47
L.31	DOUBLE	WORD 49	L.32	DOUBLE	WORD 51
L.35	DOUBLE	WORD 55	L.37	DOUBLE	WORD 57
L.36	DOUBLE	WORD 59	L.45	DOUBLE	WORD 67
L.45	DOUBLE	WORD 69	L.47	DOUBLE	WORD 71
L.51	DOUBLE	WORD 73	L.52	DOUBLE	WORD 75
L.53	DOUBLE	WORD 77	L.54	DOUBLE	WORD 79
L.55	INTEGER	WORD 83	LOW	REAL	WORD 65
M1	REAL	WORD 81	N	INTEGER	WORD 5
N.1	INTEGER	WORD 15	NN	REAL	WORD 21
NUM	REAL	WORD 62	OBSERV.BLC	REAL	WORD 1
R	REAL	WORD 64	R.1	DOUBLE	WORD 9
SEC.SQ	REAL	WORD 61	SECOND	REAL	WORD 45
SQ.FRONT	REAL	WORD 22	VARN	REAL	WORD 82
Z	REAL	WORD 66			

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                    06/28/76

ROUTINE RTEST (R,N)       '' A T-TEST ON NULL HYPOTHESIS THAT R = 0.0
        NORMALLY MODE IS REAL
        DEFINE N AS AN INTEGER VARIABLE
        '' SEE SASAKI, STATISTICS FOR MODERN BUSINESS DECISION MAKING P. 433
        LET SB=SQRT.F ((1.0-(R**2))/(N-2.0))
        LET T=R/SB
        RETURN WITH T
END

          LOCAL VARIABLES OF THIS ROUTINE

N         INTEGER  WORD  3     R           REAL     WORD  1
SB        REAL     WORD  5     T           REAL     WORD  6
```

```
ROUTINE SORT.D1 (D1)
NORMALLY MODE IS REAL
DEFINE D1 AS A 1-DIM REAL ARRAY
          '' THIS ROUTINE SORTS A GIVEN VECTOR CALLED D1
          DEFINE BUCKETS AS A 1-DIM ARRAY
          DEFINE FREQUENCY, PROB AS A 1-DIM ARRAY
          DEFINE BOX, I, J, INDEX, BUBBLES AS INTEGER VARIABLES
          DEFINE DIMENSION, HIGH, LOW, REFINE, INTERVAL, HOLD,
               BOT, TOP, PERCENT, TEMP, BUMP AS REAL VARIABLES
          DEFINE K AS AN INTEGER VARIABLE
          LET REFINE = 50.0
          RESERVE FRECUENCY, PROB AS REFINE
          LET DIMENSION = DIM.F (D1(*))
          '' SEE IF VECTOR D1 NEEDS TO BE SORTED
          FOR I= 2 TO DIMENSION
               DO IF D1(I) < D1(I-1) GO TO SORT ELSE
               LOOP RETURN
          '' CALCULATE FREQUENCY DISTRIBUTION
 'SORT'   LET HIGH = - RINF.C  LET LOW = RINF.C
          FOR I=1 TO DIMENSION
               DO IF D1(I) > HIGH LET HIGH = D1(I) ELSE
               IF D1(I) < LOW LET LOW = D1(I) ELSE
               LOOP
          LET INTERVAL = (HIGH - LOW)/ REFINE
          FOR I= 1 TO DIMENSION
               DO LET BOX = TRUNC.F((D1(I) - LOW)/ INTERVAL) +1
                       IF BOX > REFINE LET BOX = REFINE ELSE
                       LET FREQUENCY (BOX) = FREQUENCY (BOX) +1
               LOOP
          '' CALCULATE CUMULATIVE PROBABILITY DISTRIBUTION FROM FREQUENCY DISTRIB.
          LET PROB (1) = FREQUENCY (1) / DIMENSION
          FOR I = 2 TO REFINE LET PROB (I)=PROB(I-1)+(FREQUENCY(I)/DIMENSION)
          LET PROB (REFINE) = 1.0  '' TO OFFSET ROUNDING ERRORS
          '' MAP VALUES TO BE SORTED INTO BUCKETS
          RESERVE BUCKETS AS DIMENSION * 2
          LET TEMP = RINF.C
          FOR I= 1 TO DIM.F (BUCKETS (*)) LET BUCKETS (I) = TEMP
          FOR I=1 TO DIMENSION
               DO LET INDEX = TRUNC.F ((D1(I) - LOW)/INTERVAL) +1
                       IF INDEX > REFINE LET INDEX = REFINE ELSE
                       LET BOT = LOW + (INDEX * INTERVAL) - INTERVAL
                       LET TOP = BOT + INTERVAL
                       LET PERCENT = (D1(I) - BOT) / INTERVAL
                       IF INDEX NE 1 LET BOX = (PROB(INDEX - 1) + PERCENT *
                              (PROB(INDEX) - PROB (INDEX -1))) * DIM.F (BUCKETS(*))
                       GO TO NEXT ELSE
                       IF INDEX = 1 AND PERCENT > 0.0
                            LET BOX = PERCENT * PROB (1) * DIM.F (BUCKETS (*))
                       GO TO NEXT
                       ELSE LET BOX = 1
           'NEXT'
                       IF BOX = 0 LET BOX = 1 ELSE
                       IF BOX > DIM.F(BUCKETS(*)) LET BOX = DIM.F (BUCKETS(*)) ELSE
```

```
                LET HOLD = DI(I)
'AGAIN'         IF BUCKETS (BOX) = TEMP LET BUCKETS (BOX) = HOLD CYCLE ELSE
                IF HOLD < BUCKETS (BOX) LET BUMP = BUCKETS (BOX)
                      LET BUCKETS (BOX) = HOLD
                      LET HOLD = BUMP ELSE
                IF BOX NE DIM.F (BUCKETS (*)) LET BOX = BOX + 1
                GO TO AGAIN ELSE     ''OVERFLOW
                CREATE AN OVERFLOW
                LET VALUE (OVERFLOW) = HOLD
                FILE OVERFLOW IN OVFL.AREA
        LOOP
'' PUT BUCKETS AND OVERFLOW QUEUE BACK INTO DI
LET J=1
FOR I=1 TO DIM.F (BUCKETS (*))
    DO IF BUCKETS (I) = TEMP CYCLE ELSE
           LET DI(J) = BUCKETS (I)
           LET J=J+1
    LOOP
IF OVFL.AREA IS NOT EMPTY
LET K = N.OVFL.AREA
FOR I= 1 TO K
           DO REMOVE FIRST OVERFLOW FROM OVFL.AREA
                 LET DI(J) = VALUE (OVERFLOW)
                 LET J=J+1
                 DESTROY THE OVERFLOW
           LOOP
ELSE
''TEST GOODNESS OF SORT
FOR I= 2 TO DIMENSION
    DO IF DI(I) < DI(I-1) GO TO DOWN ELSE
    LOOP
GO TO MEMORY
'DOWN'
'' BUBBLE SORT IF NECESSARY
FOR K = 1 TO DIMENSION
    DO FOR I = 1 TO DIMENSION - K WITH DI(I) > DI(I+1) FIND J =
            THE FIRST I
            IF NONE JUMP AHEAD ELSE
            FOR I = J TO DIMENSION - K WITH DI(I)> DI(I+1)
                  DO LET HOLD = DI(I)
                  LET DI(I) = DI(I+1)
                  LET DI(I+1) = HOLD
                  ADD 1 TO BUBBLES
            LOOP
    LOOP
HERE
LIST BUBBLES
'' LAST CHECK ON SORT
FOR I = 2 TO DIMENSION
    DO IF DI(I) < DI(I-1)
    PRINT 1 LINE WITH I, BUBBLES THUS
    RUN ABORTED SORT ROUTINE IN ERROR AT      * BUBBLES*          *
PRINT 1 LINE WITH I, I - 1 THUS
```

LINE CACI SIMSCRIPT II.5 RELEASE 8D 06/28/76

```
THE VECTOR IS NOT SORTED  I =     *    I-1 =      *
                        STOP
                  ELSE
                  LOOP
              'MEMORY'
              RELEASE FREQUENCY, PROB, BUCKETS
              RETURN END
```

LOCAL VARIABLES OF THIS ROUTINE

BOT	REAL	WORD	17		BOX	INTEGER	WORD	6
BUBBLES	INTEGER	WORD	10		BUCKETS	REAL	WORD	3
BUMP	REAL	WORD	21		DIMENSION	REAL	WORD	11
D1	REAL	WORD	1		FREQUENCY	REAL	WORD	4
HIGH	REAL	WORD	12		HOLD	REAL	WORD	16
I	INTEGER	WORD	7		I.1	INTEGER	WORD	25
I.2	INTEGER	WORD	26		I.3	INTEGER	WORD	27
INDEX	INTEGER	WORD	9		INTERVAL	REAL	WORD	15
J	INTEGER	WORD	8		J.1	INTEGER	WORD	28
K	INTEGER	WORD	22		K.1	INTEGER	WORD	30
K.2	INTEGER	WORD	31		K.3	INTEGER	WORD	32
K.4	INTEGER	WORD	33		LOW	REAL	WORD	13
N.1	INTEGER	WORD	29		PERCENT	REAL	WORD	19
PROB	REAL	WORD	5		R.1	DOUBLE	WORD	23
REFINE	REAL	WORD	14		TEMP	REAL	WORD	20
TOP	REAL	WORD	18					

