AN INVESTIGATION OF CLUSTER ANALYSIS TECHNIQUES AS A MEANS OF STRUCTURING SPECIFICATIONS IN THE DESIGN OF COMPLEX SYSTEMS

by

TIMOTHY A. HOLDEN

B.S., U. S. Naval Academy
(1972)

SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE
DEGREE OF

OCEAN ENGINEER

AND FOR THE DEGREE OF

MASTER OF SCIENCE IN MANAGEMENT

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June, 1978

Signature of Author.....
Department of Ocean Engineering
May 12, 1978

Certified by.....
Thesis Supervisor, Sloan School of Management

Certified by.....
Thesis Supervisor, Department of Ocean Engineering

Accepted by.....
Chairman, Department Committee

Accepted by.....
Chairman, Departmental Graduate Committee

AN INVESTIGATION OF CLUSTER ANALYSIS TECHNIQUES AS A MEANS OF STRUCTURING SPECIFICATIONS IN THE DESIGN OF COMPLEX SYSTEMS

by

TIMOTHY A. HOLDEN

Submitted to the Department of Ocean Engineering on May 12, 1978 in partial fulfillment of the requirements for the Degree of Ocean Engineer
and to
the Sloan School of Management on May 12, 1978 in partial fulfillment of the requirements for the Degree of Master of Science in Management

ABSTRACT

Complex design problems are characterized by a multitude of competing requirements. The designer of such a system frequently finds the scope of the problem beyond his conceptual abilities and attempts to solve this problem by decomposing the design problem into smaller more manageable subproblems. Since design requirements form the interface between the users of a system and its designers, a disciplined framework is required for the decomposition of the design problem into subproblems which will best satisfy the overall problem objective.

Cluster analysis is a heuristically based technique by which attributes of a system are sorted into groups; such that, the degree of "natural" association is high among members of the same group and low between members of different groups.

The purpose of this thesis is to investigate the use of a specific cluster analysis technique, developed by Dr. Raphael Andreu. As a means of imposing a framework upon the requirements for an existing computer operating system forming the first step in the decomposition of the global design problem into subproblems. It is envisioned that the imposition of such a framework on design requirements will provide new insights and understanding of the relationships among requirements which may verify the design or suggest improvements to the design of a sample operating system.

Stuart Madnick
Professor of Management
Thesis Supervisor

ACKNOWLEDGEMENTS

The author wishes to express his appreciation to the following people for their help and guidance in the preparation and completion of this research: Professors Stuart Madnick of the A. P. Sloan School of Management, thesis supervisor; Chryssostomos Chryssostomidis, Department of Ocean Engineering, thesis advisor; and Kevin J. O'Toole, Department of Ocean Engineering, academic advisor.

The following graduate students in the Sloan School of Management also provided invaluable assistance in the critique of this research: Raphael Andreu, Sid Huff, and Chat-Yu Lam.

The author also wishes to thank his true friends for their assistance and understanding during the three years at MIT.

TABLE OF CONTENTS

(Table of Contents.....Continued) Page

(Table of Contents.....Continued) Page

(Table of Contents.....Continued) Page

LIST OF FIGURES

CHAPTER I

DESCRIPTION OF THE PROBLEMS INHERENT IN LARGE SCALE SYSTEM DESIGN

1.1 Problem Description

The design of complex systems is characterized by many of the following problems as identified by Andreu and Madnick.[1]

- There is no established framework in which the design decisions can be coordinated among various design groups. This can lead to an optimization of sub-problems, but sub-optimization of the aggregate design problem.
- The adaptiveness of the system to changes in operational requirements is made difficult and time consuming since such changes often impact the entire system.
- The incorporation of new technology into an existing system is cumbersome and expensive since there is no systematic means of assessing the impact of new technology on the system operation.
- System performance evaluation may require an enormous model to represent the entire system.

---

[1] *Raphael Andreu and Stuart Madnick, "A Systematic Approach to the Design of Complex Systems: Application to DBMS Design and Evaluation", Center for Information Systems Research, Report 32, Sloan School of Management, MIT (Cambridge, MA, 1977) p.6.*

. The designer has no means to determine if the problem has been completely and consistently defined, or alternatively, over-constrained.