```
ROUTINE MANN.WHITNEY (L1,L2)
        NORMALLY MODE IS REAL
        DEFINE E, U, VAR, Z, HOLD AS REAL VARIABLES
        DEFINE I, J, K, RANK, L1, L2, SWITCH, TOT AS INTEGER VARIABLES
                LIST L1, L2
        CALL SORT.D1 (D1(*))
        CALL SORT.D1 (D2(*))
        ''CHECK FOR CORRECTNESS OF DATA
        FOR I=1 TO L1 WITH D1(I)=0.0                    FIND THE FIRST CASE
            IF FOUND PRINT 1 LINE WITH D1(I), I THUS
      ------     A       *.**** IS IN THE D1 VECTOR AT INDEX    *
        ELSE FOR I=1 TO L2 WITH D2(I)=0.0                        FIND THE FIRST CASE
            IF FOUND PRINT 1 LINE WITH D2(I) AND I THUS
      ------     A       *.**** IS IN THE D2 VECTOR AT      *    ------
        ELSE
            '' CALCULATE STATISTICS AND PRINT THEM OUT FOR EACH VECTOR
        FOR I=1 TO L1 COMPUTE AV.FIR = MEAN, ST.FIR = STD OF D1(I)
        FOR I=1 TO L2 COMPUTE AV.SEC = MEAN, ST.SEC = STD OF D2(I)
        SKIP 1 LINE  PRINT 7 LINES WITH D1(L1), D2(L2), D1(1), D2(1),
            AV.FIR, AV.SEC, ST.FIR, ST.SEC, D1(L1/2), D2(L2/2) THUS
   MANN WHITNEY TEST  -------------------------------
              D1                                         D2
        MAX=     *.****                       MAX=     *.****
        MIN=     *.****                       MIN=     *.****
        AVE=     *.****                       AVE=     *.****
        STD=     *.****                       STD=     *.****
     MEDIAN=     *.****                    MEDIAN=     *.****
        SKIP 1 LINE
 LET RANK=0
 LET K=1
 FOR I=1 TO L1
     DO FOR J=K TO L2
          DO IF D2(J) < D1(I)
               ADD 1 TO RANK CYCLE ELSE
               ''CHECK FOR A SINGLE TIE
                    '' USING REAL FLOATING POINT PRECISION THE PROBABILITY OF
                    '' MULTIPLE TIES APPROACHES ZERO
               IF D2(J)=D1(I)
               ADD (RANK+1+RANK+2)/2 TO TOT
                    ADD 2 TO RANK ADD 1 TO J LET SWITCH = 1    LEAVE ELSE
               ADD 1 TO RANK
               ADD RANK TO TOT LEAVE
          LOOP
          IF J<L2+1 LET K=J ELSE
             IF J = L2+1 AND SWITCH = 1 LET SWITCH = 0
             LET K= L2+1 CYCLE ELSE
             LET SWITCH = 0
             IF J=L2+1 ADD 1 TO RANK ADD RANK TO TOT
             LET K=L2+1 ELSE
      LOOP
  LET U=L1*L2+(L1*(L1+1))/2.0 - TOT
  LET E=L1*L2/2.0
  LET VAR=(L1*L2*(L1+L2+1.0))/12.0
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                06/28/76

  45   LET Z=(U-E)/SQRT.F(VAR)
  46   RETURN WITH Z END


                 LOCAL VARIABLES OF THIS ROUTINE

 AV.FIR      REAL       WORD 37      AV.SEC      REAL       WORD 47
 E           REAL       WORD  5      HOLD        REAL       WORD  9
 I           INTEGER    WORD 10      I.1         INTEGER    WORD 19
 I.2         INTEGER    WORD 20      I.3         INTEGER    WORD 21
 J           INTEGER    WORD 11      J.1         INTEGER    WORD 22
 K           INTEGER    WORD 12      K.1         INTEGER    WORD 24
 K.2         INTEGER    WORD 25      K.3         INTEGER    WORD 26
 K.4         INTEGER    WORD 27      L.10        DOUBLE     WORD 29
 L.11        DOUBLE     WORD 31      L.12        DOUBLE     WORD 33
 L.13        DOUBLE     WORD 35      L.17        DOUBLE     WORD 39
 L.15        DOUBLE     WORD 41      L.19        DOUBLE     WORD 43
 L.20        DOUBLE     WORD 45      L.21        INTEGER    WORD 49
 L.23        INTEGER    WORD 50      L1          INTEGER    WORD  1
 L2          INTEGER    WORD  3      N.1         INTEGER    WORD 23
 R.1         DOUBLE     WORD 17      RANK        INTEGER    WORD 13
 ST.FIR      REAL       WORD 38      ST.SEC      REAL       WORD 48
 SWITCH      INTEGER    WORD 14      TOT         INTEGER    WORD 15
 U           REAL       WORD  6      VAR         REAL       WORD  7
 Z           REAL       WORD  8
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 80                                       06/28/76

    1  ROUTINE DISC.STAT
    2          NORMALLY MODE IS REAL
    3          DEFINE L1, I AS INTEGER VARIABLES
    4          LET L1 = DIM.F (D1(*))
    5          START NEW PAGE PRINT 1 LINE WITH DISCIPLINE THUS
FOR DISCIPLINE  *  ------ FOR RESPONSE TIMES  --------------
    6          CALL SORT.D1 (D1(*)) CALL HISTOGRAM
    7          START NEW PAGE PRINT 1 LINE WITH DISCIPLINE THUS
FOR DISCIPLINE  *     ------- FOR LOSS FUNCTION -------------
    8          FOR I=1 TO L1 DO LET D=D1(I) LET D1(I)= LOSS.FUNCTION (D) LOOP
    9          CALL SORT.D1 (D1(*)) CALL HISTOGRAM
   10          RETURN END

             LOCAL VARIABLES OF THIS ROUTINE

 D          REAL      WORD 14      I          INTEGER  WORD  2
 I.1        INTEGER   WORD  5      I.2        INTEGER  WORD  6
 I.3        INTEGER   WORD  7      J.1        INTEGER  WORD  8
 K.1        INTEGER   WORD 10      K.2        INTEGER  WORD 11
 K.3        INTEGER   WORD 12      K.4        INTEGER  WORD 13
 L1         INTEGER   WORD  1      N.1        INTEGER  WORD  9
 R.1        DOUBLE    WORD  3
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                        06/28/76

ROUTINE HISTOGRAM
        ''ASSUMES A GIVEN VECTOR D1 IS ALREADY SORTED
        NORMALLY MODE IS REAL
        DEFINE BIG AS A REAL VARIABLE
        DEFINE TABLE AND L AS A 1-DIM INTEGER ARRAY
        DEFINE I,J, N AS INTEGER VARIABLES
        RESERVE TABLE AND L AS 100
        FOR I= 1 TO 100 LET L(I) = 1
        LET N=DIM.F(D1(*))
        LET RANGE = D1(N) - D1(1)
        LET INCREMENT = RANGE / 100.0 LET HIGH = D1(1) + INCREMENT
        LET J = 1
        FOR I = 1 TO N DO
        HERE IF D1(I) > HIGH AND J < 100
            ADD 1 TO J ADD INCREMENT TO HIGH JUMP BACK ELSE
            ADD 1 TO TABLE (J)
        LOOP
        '' SCALE TABLE
        FOR I=1 TO 100 COMPUTE LARGE=MAX OF TABLE(I)
        LET POINT = LARGE / 50.0
        FOR I=1 TO N COMPUTE AVE = MEAN, SMALL = MIN, BIG = MAX OF D1(I)
        LET MEDIAN = D1(N/2)
        PRINT 1 LINE WITH AVE, MEDIAN, SMALL, BIG THUS
THE AVE =    *.*** THE MEDIAN =       *.*** MIN=        *.*** MAX=         *.***
        SKIP 1 LINE     '' PRINT HISTOGRAM
        FOR I BACK FROM 50 TO 1
        DO FOR J=1 TO 100 WITH TABLE (J) GE POINT *(I-1)
            UNLESS TABLE (J) = 0
            LET L(J) = 8
        PRINT 1 DOUBLE LINE WITH L(1), L(2), L(3), L(4), L(5), L(6), L(7),
            L(8), L(9), L(10), L(11), L(12), L(13), L(14), L(15), L(16),  L(17),
          L(18), L(19), L(20), L(21), L(22), L(23), L(24), L(25), L(26),  L(27),
          L(28), L(29), L(30), L(31), L(32), L(33), L(34), L(35), L(36),  L(37),
          L(38), L(39), L(40), L(41), L(42), L(43), L(44), L(45), L(46),  L(47),
          L(48), L(49), L(50), L(51), L(52), L(53), L(54), L(55), L(56),  L(57),
          L(58), L(59), L(60), L(61), L(62), L(63), L(64), L(65), L(66),  L(67),
          L(68), L(69), L(70), L(71), L(72), L(73), L(74), L(75), L(76),  L(77),
          L(78), L(79), L(80), L(81), L(82), L(83), L(84), L(85), L(86),  L(87),
          L(88), L(89), L(90), L(91), L(92), L(93), L(94), L(95), L(96),  L(97),
          L(98), L(99), L(100) THUS
    ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
        LOOP
        RELEASE TABLE, L
        RETURN END
```

LINE CACI SIMSCRIPT II.5 RELEASE 8D 06/28/76

LOCAL VARIABLES OF THIS ROUTINE

AVE	REAL	WORD 29	BIG	REAL	WORD 1
HIGH	REAL	WORD 9	I	INTEGER	WORD 4
I.1	INTEGER	WORD 35	I.2	INTEGER	WORD 36
I.3	INTEGER	WORD 37	INCREMENT	REAL	WORD 8
J	INTEGER	WORD 5	J.1	INTEGER	WORD 38
K.1	INTEGER	WORD 40	K.2	INTEGER	WORD 41
K.3	INTEGER	WORD 42	K.4	INTEGER	WORD 43
L	INTEGER	WORD 3	L.10	DOUBLE	WORD 11
L.11	DOUBLE	WORD 13	L.12	DOUBLE	WORD 15
L.16	DOUBLE	WORD 19	L.17	DOUBLE	WORD 21
L.18	DOUBLE	WORD 23	L.19	DOUBLE	WORD 25
L.20	DOUBLE	WORD 27	L.21	INTEGER	WORD 44
LARGE	REAL	WORD 17	MEDIAN	REAL	WORD 31
N	INTEGER	WORD 6	N.1	INTEGER	WORD 39
POINT	REAL	WORD 18	R.1	DOUBLE	WORD 33
RANGE	REAL	WORD 7	SMALL	REAL	WORD 30
TABLE	INTEGER	WORD 2			

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                   06/28/76

  ROUTINE LOSS.FUNCTION (D)
          NORMALLY MODE IS REAL
          IF D > 170 LET D = 170 ELSE
          LET EXP = ABS.F ( -CCC*D)
          IF EXP = 0.0 PRINT 1 LINE THUS
---------- D IN LOSS FUNCTION = 0   AN ERROR  ----------
          LET D=0.0 RETURN WITH D ELSE
          LET EXP = BBB ** EXP
          LET D = AAA/(AAA+(1.0/EXP))
          RETURN WITH D END

            LOCAL VARIABLES OF THIS ROUTINE

D           REAL     WORD  1      EXP        REAL     WORD  3
I.1         INTEGER  WORD  7      I.2        INTEGER  WORD  8
I.3         INTEGER  WORD  9      J.1        INTEGER  WORD 10
K.1         INTEGER  WORD 12      K.2        INTEGER  WORD 13
K.3         INTEGER  WORD 14      K.4        INTEGER  WORD 15
N.1         INTEGER  WORD 11      R.1        DOUBLE   WORD  5
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                              06/28/76

  ROUTINE ARRIV.SERV (D,SWITCH, TIME)
          NORMALLY MODE IS REAL
          DEFINE NU.DISTR AS AN INTEGER VARIABLE
          DEFINE TYPE AND MEASURE AS ALPHA VARIABLES
          DEFINE I,J,D,SIZE,SWITCH,L AS INTEGER VARIABLES
          DEFINE BIG AS A REAL VARIABLE
          IF ARRIVAL,SERVICE NE "YES" LET ARR.TEST = 1 RETURN ELSE
          IF SHUNT = 1 RETURN ELSE
          LET L = N.RAD * 2
          IF SWITCH = 2 LET D=D + N.RAD ELSE
          FOR I=1 TO 100 WITH DIST.ARRIVAL (I,D)=0.0 FIND THE FIRST CASE
          IF FOUND IF SWITCH = 1 LET DIST.ARRIVAL (I,D)= TIME.V ELSE
              IF SWITCH = 2 LET DIST.ARRIVAL (I,D) = TIME ELSE
          ELSE
          FOR J=1 TO L FOR I=1 TO 100 WITH DIST.ARRIVAL (I,J)=0.0 FIND THE FIRST
              CASE IF FOUND RETURN ELSE
          LET ARR.TEST = 1
          '' TABLE IS FULL.   NOW TEST FOR POISSON ARRIVALS
           '' FIRST CALCULATE MEAN INTERARRIVAL RATES AND SERVICE RATES
          FOR J=1 TO L/2 LET RATE (J) = (DIST.ARRIVAL (100,J) - DIST.ARRIVAL (1,J)
              ) / 99.0
          FOR J=L/2+1 TO L DO
              FOR I=1 TO 100 LET DIST.ARRIVAL (I,J)=1.0/DIST.ARRIVAL(I,J)
              FOR I= 1 TO 100 COMPUTE RATE (J) = MEAN, VAR.SERV (J-L/2) = VAR OF
                      DIST.ARRIVAL (I,J)
          LOOP
          LIST RATE, VAR.SERV, NO.OF.QUERIES, CR
          ''SCALE TO OBTAIN INTERARRIVAL TIMES
          FOR J=1 TO L/2 FOR I BACK FROM 100 TO 2 LET DIST.ARRIVAL (I,J) =
              DIST.ARRIVAL (I,J) - DIST.ARRIVAL (I-1,J)
          FOR J=1 TO L/2 LET DIST.ARRIVAL (1,J)=0.0  SKIP 1 LINE
          FOR J=1 TO L/2 DO FOR I=2 TO 100 COMPUTE MN=MEAN, BIG=MAX, SMALL=MIN
              OF DIST.ARRIVAL (I,J)
          LET TYPE = "ARRV"
          PRINT 1 LINE WITH TYPE, J, BIG, MN, SMALL THUS
FOR INTER**** RATE   * LONGEST=      *.*** MEAN=   *.**** SHORTEST=  *.*****SECS
          LOOP
          FOR J=L/2+1 TO L DO FOR I=1 TO 100 COMPUTE MN=MEAN, BIG=MAX, SMALL=MIN
              OF DIST.ARRIVAL (I,J)
          LET TYPE = "SERV"
          PRINT 1 LINE WITH TYPE,J - L/2, BIG, MN, SMALL THUS
FOR **** RATE  * LARGEST=     *.*** MEAN=     *.*** SMALLEST=      *.*** SECS
          LOOP
          LET SIZE=DIM.F (DIST.ARRIVAL(*,*)) -1
          RESERVE D1, D2 AS SIZE
          FOR J=1 TO L DO
              FOR I=2 TO 100 COMPUTE MN= MEAN OF DIST.ARRIVAL (I,J)
          IF MN = 0.0 JUMP AHEAD ELSE
              FOR I=1 TO 99 DO LET D1(I) = DIST.ARRIVAL (I+1,J)
              LET D2(I) = EXPONENTIAL.F (MN,E)
              LOOP
          START NEW PAGE
          LET Z.SCORE = ABS.F ( MANN.WHITNEY (SIZE,SIZE))
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                    06/28/76

             IF J<L/2+1 LET TYPE="INTR"  LET MEASURE = "ARPV" ELSE
                 IF J>L/2 AND J<L+1 LET TYPE = "SERV" LET MEASURE = "RATE" ELSE
             LET Q=J - (10* TRUNC.F(J/(N.RAD+1)))
             PRINT 1 LINE WITH TYPE, MEASURE, Q, Z.SCORE THUS
FOR **** **** * Z=     *.*** TESTING EXPONENTIAL DISTRIB. HISTOGRAM BELOW
             IF DISCIPLINE =1 AND E=1 AND (J=1 OR J=2 OR J=5 OR J=8 OR J=11 OR J=12
                 OR J=15 OR J=18)
             START NEW PAGE CALL SORT.D1 (D1(*)) CALL HISTOGRAM
             ELSE
             FOR I=2 TO 100 COMPUTE MN= MEAN, STAN=STD OF DIST.ARRIVAL (I,J)
             FOR I=1 TO 99 LET D2(I) = NORMAL.F (MN,STAN,E)
             START NEW PAGE
             LET Z.SCORE = ABS.F ( MANN.WHITNEY (SIZE, SIZE))
             PRINT 1 LINE WITH TYPE, MEASURE, Q, Z.SCORE THUS
FOR **** **** * Z =     *.**** TESTING NORMAL DISTRIBUTION
             HERE
             LOOP
             RELEASE D1, D2
     RETURN END
```

LOCAL VARIABLES OF THIS ROUTINE

BIG	REAL	WORD 14	D	INTEGER	WORD 1
I	INTEGER	WORD 10	I.1	INTEGER	WORD 25
I.2	INTEGER	WORD 26	I.3	INTEGER	WORD 27
J	INTEGER	WORD 11	J.1	INTEGER	WORD 28
K.1	INTEGER	WORD 30	K.2	INTEGER	WORD 31
K.3	INTEGER	WORD 32	K.4	INTEGER	WORD 33
L	INTEGER	WORD 13	L.21	DOUBLE	WORD 15
L.22	DOUBLE	WORD 17	L.23	DOUBLE	WORD 19
L.24	DOUBLE	WORD 21	L.34	INTEGER	WORD 34
L.42	DOUBLE	WORD 35	L.43	DOUBLE	WORD 37
L.44	DOUBLE	WORD 39	L.45	DOUBLE	WORD 41
L.46	DOUBLE	WORD 43	L.53	DOUBLE	WORD 47
L.54	DOUBLE	WORD 49	L.55	DOUBLE	WORD 51
L.56	DOUBLE	WORD 53	L.57	DOUBLE	WORD 55
L.64	DOUBLE	WORD 57	L.65	DOUBLE	WORD 59
L.66	DOUBLE	WORD 61	L.73	DOUBLE	WORD 65
L.74	DOUBLE	WORD 67	L.75	DOUBLE	WORD 69
L.76	DOUBLE	WORD 71	MEASURE	ALPHA	WORD 9
MN	REAL	WORD 45	NO.DISTR	INTEGER	WORD 7
N.1	INTEGER	WORD 29	Q	REAL	WORD 63
R.1	DOUBLE	WORD 23	SIZE	INTEGER	WORD 12
SMALL	REAL	WORD 46	STAN	REAL	WORD 73
SWITCH	INTEGER	WORD 3	TIME	REAL	WORD 5
TYPE	ALPHA	WORD 8			

LINE CACI SIMSCRIPT II.5 RELEASE 8D 06/28/76

```
ROUTINE TERMINATE
        PRINT 1 LINE THUS
     SIMULATION TERMINATING
        FOR EACH   OFF.CPU             IN EV.S   (I.OFF.CPU)
            DO CANCEL THE OFF.CPU DESTROY THE OFF.CPU LOOP
        FOR EACH RUN.IT IN EV.S (I.RUN.IT)
            DO CANCEL THE RUN.IT DESTROY THE RUN.IT LOOP
        FOR EACH CRT.TRANS.READY IN EV.S  (I.CTR)
            DO CANCEL THE CRT.TRANS.READY DESTROY THE CRT.TRANS.READY LOOP
        FOR EACH CRT.TRANS.FINISHED IN EV.S (I.CRT.TRANS.FINISHED)
            DO CANCEL THE CRT.TRANS.FINISHED DESTROY THE CRT.TRANS.FINISHED LOOP
        FOR EACH SEEK.END IN EV.S (I.SEEK.END)
            DO CANCEL THE SEEK.END DESTROY THE SEEK.END LOOP
        FOR EACH LATENCY.END IN EV.S (I.LATENCY.END)
            DO CANCEL THE LATENCY.END DESTROY THE LATENCY.END LOOP
        FOR EACH READ.END IN EV.S (I.READ.END)
            DO CANCEL THE READ.END DESTROY THE READ.END LOOP
        FOR EACH ANSWER IN EV.S (I.ANSWER)
            DO CANCEL THE ANSWER DESTROY THE ANSWER LOOP
        FOR EACH TELEPHONE.CALL IN EV.S (I.TELEPHONE.CALL)
            DO CANCEL THE TELEPHONE.CALL DESTROY THE TELEPHONE.CALL LOOP
        FOR EACH ARRIVAL.AT.CRT IN EV.S (I.ARRIVAL.AT.CRT)
            DO CANCEL THE ARRIVAL.AT.CRT DESTROY THE ARRIVAL.AT.CRT LOOP
        FOR EACH AN.END IN EV.S (I.AN.END)
            DO CANCEL THE AN.END DESTROY THE AN.END LOOP
  RETURN END
```

LOCAL VARIABLES OF THIS ROUTINE

I.1	INTEGER	WORD	3		I.2	INTEGER	WORD	4
I.3	INTEGER	WORD	5		J.1	INTEGER	WORD	6
K.1	INTEGER	WORD	8		K.2	INTEGER	WORD	9
K.3	INTEGER	WORD	10		K.4	INTEGER	WORD	11
L.12	INTEGER	WORD	14		L.16	INTEGER	WORD	15
L.20	INTEGER	WORD	16		L.24	INTEGER	WORD	17
L.28	INTEGER	WORD	18		L.32	INTEGER	WORD	19
L.36	INTEGER	WORD	20		L.4	INTEGER	WORD	12
L.40	INTEGER	WORD	21		L.44	INTEGER	WORD	22
L.3	INTEGER	WORD	13		N.1	INTEGER	WORD	7
M.1	DOUBLE	WORD	1					

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                      06/28/76