The most common technique currently in use to simplify the design of a complex system is to decompose the global problem into smaller sub-problems. However, without proper guidance, this leads to many of the following problems as documented by Mandel and Chryssostomidis.[2]

. The subdivision of a given problem into lower level problems imposes limitations on accuracy and is, therefore, an approximation. This implies that the optimization of the subproblems does not necessarily lead to total system optimization.

. A designer of a specific sub-problem is likely to have incomplete knowledge of the total problem.

. The decomposition process should be independent of any specific technology or implementation technique.

The designer of a large scale system is faced with a number of possible pitfalls as the size and complexity of the design problem increases. The problems can be loosely defined as a lack of a consistent framework in which to make design decisions. Fred Brooks[3] has defined this problem as

---

[2] *P. Mandel and C. Chryssostomidis, "A Design Methodology For Ships and Other Complex Systems", Phil. Trans. R. Soc., London A.273, (London, 1972), p.87.*

[3] *Fred Brooks, The Mythical Man-Month: Essays on Software Engineering, (Reading, MA), p. 16-17.*

one of conceptual integrity and identified this as the most important consideration in system design. Conceptual integrity in this context dictates rigorous design sequence, for if there is no rigor in the design, the resulting product of the design process is highly idiosyncratic; in the worst case, it is based on the failure history of the participants. As a final measure, rigorous design should survive its implementation and provide a framework for intellectual control of changes to design requirements change.

1.2 System Development Cycle

In order to develop a rigorous and consistent framework for the design process, one must examine structure of the design problem as it exists in general in order to propose improvements to the structure. Although many procedures have been defined for a typical computer software design problem, Andreu favored the following System Development Cycle as proposed by Freeman to illustrate the nature of the design problem.

Figure 1 is a representation of the five steps which Freeman recognized in the design cycle. Each step consists of an input and output and an operation which take place in each step. The function of each step is now further defined from the perspective of the need to establish a framework in which the global design problem may be decomposed.