  ROUTINE TRACE
          IF SHUT.OFF.TRACE = 0 LET BETWEEN.V = 0 RETURN ELSE
          GO TO OFF, RUN, READY, FINISHED, SEEK, LATENCY, READ, ANSWER,
          TEL, ARRIVE OR END.ANS PER EVENT.V
   'OFF' PRINT 1          LINE WITH TIME.V AND VR.NO ( VIR.ADDR ), TYPE ( VIR.ADDR )
          THUS
AN OFF.CPU EVENT OCCURRED  AT TIME            *.***** FOR VR        * FOR TYPE    *
          CALL SECOND.TRACE ( VIR.ADDR ) RETURN
   'RUN' PRINT 1 LINE WITH TIME.V AND VR.NO ( VIRT.ADDR ), TYPE ( VIRT.ADDR) THUS
A RUN.IT EVENT OCCURRED AT TIME               *.***** FOR VR        * FOR TYPE    *
          CALL SECOND.TRACE ( VIRT.ADDR ) RETURN
   'READY' PRINT 1 LINE WITH TIME.V AND CRT.NUMB THUS
         A CRT.READY EVENT OCCURRED AT TIME                 *.****** FOR CRT        *
          CALL SECOND.TRACE ( CRT.NUMB ) RETURN
   'FINISHED' PRINT 1 LINE WITH TIME.V AND CATH.NO THUS
         A CRT.FINISH OCCURRED AT TIME                      *.****** FOR CRT        *
          CALL SECOND.TRACE ( CATH.NO ) RETURN
   'SEEK' PRINT 1 DOUBLE LINE WITH TIME.V AND VR.NO ( VIR.SEEK.ADDR ),
              TYPE ( VIR.SEEK.ADDR ), RAD.CHAN ( VIR.SEEK.ADDR ),
              RAD.DEVICE ( VIR.SEEK.ADDR ) THUS
A SEEK EVENT OCCURRED AT TIME                 *.***** FOR VR        * FOR TYPE    *  FOR CHANNEL   * FOR DEVICE   *
          CALL SECOND.TRACE ( VIR.SEEK.ADDR) RETURN
   'LATENCY' PRINT 1 DOUBLE LINE WITH TIME.V AND VR.NO ( VIR.LAT.ADDR ),
              RAD.CHAN ( VIR.LAT.ADDR ), RAD.DEVICE ( VIR.LAT.ADDR ) THUS
         A LATENCY OCCURRED AT TIME                        *.****** FOR VR        *FOR CHANNEL   * FOR DEVICE   *
          CALL SECOND.TRACE ( VIR.LAT.ADDR ) RETURN
   'READ' PRINT 1 DOUBLE LINE WITH TIME.V AND VR.NO ( VIR.READ.ADDR ),
              RAD.CHAN ( VIR.READ.ADDR ), RAD.DEVICE ( VIR.READ.ADDR ) THUS
         A READ OCCURRED AT TIME                           *.****** FOR VR        *FOR CHANNEL   * FOR DEVICE   *
          CALL SECOND.TRACE ( VIR.READ.ADDR ) RETURN
   'ANSWER' PRINT 1 LINE WITH TIME.V AND CRT.NUMBER ( VIR.ADDRESS ) AND
              VR.NO ( VIR.ADDRESS ) THUS
AN ANSWER EVENT OCCURRED AT TIME              *.***** FOR CRT       * FOR VR NO   *
          CALL SECOND.TRACE ( VIR.ADDRESS ) RETURN
   'TEL'  PRINT 1 LINE WITH TIME.V THUS
         A TELEPHONE CALL AT TIME                          *.******
          CALL SECOND.TRACE (DUM)
          RETURN
   'ARRIVE' PRINT 1 LINE WITH TIME.V AND CATHODE.TUBE THUS
         AN ARRIVAL AT CRT AT TIME                         *.****** FOR CRT        *
          CALL SECOND.TRACE ( CATHODE.TUBE )
          RETURN
   'END.ANS' PRINT 1 LINE WITH TIME.V AND VR.NO (VADDR) THUS
AN END OF ANSWER OCCURRED AT TIME             *.****** FOR VR       *
          CALL SECOND.TRACE (VADDR) RETURN
  END
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                  06/28/76

              LOCAL VARIABLES OF THIS ROUTINE

DUM          INTEGER   WORD 12        I.1          INTEGER   WORD  3
I.2          INTEGER   WORD  4        I.3          INTEGER   WORD  5
J.1          INTEGER   WORD  6        K.1          INTEGER   WORD  6
K.2          INTEGER   WORD  9        K.3          INTEGER   WORD 10
K.4          INTEGER   WORD 11        N.1          INTEGER   WORD  7
R.1          DOUBLE    WORD  1
```

```
ROUTINE SECOND.TRACE ( ADDRESS )
        '' THE PARAMETER ADDRESS WILL CONTAIN EITHER A ROUTINE ADDRESS OR A
        '' CRT NUMBER
        DEFINE H1 AND H2 AS ALPHA VARIABLES
        IF PRINT.CR = 1 RETURN ELSE
        IF DESIRE.PRINT = 1 GO TO PRINT ELSE
        GO TO ONE, ONE, TWO, TWO, ONE, ONE, ONE, ONE, THREE, THREE OR THREE
            PER EVENT.V
        'ONE' LET Z = QUERY.NUM ( QUERY.ADDR ( CRT.NUMBER (ADDRESS)))
        IF Z > SHAPE RETURN ELSE
        FOR I=1 TO MAT.SIZE WITH TRACE.MATRIX (Z,I,1)=0.0 FIND ROW= THE FIRST I
        IF NONE PRINT 1 LINE THUS
    TRACE MATRIX TABLE IS FULL
            GO TO PRINT ELSE
        LET TRACE.MATRIX (Z,ROW,1)=Z
        LET TRACE.MATRIX (Z,ROW,2) = CRT.NUMBER (ADDRESS)
        LET TRACE.MATRIX (Z,ROW,3)= TIME.V
        LET TRACE.MATRIX (Z,ROW,4)= EVENT.V
        LET TRACE.MATRIX (Z,ROW,5)= TYPE (ADDRESS)
        LET TRACE.MATRIX (Z,ROW,6)= VR.NO (ADDRESS)
        RETURN
        'TWO' LET Z = QUERY.NUM ( QUERY.ADDR (ADDRESS))
        IF Z > SHAPE RETURN ELSE
        IF QUERY.ADDR (ADDRESS) = 0 LET Z = 1 ELSE
        FOR I=1 TO MAT.SIZE WITH TRACE.MATRIX (Z,I,1)=0.0 FIND ROW= THE FIRST I
        IF NONE PRINT 1 LINE THUS
    TRACE MATRIX TABLE IS FULL
            GO TO PRINT ELSE
        LET TRACE.MATRIX (Z,ROW,1)=Z
        LET TRACE.MATRIX (Z,ROW,2)= ADDRESS
        LET TRACE.MATRIX (Z,ROW,3)= TIME.V
        LET TRACE.MATRIX (Z,ROW,4)= EVENT.V
        'THREE' RETURN
        'PRINT' LET PRINT.CR = 1
        FOR DEPTH = 1 TO SHUT.OFF.TRACE +50 UNLESS TRACE.MATRIX (DEPTH,1,1)=0.0
            FOR ROW=1 TO MAT.SIZE UNLESS TRACE.MATRIX (DEPTH,ROW,1)=0.0
        DO          LET K=TRACE.MATRIX(DEPTH,ROW,4)
                    IF K=1 LET H1 = "OFF" ELSE
                    IF K=2 LET H1 ="RUN" ELSE
                    IF K=3 LET H1 ="RDY " ELSE
                    IF K=4 LET H1="FIN" ELSE
                    IF K=5 LET H1 ="SEEK" ELSE
                    IF K=6 LET H1="LAT" ELSE
                    IF K=7 LET H1="REND" ELSE
                    IF K=8 LET H1="ANS" ELSE
                    IF K=9 LET H1="TEL" ELSE
                    IF K=10 LET H1="ARR" ELSE
                    IF K=11 LET H1 ="RESP" ELSE
         LET K=TRACE.MATRIX (DEPTH,ROW,5)
             IF K=1 LET H2 ="CINT" ELSE
         IF K=2 LET H2="FL01" ELSE  ''FIRST LINK ON Q1
             IF K=3 LET H2="BL01" ELSE
         IF K=4 LET H2="LRAD" ELSE     '' LINK TO RAD
```