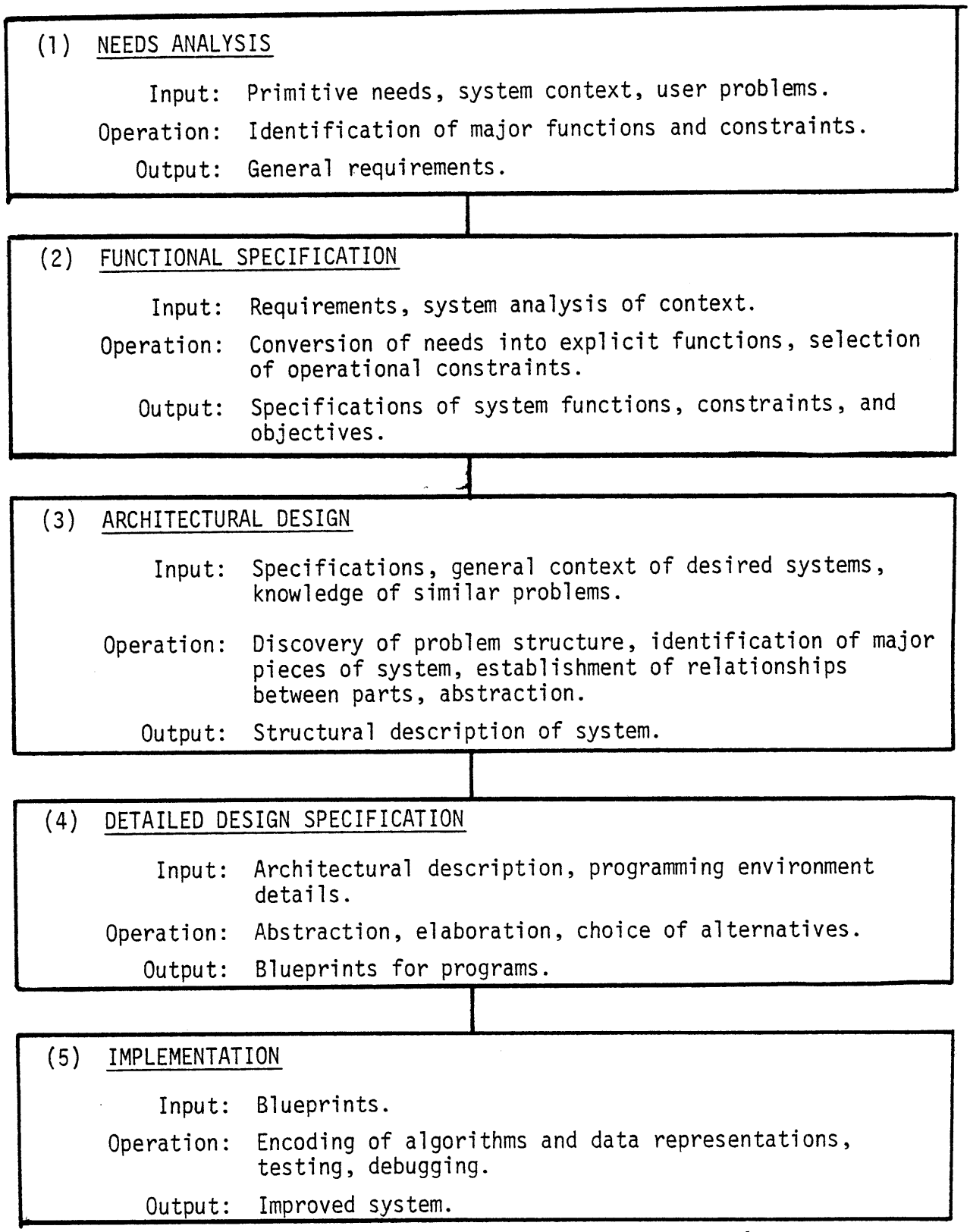


FIGURE 1.1: The System Development Cycle[4]

[4]*Raphael Andreu, "A Systematic Approach to the Design and Structuring of Complex Software System", unpublished Doctoral thesis, MIT Sloan School of Management, February, 1978.*

1.2.1 NEEDS ANALYSIS:

This stage of the design process incorporates a careful assessment of the needs which the final system must fulfill. This stage is generally the most unstructured of all the stages; since a new system must be designed to respond to the user's percieved needs. The information derived from the stage ranges from the most nebulous of statements of need, to statements of such detail as to actually specify implementation. The lack of structure in this phase of the design process is likely to introduce errors which will be repreated throughout the remaining stages of the design process.

In order to avoid the errors introduced by a poor needs analysis phase and driven by a desire to apply the decomposition methodology to an untested system design, an existing well-documented computer operating system was selected for analysis.

1.2.2 FUNCTIONAL SPECIFICATION:

This stage of the design process is concerned with the development of documentation aids in order to generate formal and accurate statements of the system requirements. Typically, functional specifications are characterized by many of the following properties: completeness, consistency, correctness, testability, non-ambiguity, design freedom, and robustness to change. Obviously, the generation of functional specifications is not an easy task, usually taking

place as an iterative or refining process in which the global system requirements are continually refined until the system is completely defined.

Numerous research efforts are currently underway to formalize the process of functional specifications. One particular method developed by TRW, Defense and Space Systems Group is called the Software Requirements Engineering Methodology (SREM).[5] It is an automated system which attempts to enforce the discipline of a framework in the individual interpretation of the problem by the design engineer to reduce the ambiguity of software requirements and thereby lead to increased consistency in functional specifications.

In addition, other "problem statement languages" developed by Tiechroew and others[6] have identified two classes of requirements; specifically, "functional" requirements, what the system is to do and "performance" requirements, regarding constraints on measures of system behavior.

No attempts were made in this thesis to implement any of the problem statement languages, as such. However, a series of guidelines for requirements definition were established to insure that the requirements had all the characteristics of a "well-defined" set of requirements. The

---

[5] *Carl G. Davis and Charles R. Vick, "The Software Development System", IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, Vol. SE-3, No. 1 (Jan 1977), p.70.*

[6] *Sloan School, MIT, "System Documentation Language Report", unpublished Sloan School report, MIT, Sloan School (Cambridge, MA), p.2.*

classification of functional versus performance requirements, developed in the problem statement languages, were used in the interdependency assessment process.

1.2.3 ARCHITECTURAL DESIGN:

This stage of the design process is concerned with the discovery of problem structure in the design as defined by Freeman. That is, the identification of major sub-problems of the system and the establishment of relationships between these sub-problems. Utilizing this technique, Andreu has noted the existence of both a problem structure and a system structure inherent in each system design.