```
          IF K=5 LET H2="NEWO" ELSE     ''LINK TO G1
          IF K=6 LET H2="ANS" ELSE
          IF K=20 LET H2="OA20" ELSE
          IF K=21 LET H2="PA21" ELSE
          PRINT 1 DOUBLE LINE WITH TRACE.MATRIX (DEPTH,ROW,1),
                                   TRACE.MATRIX (DEPTH,ROW,2),    H1,
                                   TRACE.MATRIX (DEPTH,ROW,3),    H2,
                                   TRACE.MATRIX (DEPTH,ROW,5)   THUS
                                                                  *.*****THE ROUTINE RUN WAS OF TYPE **** HAVING TYPE    *.
FOR QUERY  *.* ON CRT   *.* AN EVENT CALLED **** WAS RUN AT TIME
          LET H1="     " LET H2="    "
          LOOP
          RETURN END
```

LOCAL VARIABLES OF THIS ROUTINE

ADDRESS	INTEGER	WORD 1	DEPTH	INTEGER	WORD 20
H1	ALPHA	WORD 3	H2	ALPHA	WORD 4
I	INTEGER	WORD 6	I.1	INTEGER	WORD 11
I.2	INTEGER	WORD 12	I.3	INTEGER	WORD 13
J.1	INTEGER	WORD 14	K	INTEGER	WORD 21
K.1	INTEGER	WORD 16	K.2	INTEGER	WORD 17
K.3	INTEGER	WORD 18	K.4	INTEGER	WORD 19
N.1	INTEGER	WORD 15	R.1	DOUBLE	WORD 9
ROW	INTEGER	WORD 7	Z	INTEGER	WORD 5

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                          06/28/76

ROUTINE TAPE
          DEFINE SIZE AS AN INTEGER VARIABLE
          USE UNIT 9 FOR OUTPUT
          LET DISCIPLINE = 100
          WRITE DIM.F ( FIR.RESPONSE (*)), WORK.LOAD, BLOCKING.FACTOR, DISCIPLINE,
               DATA.FAULTS, SPEED (1)
               AS D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/
          FOR I=1 TO DIM.F ( FIR.RESPONSE (*)) WRITE FIR.RESPONSE (I) AS D(14,7),/
          LET DISCIPLINE = 200
          WRITE DIM.F ( SEC.RESPONSE (*)), WORK.LOAD, BLOCKING.FACTOR, DISCIPLINE,
               DATA.FAULTS, SPEED (1)
               AS D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/
          FOR I=1 TO DIM.F ( SEC.RESPONSE (*)) WRITE SEC.RESPONSE (I) AS D(14,7),/
          LET DISCIPLINE = 300
          LET SIZE = DIM.F (RESPONSE(*)) - TRANSIENT (3)
          WRITE SIZE, WORK.LOAD, BLOCKING.FACTOR, DISCIPLINE,
               DATA.FAULTS, SPEED (1)
               AS D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/
          FOR I=TRANSIENT (3)+1 TO DIM.F(RESPONSE(*)) WRITE RESPONSE (I) AS
               D(14,7),/
          USE UNIT 6 FOR OUTPUT
          RETURN END


            LOCAL VARIABLES OF THIS ROUTINE

I          INTEGER   WORD 14      I.1        INTEGER   WORD  5
I.2        INTEGER   WORD  6      I.3        INTEGER   WORD  7
J.1        INTEGER   WORD  8      K.1        INTEGER   WORD 10
K.2        INTEGER   WORD 11      K.3        INTEGER   WORD 12
K.4        INTEGER   WORD 13      N.1        INTEGER   WORD  9
R.1        DOUBLE    WORD  3      SIZE       INTEGER   WORD  1
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                    07/12/76

          PREAMBLE
'' ************************************************************************************
'' *                                                                                  *
'' *                  PRODUCT INFORMATION SYSTEM                                      *
'' *                  PROGRAM 2                                                       *
'' *                                                                                  *
'' *                  F. PAUL FUHS                                                    *
'' *                                                                                  *
'' *                  GRAPHS AND STATISTICS ON SYSTEM RESPONSE TIMES AND              *
'' *                  LOST POTENTIAL SALES                                            *
'' *                                                                                  *
'' ************************************************************************************
NORMALLY MODE IS REAL
DEFINE ELSE TO MEAN ALWAYS
THE SYSTEM OWNS AN OVFL.AREA
TEMPORARY ENTITIES
          EVERY OVERFLOW HAS A VALUE AND MAY BELONG TO AN OVFL.AREA
               DEFINE VALUE AS A REAL VARIABLE
DEFINE OVFL.AREA AS A SET RANKED BY LOW VALUE
DEFINE FIR.GOOD AS A DOUBLE VARIABLE
DEFINE AAA, BBB, CCC, Z.SCORE, CORREL, CUT.OFF, F,
          MED.TEST AS REAL VARIABLES
DEFINE PAGE.HEIGHT, N, COMPARISONS, BEGIN, END, HOLD1, HOLD2, HOLD3
AS INTEGER VARIABLES
DEFINE SLOTS, TAPE.NO, TAG.STOP, NN, FREQ.BLKS, DEPEND.CODE AS AN INTEGER
          VARIABLE
DEFINE D1, D2 AS A 1-DIM ARRAY
DEFINE MINI, MAXIM AS 1-DIM ARRAYS
DEFINE FREQ AS A 2-DIM ARRAY
DEFINE LAGS AS AN INTEGER, 1-DIM ARRAY
DEFINE ORG AS A 2-DIM ARRAY
DEFINE DATA AS A 2-DIM ARRAY
DEFINE WHICH AS A 2-DIM INTEGER ARRAY
DEFINE PROP AS A 2-DIM INTEGER ARRAY
DEFINE ALPHA.VALUE AS AN ALPHA VARIABLE
DEFINE PLOT AS A 2-DIM ALPHA ARRAY
DEFINE LOSS.FUNCTION AS A REAL FUNCTION
END
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                   07/12/76

   MAIN
   DEFINE M AS A REAL VARIABLE
   DEFINE K, L AS INTEGER VARIABLES
   DEFINE NU, WO, BLK, FAUL, UNIQ AS A 1-DIM DOUBLE ARRAY
   DEFINE DATA.HOLD AS A 2-DIM ARRAY
   DEFINE HEAD AS AN ALPHA VARIABLE
   DEFINE CEASE, I, II, III, IIII, J, JJ, JJJ, JJJJ, X, Y AS INTEGER VARIABLES
   DEFINE PLACE AS AN INTEGER VARIABLE
   RESERVE NU, WO, BLK, FAUL,UNIQ AS 2
   RESERVE MINI, MAXIM AS 3
   LET AAA=0.015 LET BBB=1.2 LET CCC=1.0
   READ HEAD, TAPE.NO
   PRINT 1 LINE WITH TAPE.NO THUS
PROCESSING TAPE   *
   READ HEAD, CEASE READ HEAD, NU(1), WO(1), BLK(1), FAUL(1),UNIQ(1),
            NU(2), WO(2),BLK(2),FAUL(2),UNIQ(2)
   '' TO GET A MATCH WITHOUT ROUNDING MISMATCHES
   LET UNIQ(1) = TRUNC.F(UNIQ(1) * 10000.0)
   LET UNIQ(2) = TRUNC.F(UNIQ(2) * 10000.0)
   LIST CEASE, NU, WO, BLK, FAUL, UNIQ
   READ HEAD, COMPARISONS, HEAD,PAGE.HEIGHT
   READ HEAD, SLOTS
   RESERVE LAGS AS SLOTS        READ LAGS       LIST PAGE.HEIGHT, LAGS
   RESERVE WHICH AS 3 BY 2
   LET WHICH (1,1) = 1 LET WHICH (1,2)= 2 LET WHICH (2,1)=1 LET WHICH(2,2)=3
   LET WHICH (3,1)=2 LET WHICH(3,2)=3
   USE 9 FOR INPUT
   LET EOF.V = 1
   '' FIND FIRST SET OF DATA
   'START'
   ADD 1 TO DATA.SET PRINT 1 LINE WITH DATA.SET THUS
LOOKING AT DATA SET   *
   READ NUM, WORK.LOAD, BLK.FACTOR, DISCIP, DAT.FAULTS, SPEED.CPU, FIR.GOOD AS
            D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/
   LET FIR.GOOD = TRUNC.F(FIR.GOOD * 10000.0)
   IF NUM = NU(1) AND WORK.LOAD=WO(1) AND BLK.FACTOR=BLK(1) AND DAT.FAULTS=FAUL(1)
        .   AND FIR.GOOD = UNIQ(1) JUMP AHEAD ELSE
   FOR I = 1 TO NUM -1 READ TRASH AS D(14,7),/
   GO TO START
   HERE
   'AGAIN'
   IF NUM = NU(2) AND WORK.LOAD=WO(2) AND BLK.FACTOR = BLK(2) AND DAT.FAULTS =
            FAUL(2) AND FIR.GOOD = UNIQ(2) OR CEASE = 1 LET TAG.STOP = 1 ELSE
   LET FIR.GOOD = FIR.GOOD / 10000.0
   IF DISCIP = 100 LET DISCIP = 1 ELSE
   IF DISCIP = 200 LET DISCIP = 2 ELSE
   IF DISCIP = 300 LET DISCIP = 3 ELSE
   START NEW PAGE
   PRINT 5 LINES WITH NUM, WORK.LOAD, BLK.FACTOR, DISCIP, DAT.FAULTS THUS
--------------------------------------------------------------------------------
SYSTEM SPACE STATISTICS FOR: NUMBER=    * WORK LOAD = * MEMORY FACTOR = *
DISCIPLINE = * DATA FAULTS =  *
PLOT NUMBER IS
```

```
RESERVE ORG AS 7300 BY 3
        LET ORG (1,1)=FIR.GOOD
        FOR K=2 TO NUM READ ORG (K,1) AS D(14,7),/
FOR J = 2 TO 3 DO
READ NUM, WORK.LOAD, BLK.FACTOR, DISCIP, DAT.FAULTS, SPEED.CPU AS
        D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/
        FOR K = 1 TO NUM READ ORG(K,J) AS D(14,7),/
LOOP
''RESHAPE MATRIX CALLED ORG
FOR I=1 TO 7300 FOR J=1 TO COMPARISONS WITH ORG (I,J) = 0.0 FIND THE FIRST CASE
LET N = I - 1
RESERVE DATA.HOLD AS N BY 3
FOR J = 1 TO COMPARISONS
        DO FOR I BACK FROM 7300 TO 1
            DO IF ORG (I,J) NE 0.0 LEAVE ELSE
            LOOP
        FOR II = 1 TO N LET DATA.HOLD (II,J) = ORG(I-II+1, J)
        LOOP
RELEASE ORG RESERVE ORG AS N BY 3
FOR I = 1 TO N FOR J = 1 TO COMPARISONS LET ORG (I,J) = DATA.HOLD (I,J)
RELEASE DATA.HOLD
'' BEGIN LOOP FOR EACH LAG     ------------------------------------
FOR III= 1 TO SLOTS DO
LET Q = N
LET NN=TRUNC.F(Q/LAGS(III))
RESERVE DATA AS NN BY 3
FOR J=1 TO COMPARISONS
        FOR I = 1 TO NN
                 DO LET X=1+(LAGS(III)*(I-1)) LET Y=(LAGS(III)*(I-1))+LAGS(III)
                         FOR II=X TO Y COMPUTE M = MEAN OF ORG(II,J)
                         LET DATA(I,J)=M
                 LOOP
'NEXT'
IF DEPEND.CODE = 1 FOR J=1 TO COMPARISONS FOR I=1 TO NN
     .    LET DATA(I,J)=LOSS.FUNCTION (DATA(I,J))
ELSE
''CALCULATE FREQUENCY DISTRIBUTIONS
IF DEPEND.CODE = 0 LET FREQ.BLKS = 260
RESERVE FREQ AS FREQ.BLKS + 1 BY COMPARISONS
RESERVE PLOT AS PAGE.HEIGHT BY FREQ.BLKS
FOR K = 1 TO COMPARISONS
        FOR J = 1 TO NN
             DO IF DATA (J,K) < 1.0 ADD 1 TO FREQ (1,K) CYCLE ELSE
                    IF DATA(J,K) > FREQ.BLKS ADD 1 TO FREQ(FREQ.BLKS+1,K) CYCLE
                          ELSE
                ADD 1 TO FREQ (DATA(J,K),K)
             LOOP
ELSE
IF DEPEND.CODE = 1 RESERVE FREQ AS 130 BY 3
LET FREQ.BLKS = 130
RESERVE PLOT AS PAGE.HEIGHT BY 130
FOR K = 1 TO COMPARISONS FOR J = 1 TO NN
```

```
        DO LET PLACE = DATA (J,K)*130.0
            IF PLACE < 1 ADD 1 TO FREQ (1,K) CYCLE ELSE
            IF PLACE > 130 ADD 1 TO FREQ (130,K) CYCLE ELSE
            ADD 1 TO FREQ (PLACE,K)
        LOOP
ELSE
FOR K = 1 TO COMPARISONS
        ALSO
        FOR J = 1 TO FREQ.BLKS COMPUTE LARGEST = MAXIMUM OF FREQ (J,K)
''CALCULATE PROPORTIONS FOR GRAPH
        RESERVE PROP AS COMPARISONS BY 3
FOR K = 1 TO FREQ.BLKS
        DO FOR J=1 TO COMPARISONS
            DO LET PROP (J,1)=(PAGE.HEIGHT*FREQ(K,J))/LARGEST
               LET PROP (J,2)= J
            LOOP
'' CHECK SIMILAR PROPORTIONS
        FOR J=1 TO COMPARISONS IF PROP(J,1)>PAGE.HEIGHT LET PROP(J,1)=
                    PAGE.HEIGHT ELSE
            IF PROP(1,1) NE PROP(2,1) AND PROP(1,1) NE PROP(3,1)
                   AND PROP (2,1) NE PROP(3,1)
                   LET PROP (1,3)= 1
                   LET PROP (2,3)= 2
                   LET PROP (3,3)= 3 JUMP AHEAD
                   ELSE
            IF PROP (1,1) = PROP (2,1) AND PROP(1,1) = PROP (3,1)
                   LET PROP (1,3)= 7
                   LET PROP (2,3)= 7
                   LET PROP (3,3)= 7 JUMP AHEAD ELSE
            IF PROP (1,1) = PROP (2,1)
                   LET PROP (1,3)= 4
                   LET PROP (2,3)= 4
                   LET PROP (3,3)= 3 JUMP AHEAD ELSE
            IF PROP (1,1) = PROP (3,1)
                   LET PROP (1,3)= 5
                   LET PROP (2,3)= 2
     :             LET PROP (3,3)= 5 JUMP AHEAD ELSE
            IF PROP (2,1) = PROP (3,1)
                   LET PROP (1,3)=1
                   LET PROP (2,3)=6
                   LET PROP (3,3)=6 JUMP AHEAD ELSE
HERE
        ''SORT PROP
            FOR J= 1 TO COMPARISONS FOR I = 1 TO COMPARISONS -1
              DO   IF PROP(I,1)> PROP(I+1,1)
                         LET HOLD1= PROP(I,1) LET HOLD2= PROP(I,2)
                         LET HOLD3 = PROP(I,3)
                         LET PROP(I,1)=PROP(I+1,1) LET PROP(I,2)=PROP(I+1,2)
                         LET PROP(I,3)=PROP(I+1,3)
                         LET PROP(I+1,1)=HOLD1 LET PROP(I+1,2)=HOLD2
                         LET PROP(I+1,3)=HOLD3
                   ELSE
              LOOP
```