The problem structure is concerned with how different parts of the system interact from a design standpoint; that is, what parts of the system can be designed independently of others as opposed to what parts must be designed at the same time. The problem structure then is used to identify the trade-offs that must be taken into account between completing solutions of the design problem. The concept of a problem structure was a key element used in the inter-dependency assessment phase.

The system structure, on the other hand, is concerned with how system parts interact once the system is designed and in operation. Andreu has pointed out that the two structures do not necessarily coincide:

> "Traditionally the 'design problem structure' has been determined by the system structure in

> that it is very common to organize the design of a new system around 'standard' system structures, drawn from similar systems previously designed."[7]

As previously stated, a subdivision of a given problem into lower level problems imposes limitations on accuracy and is, therefore, an approximation. Secondly, unless the process is rigorous, it is highly idiosyncratic. The goal of any proposed framework must be to reduce the designer's dependency on "standard" system structures in such a way as to rigorously decompose the design problem into well-defined subproblems.

Therefore, a framework is required at this stage of the design process, to resolve the trade-offs that may exist among system requirements as implied by available alternative implementation techniques. Andreau has proposed a framework which addresses this issue based on cluster analysis techniques. The purpose of the framework is to:

- explicitly establish the nature of the problem by decomposition;
- establish a consistent framework in which trade-offs can be assessed.

The methodology constitutes a well-structured series of activities that the software engineer should perform during the design process. The value of such a methodology, claims Andreu,

---

[1] *Andreu, p. 41.*

"...is that the concept of interrelated design subproblems stemming from the explicit inter-dependencies among requirements, constitutes a better basis for the subsequent detailed design stages than the original disjointed set of requirements."[8]

1.2.4 DETAILED DESIGN SPECIFICATION:

This stage constitutes the actual design of program modules as opposed to system design. The work of Parnas[9] has focused on the means of structuring the software modules in order to implement the system. This stage is beyond the concern of this thesis.

1.2.5 IMPLEMENTATION:

This stage is concerned with the actual programming of the system. Efforts by Liskov[10] have attempted to develop structural programming tools to systematize the activities at this stage of the design. This stage is also beyond the concern of this thesis.

1.3 Summary

The system development cycle is characterized by

---

[8] *Andreu, p.41.*

[9] *David L. Parnas, "On the Criteria to be Used in Decomposing System into Modules", Communications of the ACM., Vol. 15, Number 12 (Dec. 1972), pp. 1053-1058.*

[10] *Barbara Liskov and Valdis Berzins, "An Appraisal of Program Specifications", Computation Structures Group Memo 141-1, MIT, Laboratory for Computer Science (April 1977), p.1-12.*

increasing attempts to structure or systematize each stage of the design process. The purpose of this thesis is to apply the techniques developed by Andreu in order to verify the design of the computer operating system under investigation by the application of the methodology as proposed by Andreu.

1.4 Thesis Outline

The computer operating system entitled, "The Sample Operating System (SOS)" was developed by Professor Madnick and Professor Donovan[11] of the MIT Sloan School of Management. This system design problem was selected for examination since it is a reasonably non-trivial and well-documented software design problem.

Chapter II is devoted to a discussion of cluster analysis techniques in general, and a description of the specific methodology proposed by Andreu.

Chapter III presents a description of the general characteristics of the sample operating system.

Chapter IV presents both the procedure and the results of the requirements definition phase for the sample operating system. This chapter presents in detail the guidelines which were used to generate the requirements and, by example, demonstrate some of the pitfalls encountered in requirements definition.

---

[11] *Stuart E. Madnick and John J. Donovan, Operating Systems, (New York, 1974), pp.381-431.*

Chapter V presents the methodology for the interdependency assessment phase and the resulting input for analysis utilizing Andreu's methodology.

Chapter VI presents the results of the first decomposition using the analytic techniques previously described. These results are considered an intermediate step; therefore, the results are analyzed as motivation to continue along in the next set decomposition.

Chapter VII presents the results of the second decomposition analysis and compares the results with those previously obtained.

Chapter VIII presents a comparison of the design framework implied by the decomposition methodology vis-a-vis the actual design of the sample operating system.

The final chapter will present suggestions for changes or improvements to the cluster analysis techniques proposed by Andreu, based on the experience of the user.