LINE CACI SIMSCRIPT II.5 RELEASE 8D 07/12/76

```
'' FILL GRAPH WITH ALPHA VALUES
        LET BEGIN = 0 LET TOGGLE = 0.0
        FOR J=1 TO COMPARISONS
            DO IF PROP(J,1) = 0 CYCLE ELSE
                   CALL ALPHA.TRANS (J)
                   IF TOGGLE = 0.0 LET TOGGLE = 1.0 FOR L=1 TO PROP(J,1)
                        LET PLOT (L,K)=ALPHA.VALUE ADD PROP(J,1) TO BEGIN
                        CYCLE ELSE
                   FOR L=1 TO PROP(J,1) - PROP(J-1,1)
                        UNLESS BEGIN + 1 > PAGE.HEIGHT
                        DO LET PLOT (BEGIN+1,K) =ALPHA.VALUE ADD 1 TO BEGIN LOOP
            LOOP
   LOOP '' END OF K LOOP
CALL SECOND
RELEASE FREQ, PROP, PLOT
IF DEPEND.CODE = 0 LET DEPEND.CODE = 1 GO TO NEXT ELSE
IF DEPEND.CODE = 1 LET DEPEND.CODE = 0 ELSE
RELEASE DATA
LOOP   ''END OF III LOOP    (NEXT LAG)
IF TAG.STOP = 1 GO TO END.IT ELSE
RELEASE ORG
READ NUM, WORK.LOAD, BLK.FACTOR, DISCIP, DAT.FAULTS, SPEED.CPU, FIR.GOOD AS
        D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/,D(14,7),/
LET FIR.GOOD = TRUNC.F(FIR.GOOD * 10000.0)
GO TO AGAIN
'END.IT' SKIP 5 LINES PRINT 1 LINE THUS
----------- NORMAL END OF JOB -----------
END
```

LOCAL VARIABLES OF THIS ROUTINE

BLK	DOUBLE	WORD 6	BLK.FACTOR	REAL	WORD 39
CEASE	INTEGER	WORD 11	DAT.FAULTS	REAL	WORD 41
DATA.HOLD	REAL	WORD 9	DATA.SET	REAL	WORD 36
DISCIP	REAL	WORD 40	FAUL	DOUBLE	WORD 7
HEAD	ALPHA	WORD 10	I	INTEGER	WORD 12
I.1	INTEGER	WORD 25	I.2	INTEGER	WORD 26
I.3	INTEGER	WORD 27	II	INTEGER	WORD 13
III	INTEGER	WORD 14	IIII	INTEGER	WORD 15
J	INTEGER	WORD 16	J.1	INTEGER	WORD 28
JJ	INTEGER	WORD 17	JJJ	INTEGER	WORD 18
JJJJ	INTEGER	WORD 19	K	INTEGER	WORD 2
K.1	INTEGER	WORD 30	K.2	INTEGER	WORD 31
K.3	INTEGER	WORD 32	K.4	INTEGER	WORD 33
L	INTEGER	WORD 3	L.2	INTEGER	WORD 34
L.3	INTEGER	WORD 35	L.50	DOUBLE	WORD 45
L.51	DOUBLE	WORD 47	L.52	DOUBLE	WORD 49
L.73	DOUBLE	WORD 51	L.74	DOUBLE	WORD 53
L.75	DOUBLE	WORD 55	L.99	INTEGER	WORD 59
LARGEST	REAL	WORD 57	M	REAL	WORD 1
N.1	INTEGER	WORD 29	NU	DOUBLE	WORD 4
NUM	REAL	WORD 37	PLACE	INTEGER	WORD 22

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                    07/12/76

 O            REAL     WORD 44      R.1         DOUBLE   WORD 23
 SPEED.CPU    REAL     WORD 42      TOGGLE      REAL     WORD 58
 TRASH        REAL     WORD 43      UNIO        DOUBLE   WORD  8
 WO           DOUBLE   WORD  5      WORK.LOAD   REAL     WORD 38
 X            INTEGER  WORD 20      Y           INTEGER  WORD 21
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                    07/12/76

ROUTINE ALPHA.TRANS (J)
DEFINE J AS AN INTEGER VARIABLE
        IF PROP (J,3) = 1 LET ALPHA.VALUE = "1" ELSE
        IF PROP (J,3) = 2 LET ALPHA.VALUE = "2" ELSE
        IF PROP (J,3) = 3 LET ALPHA.VALUE = "3" ELSE
        IF PROP (J,3) = 4 LET ALPHA.VALUE = "4" ELSE
        IF PROP (J,3) = 5 LET ALPHA.VALUE = "5" ELSE
        IF PROP (J,3) = 6 LET ALPHA.VALUE = "6" ELSE
        IF PROP (J,3) = 7 LET ALPHA.VALUE = "7" ELSE
        IF PROP (J,3) = 8 LET ALPHA.VALUE = "8" ELSE
        IF PROP (J,3) = 9 LET ALPHA.VALUE = "9" ELSE
RETURN END

          LOCAL VARIABLES OF THIS ROUTINE

 J         INTEGER  WORD  1
```

```
ROUTINE SORT.D1 (D1)
        '' THIS ROUTINE SORTS A GIVEN VECTOR CALLED D1
        DEFINE D1 AS A 1-DIM REAL ARRAY
        DEFINE BUCKETS AS A 1-DIM ARRAY
        DEFINE DIMENSION, HIGH, LOW, REFINE, INTERVAL, HOLD,
            BOT, TOP, PERCENT, TEMP, BUMP AS REAL VARIABLES
        DEFINE FREQUENCY, PROB AS A 1-DIM ARRAY
        DEFINE BOX, I, J, INDEX, BUBBLES AS INTEGER VARIABLES
        DEFINE K AS AN INTEGER VARIABLE
        LET REFINE = 50.0
        RESERVE FREQUENCY, PROB AS REFINE
        LET DIMENSION = DIM.F (D1(*))
        '' SEE IF VECTOR D1 NEEDS TO BE SORTED
        FOR I= 2 TO DIMENSION
            DO IF D1(I) < D1(I-1) GO TO SORT ELSE
        LOOP RELEASE FREQUENCY, PROB RETURN
        '' CALCULATE FREQUENCY DISTRIBUTION
'SORT'  LET HIGH = - RINF.C  LET LOW = RINF.C
        FOR I=1 TO DIMENSION
            DO IF D1(I) > HIGH LET HIGH = D1(I) ELSE
            IF D1(I) < LOW LET LOW = D1(I) ELSE
            LOOP
        LET INTERVAL = (HIGH - LOW)/ REFINE
        FOR I= 1 TO DIMENSION
            DO LET BOX = TRUNC.F((D1(I) - LOW)/ INTERVAL) +1
                   IF BOX > REFINE LET BOX = REFINE ELSE
                   LET FREQUENCY (BOX) = FREQUENCY (BOX) +1
            LOOP
        '' CALCULATE CUMULATIVE PROBABILITY DISTRIBUTION FROM FREQUENCY DISTRIB.
        LET PROB (1) = FREQUENCY (1) / DIMENSION
        FOR I = 2 TO REFINE LET PROB (I)=PROB(I-1)+(FREQUENCY(I)/DIMENSION)
        LET PROB (REFINE) = 1.0  '' TO OFFSET ROUNDING ERRORS
        '' MAP VALUES TO BE SORTED INTO BUCKETS
        RESERVE BUCKETS AS DIMENSION * 2
        LET TEMP = RINF.C
        FOR I= 1 TO DIM.F (BUCKETS (*)) LET BUCKETS (I) = TEMP
        FOR I=1 TO DIMENSION
            DO LET INDEX = TRUNC.F ((D1(I) - LOW)/INTERVAL) +1
                   IF INDEX > REFINE LET INDEX = REFINE ELSE
                   LET BOT = LOW + (INDEX * INTERVAL) - INTERVAL
                   LET TOP = BOT + INTERVAL
                   LET PERCENT = (D1(I) - BOT) / INTERVAL
                   IF INDEX NE 1 LET BOX = (PROB(INDEX - 1) + PERCENT *
                         (PROB(INDEX) - PROB (INDEX -1))) * DIM.F (BUCKETS(*))
                   GO TO NEXT ELSE
                   IF INDEX = 1 AND PERCENT > 0.0
                      LET BOX = PERCENT * PROB (1) * DIM.F (BUCKETS (*))
                   GO TO NEXT
                   ELSE LET BOX = 1
         'NEXT'
                   IF BOX = 0 LET BOX = 1 ELSE
                   IF BOX > DIM.F(BUCKETS(*)) LET BOX = DIM.F (BUCKETS(*)) ELSE
```