CHAPTER II

CLUSTER ANALYSIS METHODOLOGY AND THE DECOMPOSITION FACILITY

This chapter is devided into three sections in order to present the cluster analysis methodology as applied to the general decomposition problem.

First, the need for such a methodology is motivated by establishing the objective of such cluster analysis techniques and the types of problems encountered in the application of the methodology. Definitions of general terms are offered for use through the rest of the discussion.

Second, a solution to the decomposition problem using cluster analysis techniques is defined. Specifically, the decomposition problem is defined and the techniques for partitioning the requirements set are presented according to the work by Andreu.

Finally, the use of the decomposition software analysis techniques developed by Andreu are presented.

2.1 Cluster Analysis Problem

Cluster analysis techniques may be defined as analysis techniques to sort the attributes of objects into groups such that the degree of natural association is high among members of the same group and low between members of different groups. When successfully applied, the techniques

can be used to reveal problem structure as relationships that exist for a given set of data.

In order to apply these techniques, one must be capable of the following:

- Definition of a group of objects to be clustered; in this case, design requirements for a computer operating system.
- Selection and common definition of attributes common to all objects in this group; in this case, the singular attribute selected was the existence of an interrelationship between a given pair of requirements. The definition of interrelationship will be discussed in Chapter V.
- Definition of an evaluation parameter so that the degree of natural association among members of clusters may be measured.
- Definition of an algorithm to find the best partition of a group of objects. Specifically, an algorithm which defines a partition with the "best" measure evaluation parameter without having to evaluate all the possible partitions.

The following definitions have been applied to the cluster analysis problem.

In general, a group of objects O, may be defined as follows:

Let $O : \{O_1, \ldots O_i \ldots O_N\}$ be the set of objects in

which the clusters are to be identified. These are composed of individual design requirements and also represent the nodes of any graphs which are drawn. |N| may be defined as the cardinality of a set of objects; that is, the number of objects within a given set.

Each object may be characterized by a set of attributes:

X : $\{X_1 \ldots X_j \ldots X_N\}$ measured in some consistent scale. In the case of the discussion, the attribute is the existence of interdependencies.

Therefore, introducing a slight change of notation, an object $O_i \varepsilon O$ is characterized by a vector.

A :{(a_{ij}, aij = 1 if nodes O_i and O_j are related, an interdependency exists; = 0 otherwise. This is the so-called adjacency matrix in which it is assumed that aij = 1 when i = j.

The adjacency matrix is constructed by making a pair-wise assessment of the relationships among all pairs of requirements. The adjacency matrix is simply an NXN matrix, where N is the number of requirements objects. Once a set of objects and their interrelationships have been established the next problem is defining evaluation parameters to measure the degree of natural association.

2.1.2 EVALUATING SET DECOMPOSITIONS:

Any method for evaluating the success of a decomposition scheme must consider the strength of intra-subset relation-

ships, and some means for combining these two parameters. Therefore, the following evaluation parameters were defined by Andreu.[12]

Strength: A measure of how tightly coupled the modes in a given subgraph are is defined as follows:

$$S = \frac{\left(\begin{array}{l}\text{Number of links joining nodes}\\ \text{in the same subset}\end{array}\right) - (N - 1)}{N\ (N-1)\,|\,2}$$

where a subgraph is a graph composed of a subset of the original members of the total graph of nodes to be decomposed. Strength is evaluated by measuring the number of links joining nodes in the same subset minus N-1, N being the cardinality of the given subset, normalized by a factor of N(N-1)/2. In a subset of N nodes, N-1 is the minimum number of interdependencies which can form as subgraphs without disjointed components; thus, the number of links in excess of N-1 is a measure of subset internal coherence, beyond the minimum required for it to be coherent at all. The factor N(N-1)/2 is the maximum number of links that may exist in a subset of cardinality N; normalizing by the factor permits comparable measures for subsets of different cardinality.

Coupling: a measure of the extent to which two subsets are independent, and is defined as follows:

$$C = \frac{\left(\begin{array}{l}\text{Number of links actually joining}\\ \text{nodes of two different subsets}\end{array}\right)}{N \cdot M}$$

[12] *Andreu, p.100.*

In order to evaluate the coupling parameter, the number of interdependencies established between two nodes in different subsets are counted and normalized by the factor N*M; where N and M are the cardinalities of the two subsets.

Measure: The final evaluation parameter of clustering success for a given partition may be defined as follows:

$$M = \sum_{i=1}^{P} S_i - \sum_{\substack{i=1 \\ j=i+1}}^{P} Cij$$

The measure parameter represents the summation of all the strengths of all subsets in the given partition minus the couplings associated with all possible pairs of subsets.

P is defined as a partition or subgraph of the original requirements set. Appendix A contains a formal definition of each of the evaluation parameters listed above.

The parameters are defined so that the measure value should be large to indicate a good evaluation of the natural association of a partition generated by cluster analysis techniques. Therefore, given a group of partitions one would select the partition with the highest value of measure as representing the "best" partition.

Given a requirements set, attributes in the form of interdependencies and evaluation parameters as previously defined; one is now faced with the problem of determining an algorithm which will generate the best partition for a requirements set of non-trivial size.

2.1.3 CLUSTERING SCHEMES:

Given an adjacency matrix and evaluation parameter as previously defined, a technique is now needed to deal with a non-trivial decomposition problem which would not require having to investigate or compute all feasible decompositions that exist for a given requirements set. A heuristically based procedure was selected by Andreu [13] since he has demonstrated that neither an optimization nor a graph theoretic approach is feasible to solve a problem of non-trivial size. Therefore, the various families of cluster analysis techniques and heuristic graph decomposition techniques were investigated to determine which were the most feasible.

In general, there are two generic types of cluster analysis methods, the heirarchical method and the partitioning method. The following discussion will focus upon the similarities and differences of the two methods, concluding with the rationale for the method selected for use by Andreu.

However, prior to a discussion of actual cluster analysis methods, the following definition of the concept of a distance matrix must be presented to transition from the adjacency matrix of interdependence established between design requirements to a similarity matrix defined cluster analysis techniques. The binary assessment procedure, used for identifying requirement with dependences is simplistic

[13] *Andreu, pp.103-109.*

but it is not useful for defining distances as established for Euclidan geometry. For the purposes of cluster analysis techniques; specifically, computing similarity matrix S, scale conversions may be needed prior to the representation of a pair of objects X_i and X_j into an entry of the form $Sij = f(X_i, X_j)$ in the similarity matrix. The scale conversions must meet the properties of "metrics" which is one type of distance function.

The formal properties of metrics have been identified by Anderberg as follows:

> "Let S be a symbolic representation for a measurement space and let x, y, and z be any three points in S. Then a distance function D is a metric if and only if it satisfies the following conditions:
>
> 1. $D(x,y) = 0$ if and only if $x=y$
> 2. $D(x,y) \geq 0$ for all x and y in S
> 3. $D(x,y) = D(y,x)$ for all x and y in S
> 4. $D(x,y) \leq D(x,z)+D(y,z)$ for all x, y, and z in S"[14]

The first property implies that x is zero distance from itself and that any two points zero distance apart must be identical. The second property prohibits negative distances. The third property implies symmetry by requiring the distance from x to y to be the same as the distance from y to x. The fourth property, the triangle inequality, requires that the length of one side of the triangle be no longer then the sum of the lengths of the other two sides. The satisfaction of

[14] *Michael R. Anderberg, Cluster Analysis for Applications, (New York, 1973), p.99.*

these properties is required so that the concept of distance is the Euclidean distance of elementary geometry. Once the property is established the well-known properties of Euclidean distance geometry can be applied to similarity matrices.

A distance function which satisfies the first three conditions of a metric, but not the triangle inequality is known as a semimetric. Furthermore, a metric which additionally satisfies the following property

$$D(x,y)=MAX\{D(x,z),D(y,z)\} \text{ for all } x, y, z \text{ in } S$$

is called an ultrametric since the latter property is considerably stronger than the triangle inequality.

Andreu[15] has pointed out that the concept of cluster analysis is not a precise technique since it is heuristically based. Furthermore, Blashfield and Aldenderfer[16] have shown that the various cluster analysis methods do, in fact, generate different solutions to the same data. Therefore, the value of the methodology is strictly dependent upon:

1) The number of subsets into which the original set is decomposed, where the maximum = N and the minimum = 1.
2) The extent to which the clusters are individually coherent and collectively are distinctly different.