```
                          LET HOLD = D1(I)
          'AGAIN'         IF BUCKETS (BOX) = TEMP LET BUCKETS (BOX) = HOLD CYCLE ELSE
                          IF HOLD < BUCKETS (BOX) LET BUMP = BUCKETS (BOX)
                                LET BUCKETS (BOX) = HOLD
                                LET HOLD = BUMP ELSE
                          IF BOX NE DIM.F (BUCKETS (*)) LET BOX = BOX + 1
                          GO TO AGAIN ELSE    ''OVERFLOW
                          CREATE AN OVERFLOW
                          LET VALUE (OVERFLOW) = HOLD
                          FILE OVERFLOW IN OVFL.AREA
                  LOOP
          '' PUT BUCKETS AND OVERFLOW QUEUE BACK INTO D1
          LET J=1
          FOR I=1 TO DIM.F (BUCKETS (*))
              DO IF BUCKETS (I) = TEMP CYCLE ELSE
                          LET D1(J) = BUCKETS (I)
                          LET J=J+1
              LOOP
          IF OVFL.AREA IS NOT EMPTY
          LET K = N.OVFL.AREA
          FOR I= 1 TO K
                          DO REMOVE FIRST OVERFLOW FROM OVFL.AREA
                                LET D1(J) = VALUE (OVERFLOW)
                                LET J=J+1
                                DESTROY THE OVERFLOW
              LOOP
          ELSE
          ''TEST GOODNESS OF SORT
          FOR I= 2 TO DIMENSION
              DO IF D1(I) < D1(I-1) GO TO DOWN ELSE
          LOOP
          GO TO MEMORY
          'DOWN'
          '' BUBBLE SORT IF NECESSARY
          FOR K = 1 TO DIMENSION
              DO FOR I = 1 TO DIMENSION - K WITH D1(I) > D1(I+1) FIND J =
                          THE FIRST I
                          IF NONE JUMP AHEAD ELSE
                          FOR I = J TO DIMENSION - K WITH D1(I)> D1(I+1)
                                DO LET HOLD = D1(I)
                                LET D1(I) = D1(I+1)
                                LET D1(I+1) = HOLD
                                ADD 1 TO BUBBLES
                          LOOP
              LOOP
          HERE
          LIST BUBBLES
          '' LAST CHECK ON SORT
          FOR I = 2 TO DIMENSION
              DO IF D1(I) < D1(I-1)
              PRINT 1 LINE WITH I, BUBBLES THUS
              RUN ABORTED SORT ROUTINE IN ERROR AT      * BUBBLES*
          PRINT 1 LINE WITH I, I - 1 THUS
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                        07/12/76

THE VECTOR IS NOT SORTED  I =     *   I-1 =       *
                                  STOP
                            ELSE
                            LOOP
                      'MEMORY'
                      RELEASE FREQUENCY, PROB, BUCKETS
                      RETURN END

                   LOCAL VARIABLES OF THIS ROUTINE

BOT          REAL       WORD 10       BOX          INTEGER    WORD 17
BUBBLES      INTEGER    WORD 21       BUCKETS      REAL       WORD  3
BUMP         REAL       WORD 14       DIMENSION    REAL       WORD  4
DI           REAL       WORD  1       FREQUENCY    REAL       WORD 15
HIGH         REAL       WORD  5       HOLD         REAL       WORD  9
I            INTEGER    WORD 18       I.1          INTEGER    WORD 25
I.2          INTEGER    WORD 26       I.3          INTEGER    WORD 27
INDEX        INTEGER    WORD 20       INTERVAL     REAL       WORD  8
J            INTEGER    WORD 19       J.1          INTEGER    WORD 28
K            INTEGER    WORD 22       K.1          INTEGER    WORD 30
K.2          INTEGER    WORD 31       K.3          INTEGER    WORD 32
K.4          INTEGER    WORD 33       LOW          REAL       WORD  6
N.1          INTEGER    WORD 29       PERCENT.     REAL       WORD 12
PROB         REAL       WORD 16       R.1          DOUBLE     WORD 23
REFINE       REAL       WORD  7       TEMP         REAL       WORD 13
TOP          REAL       WORD 11
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                    07/12/76

ROUTINE MEDIAN
          ''SEE SIEGEL, NON-PARAMETRIC STATISTICS  P.111 TO 115 ALSO P.20 TO 21
          '' THE MEDIAN TEST DOES NOT ASSUME NORMALITY NOR EQUAL VARIANCES
          '' EXPECTED POWER EFFICIENCY APPROACHES 63% THEREFORE LARGE SAMPLE SIZE
          '' NEEDED
          NORMALLY MODE IS REAL
          DEFINE N AS A REAL VARIABLE
          DEFINE ONE, I AS INTEGER VARIABLES
          DEFINE A, B, C, D AS REAL VARIABLES
          DEFINE CELL AS A 2-DIM REAL ARRAY  RESERVE CELL AS 2 BY 2
          LET ONE = MIN.F (DIM.F(D1(*)), DIM.F (D2(*)))
          LET N = ONE * 2.0
          DEFINE VECTOR AS A 1-DIM ARRAY RESERVE VECTOR AS ONE * 2
          FOR I = 1 TO ONE LET VECTOR (I) = D1(DIM.F(D1(*)) +1-I)
          FOR I = 1 TO ONE LET VECTOR ( I + ONE) = D2(DIM.F(D2(*)) +1-I)
          CALL SORT.D1 ( VECTOR (*))
          LET COMBINED.MEDIAN = VECTOR (ONE)
          FOR I= 1 TO ONE
          DO IF D1(DIM.F(D1(*))+1-I) > COMBINED.MEDIAN ADD 1 TO CELL(1,1) ELSE
             IF D1(DIM.F(D1(*))+1-I) < COMBINED.MEDIAN ADD 1 TO CELL(2,1) ELSE
             IF D1(DIM.F(D1(*))+1-I) = COMBINED.MEDIAN SUBSTRACT 1 FROM N  ELSE
          LOOP
          FOR I=1 TO ONE
          DO IF D2(DIM.F(D2(*))+1-I) > COMBINED.MEDIAN ADD 1 TO CELL(1,2) ELSE
             IF D2(DIM.F(D2(*))+1-I) < COMBINED.MEDIAN ADD 1 TO CELL(2,2) ELSE
             IF D2(DIM.F(D2(*))+1-I) = COMBINED.MEDIAN SUBSTRACT 1 FROM N ELSE
          LOOP
          '' CALCULATE CHI-SQUARE '' LET A=CELL(1,1) LET B=CELL(1,2)
                                     LET C=CELL(2,1) LET D=CELL(2,2)
          LET PART = ABS.F (A*D - B*C) - N/2.0 LET MORE = PART **2
          LET NUMER = N * MORE
          LET DENOM = (A+B)*(C+D)*(A+C)*(B+D)
          LET MED.TEST = NUMER / DENOM
         RELEASE VECTOR, CELL
 RETURN END

              LOCAL VARIABLES OF THIS ROUTINE

A            REAL      WORD  4      B          REAL      WORD  5
C            REAL      WORD  6      CELL       REAL      WORD  8
COMBINED.M   REAL      WORD 10      D          REAL      WORD  7
DENOM        REAL      WORD 14      I          INTEGER   WORD  3
MORE         REAL      WORD 12      N          REAL      WORD  1
NUMER        REAL      WORD 13      ONE        INTEGER   WORD  2
PART         REAL      WORD 11      VECTOR     REAL      WORD  9
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                  07/12/76

ROUTINE FTEST (S1,S2,L1)    '' TO TEST EQUAL VARIANCES AND MEAN DIFFERENCES
     ''FOR TEST OF EQUAL VARIANCES SEE ALSO COSTIS,STATISTICS FOR BUSINESS,P.466
          '' SEE DOWNIE, BASIC STATISTICAL METHODS P.133
          NORMALLY MODE IS REAL
          DEFINE S1, S2 AS REAL VARIABLES
          DEFINE L1 AS AN INTEGER VARIABLE
          DEFINE N AS AN INTEGER VARIABLE
          '' S1 IS FIRST VARIANCE.  S2 IS 2ND VARIANCE. L1 IS SIZE OF DATA VECTORS
          LET F = MAX.F (S1,S2) / MIN.F(S1,S2)
          IF L1 GE 50 AND L1 < 100 LET CUT.OFF = 1.60 ELSE
          IF L1 GE 100 AND L1 < 200 LET CUT.OFF = 1.39 ELSE
          IF L1 GE 200 AND L1 < 300 LET CUT.OFF = 1.26 ELSE
          '' FOR LARGE NUMBER OF OBSERVATIONS THE F DISTRIBUTION TENDS TO
          '' APPROXIMATE A CHI SQUARE / D.F. L1 WITH A MEAN OF 1 AND A STAND. DEV.
          '' OF SQRT.F(2.0 / L1)
          IF L1 GE 300 LET CUT.OFF = 1 + 1.65 * SQRT.F(2.0/L1) ELSE   '' ONE TAIL
     ''
     ''CALCULATE PEARSON R
     ''
     '' D1 AND D2 MUST BE OF THE SAME SIZE
FOR I=1 TO DIM.F(D1(*)) COMPUTE CROSS = SUM OF D1(I) * D2(I)
FOR I=1 TO DIM.F(D1(*)) COMPUTE FIR.SUM = SUM OF D1(I)
FOR I = 1 TO DIM.F(D2(*)) COMPUTE SEC.SUM = SUM OF D2(I)
FOR I=1 TO DIM.F(D1(*))    COMPUTE FIR.SQR = SUM OF D1(I) **2
FOR I=1 TO DIM.F(D2(*)) COMPUTE SEC.SQR = SUM OF D2(I) **2
          LET SQ.FIR.SUM = FIR.SUM **2
          LET SQ.SEC.SUM = SEC.SUM **2
LET N = DIM.F(D1(*))
LET M=N
          LET R = ((N* CROSS)-(FIR.SUM * SEC.SUM)) / SQRT.F( ((N* FIR.SQR) -
                    SQ.FIR.SUM ) *(( N* SEC.SQR ) - SQ.SEC.SUM))
IF R > 1.0 LET R=1.0 ELSE    '' COUNTERACT ANY POSSIBLE ROUNDOFFS BECOMING > 1
LET CORREL = R
     ''CALCULATE DIFFERENCE BETWEEN MEANS FOR CORRELATED DATA
          '' CALCULATE STANDARD DEVIATION OF SAMPLES FROM VARIANCES
          LET S1 = SQRT.F (S1)     LET S2 = SQRT.F (S2)
          LET ST.ER1   = S1 / SQRT.F (M-1) LET ST.ER2   = S2 /SQRT.F(M-1)
          IF ST.ER1**2 + ST.ER2**2-(2*R*ST.ER1*ST.ER2) LE 0.0 LET Z.SCORE = 0.0
               JUMP AHEAD ELSE
          LET ERROR.OF.DIF = SQRT.F ( ST.ER1**2 + ST.ER2**2 -(2*R*ST.ER1*ST.ER2))
          FOR I = 1 TO N COMPUTE M1 = MEAN OF D1(I)
          FOR I = 1 TO N COMPUTE M2 = MEAN OF D2(I)
          LET Z.SCORE = ABS.F ( (M1-M2) / ERROR.OF.DIF )
       HERE
RETURN END
```

LINE CACI SIMSCRIPT II.5 RELEASE 8D 07/12/76

LOCAL VARIABLES OF THIS ROUTINE

```
CROSS          REAL      WORD 15      ERROR.OF.D  REAL     WORD 54
FIR.SQR        REAL      WORD 39      FIR.SUM     REAL     WORD 23
I              REAL      WORD  8      L.10        DOUBLE   WORD 17
L.11           DOUBLE    WORD 19      L.12        DOUBLE   WORD 21
L.16           DOUBLE    WORD 25      L.17        DOUBLE   WORD 27
L.18           DOUBLE    WORD 29      L.22        DOUBLE   WORD 33
L.23           DOUBLE    WORD 35      L.24        DOUBLE   WORD 37
L.28           DOUBLE    WORD 41      L.29        DOUBLE   WORD 43
L.30           DOUBLE    WORD 45      L.34        DOUBLE   WORD 55
L.35           DOUBLE    WORD 57      L.36        DOUBLE   WORD 59
L.4            DOUBLE    WORD  9      L.40        DOUBLE   WORD 63
L.41           DOUBLE    WORD 65      L.42        DOUBLE   WORD 67
L.5            DOUBLE    WORD 11      L.6         DOUBLE   WORD 13
L1             INTEGER   WORD  5      M           REAL     WORD 50
M1             REAL      WORD 61      M2          REAL     WORD 69
N              INTEGER   WORD  7      R           REAL     WORD 51
SEC.SQR        REAL      WORD 47      SEC.SUM     REAL     WORD 31
SQ.FIR.SUM     REAL      WORD 48      SQ.SEC.SUM  REAL     WORD 49
ST.ER1         REAL      WORD 52      ST.ER2      REAL     WORD 53
S1             REAL      WORD  1      S2          REAL     WORD  3
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                                    07/12/76

      ROUTINE LOSS.FUNCTION (D)
              NORMALLY MODE IS REAL
              IF D > 170 LET D = 170 ELSE
              LET EXP = ABS.F ( -CCC*D)
              IF EXP = 0.0 PRINT 1 LINE THUS
---------- D IN LOSS FUNCTION = D    AN ERROR  ----------
              LET D=0.0 RETURN WITH D ELSE
              LET EXP = BBB ** EXP
              LET D = AAA/(AAA+(1.0/EXP))
              RETURN WITH D END

                 LOCAL VARIABLES OF THIS ROUTINE

D            REAL      WORD  1      EXP          REAL      WORD  3
I.1          INTEGER   WORD  7      I.2          INTEGER   WORD  8
I.3          INTEGER   WORD  9      J.1          INTEGER   WORD 10
K.1          INTEGER   WORD 12      K.2          INTEGER   WORD 13
K.3          INTEGER   WORD 14      K.4          INTEGER   WORD 15
N.1          INTEGER   WORD 11      R.1          DOUBLE    WORD  5
```

```
LINE  CACI SIMSCRIPT II.5  RELEASE 8D                              07/12/76

ROUTINE SECOND
        DEFINE I, J, K AS INTEGER VARIABLES
DEFINE II AS AN INTEGER VARIABLE
        DEFINE PAGE AS AN INTEGER VARIABLE
'' PRINT ALPHA GRAPH
FOR PAGE = 1 TO 2 DO
START NEW PAGE IF DEPEND.CODE = 0 PRINT 1 DOUBLE LINE WITH NN THUS
  COMPARISON OF RESPONSE TIME DISTRIBUTIONS AT A SELECTED POINT IN SYSTEM SPACE    NUMBER OF OBSERVATIONS =    *
ELSE IF DEPEND.CODE = 1 PRINT 1 DOUBLE LINE WITH NN THUS
    COMPARISON OF POTENTIAL SALES LOSS DISTRIBUTIONS AT A SELECTED POINT IN SYSTEM SPACE    NUMBER OF OBSERVATIONS =    *
ELSE IF PAGE = 1
        FOR K BACK FROM PAGE.HEIGHT TO 1 DO
            FOR J = 1 TO 130 WRITE PLOT (K,J) AS (130) A 1 LOOP
ELSE
'' PRINT REFERENCE LINE
IF DEPEND.CODE = 0 AND PAGE = 1
        WRITE AS B 1, "1",B 10, "10",B 20,"20",B 30,"30",B 40,"40",B 50,"50",
            B 60, "60", B 70, "70", B 80, "80", B 90, "90", B 99,  "100",
            B 109, "110", B 119, "120", B 129, "130",/
ELSE IF DEPEND.CODE = 1 AND PAGE = 1 WRITE AS B 1, "0.0", B 12, "0.1",
B 25, "0.2", B 38, "0.3", B 51, "0.4", B 64, "0.5",
B 77, "0.6", B 90, "0.7", B 103, "0.8", B 116,"0.9", B 129, "1.0",/
ELSE IF PAGE = 2 FOR K BACK FROM PAGE.HEIGHT TO 1 DO
        FOR J = 131 TO 260 WRITE PLOT (K,J) AS (130) A 1 LOOP
'' PRINT REFERENCE LINE
WRITE AS B 1, "130", B 9, "140", B 19, "150", B 29,"160",B 39,"170",
B 49,"180", B 59, "190", B 69, "200",
B 79, "210", B 89,"220", B 99, "230", B 109, "240", B 119, "250", B 129,
"260",/
ELSE IF DEPEND.CODE = 0
        PRINT 1 LINE THUS
                                                    RESPONSE TIME
ELSE IF DEPEND.CODE = 1 PRINT 1 LINE THUS
                                                    POTENTIAL SALES LOSS
ELSE
IF PAGE = 2 GO TO BOTTOM ELSE
'' CALCULATE STATISTICS + PRINT THEM OUT ''
        FOR K = 1 TO COMPARISONS
            DO FOR J = 1 TO NN COMPUTE AVE=AVG,STD=STD,MINI(K)=MIN,MAXI(K)=MAX
                    OF DATA (J,K)
                    '' SORT VECTOR D1
RESERVE D1 AS NN
                    FOR II = 1 TO NN LET D1(II) = DATA(II,K)
            CALL SORT.D1(D1(*))
            IF D1(1) LE 0.0   PRINT 1 LINE WITH D1(1), J THUS
BAD DATA VALUE IN D1 VECTOR.   VALUE IS      *    DISCIPLINE IS *.
            STOP ELSE
                    LET MED = D1(NN/2.0)
RELEASE D1
'' SET UP FOR F TEST AND Z SCORES
RESERVE D1, D2 AS NN
FOR I = 1 TO NN DO LET D1(I)= DATA(I, WHICH(K,1))
```

LINE CACI SIMSCRIPT II.5 RELEASE 8D 07/12/76

```
          LET D2(I)=DATA(I,WHICH(K,2))
          LOOP
FOR I=1 TO NN COMPUTE V1=VAR OF D1(I)
FOR I=1 TO NN COMPUTE V2=VAR OF D2(I)
CALL FTEST (V1,V2,NN)
CALL MEDIAN
IF DEPEND.CODE = 0
PRINT 1 DOUBLE LINE WITH K, AVE, STD, MED, WHICH(K,1), WHICH(K,2), CUT.OFF,
          F, CORREL, Z.SCORE, MED.TEST THUS
DISTR * AVE=***.**** S.D.=***.**** MEDIAN=***.*** DISTR. ***: F-CUT=**.*** F=**.*** R=*.*** Z=*****.**** MEDIAN TEST=*****.**
ELSE IF DEPEND.CODE = 1
PRINT 1 DOUBLE LINE WITH K, AVE, STD, MED, WHICH(K,1), WHICH(K,2), CUT.OFF,
          F, CORREL, Z.SCORE, MED.TEST THUS
DISTR * AVE=*.***** S.D.=*.***** MEDIAN=*.***** DISTR. *** F-CUT=**.*** F=**.*** R= *.*** Z=*****.** MEDIAN TEST=*****.*
ELSE
                LOOP      ''END OF K
'BOTTOM'
IF DEPEND.CODE = 0 AND PAGE = 2
FOR K=1 TO COMPARISONS PRINT 1 LINE WITH MINI(K),MAXIM(K),FREQ.BLKS,
          FREQ(FREQ.BLKS + 1,K) THUS
MINIMUM = *.* MAXIMUM =    *.* NO. OF VALUES LARGER THAN    * =     *
ELSE
IF DEPEND.CODE = 1 JUMP AHEAD ELSE
LOOP   ''END OF PAGE
HERE RELEASE D1, D2
RETURN END
```

LOCAL VARIABLES OF THIS ROUTINE

AVE	REAL	WORD 31	I	INTEGER	WORD 1
I.1	INTEGER	WORD 9	I.2	INTEGER	WORD 10
I.3	INTEGER	WORD 11	II	INTEGER	WORD 4
J	INTEGER	WORD 2	J.1	INTEGER	WORD 12
K	INTEGER	WORD 3	K.1	INTEGER	WORD 14
K.2	INTEGER	WORD 15	K.3	INTEGER	WORD 16
K.4	INTEGER	WORD 17	L.22	DOUBLE	WORD 19
L.23	DOUBLE	WORD 21	L.24	DOUBLE	WORD 23
L.25	DOUBLE	WORD 25	L.26	DOUBLE	WORD 27
L.27	DOUBLE	WORD 29	L.37	DOUBLE	WORD 35
L.38	DOUBLE	WORD 37	L.39	DOUBLE	WORD 39
L.40	DOUBLE	WORD 41	L.44	DOUBLE	WORD 45
L.45	DOUBLE	WORD 47	L.46	DOUBLE	WORD 49
L.47	DOUBLE	WORD 51	MED	REAL	WORD 33
N.1	INTEGER	WORD 13	PAGE	INTEGER	WORD 5
R.1	DOUBLE	WORD 7	STD	REAL	WORD 32
V1	REAL	WORD 43	V2	REAL	WORD 53