---

[15] *Andreu, p.113.*

[16] *Roger K. Blashfield and Mark S. Aldenderfer, "A Consumer Report on Cluster Analysis Software", Pennsylvania State University Report (PA, 1973), p.3.*

The two approaches to cluster analysis techniques, discussed in the following section, differ in the means by which they approach a middle-ground solution to either extreme.

2.1.3.1 Agglomerative Techniques:

The first basic method of cluster analysis is called the agglomerative method. The measure of similarity used is Euclidean distance. The methodology begins with N clusters, each object in O is a simple member cluster. The method proceeds as the NxN distance matrix is searched for the two most similar entries, which are then combined to form a cluster. The method continues until all objects belong to one single cluster. This method yields a result which exhibits a strictly heirarchical pattern of relationships, in which the number of levels or ranks equals the number of steps in clustering.

The form of linkage; i.e., the criteria used to join objects together to form clusters, may vary from a single linkage cluster, in which an object is joined to a cluster if it has a certain level of similarity with at least one member of the cluster, to the complete linkage method, which requires that an object must achieve a specified level of similarity with all members of a given cluster before being joined to it.

2.1.3.2 Partitioning Method:

The second method of cluster analysis is called the

partitioning method. The partitioning method differs from the agglomerative method in that the solution does not portray a heirarchical relationship among the entities. The resulting clusters obtained from a partitioning solution are discrete and exist at a single rank.

The method proceeds as follows: the user selects a statistic to be optimized during the cluster analysis; in this case, measure (M). All objects are initially assigned to a single cluster of N objects. The user must choose the number of clusters (K) which are believed to exist in the data. The methodology must then use some scheme to determine K leader/seed objects. These seed objects represent a Kernel of objects about which the remaining objects are clustered. An object is then assigned to a cluster with the nearest centroid. The method then recalculates the centroid of the cluster, and the process is then repeated until there are no membership changes which will improve the overall solution. This method is iterative in the solution technique, as an object may actually change its membership from one cluster to another during the process. The agglomerative method, on the other hand, requires only one pass through the data for a complete solution. The partitioning method is more time-consuming, but allows a certain robustness to the solution since each cluster is re-examined and members may be re-assigned. However, the method requires the specification of certain limiting parameters "a priori" specifically: the

user must specify K, final member of clustering before proceeding with the partitioning.

Another distinction among partitioning methods is related to the calculation of the centroid for each cluster. As pointed out by Blashfield and Aldenderfer,

> "The combinatorial methods require the recalculation of the centroid of a cluster after each change on membership. Non-combinatorial methods calculate centroids only after the complete pass has been made. Therefore, combinatorial method of control calculation is considered to be more conservative."[17]

The partitioning method then avoids the major weakness of the agglomerative method, since the iterative nature of the partitioning method allows early decisions regarding which object is merged into which cluster to be re-examined as the algorithm proceeds. For this reason, the partitioning cluster analysis method was selected for use by Andreau.

In order to implement the partitioning method of cluster analysis, one is faced with the following problems:

1) Conversion of the binary adjacency matrix into a similarity matrix which satisfies the requisite metric properties.
2) Identification of the K parameters which is the number of seed nodes.
3) Identification of the actual nodes which are the seed nodes.

---

[17] *Blashfield, p.9.*

Andreau investigated the use of heuristic graph decomposition techniques, particularly the concept of a "core set" to solve the preceeding problems. The techniques are described in the following section.

2.2 Solution of the Cluster Analysis Problem by the Graph Decomposition Techniques

The purpose of this section is to present the techniques proposed by Andreau for the solution of cluster analysis problems; specifically the conversion of the adjacency matrix and identification of partitions through the use of heuristic graph decomposition techniques.

In order to solve the cluster analysis problems previously defined, Andreau investigated the use of heuristic graph decomposition techniques. The definitions of requirements, interdependencies, and the adjacency matrix still apply to the problem at hand. The following definitions apply to graph decomposition techniques:

Core set: CS_i associated with a node O_i in the set

$CS_i : \{O_j | O_j \text{ S.T. } aij = 1\}$

that is the set of all nodes related to O_i, including itself.

Connectivity of node O_i:

$C_i = |CS_i| - 1$, where $|CS_i|$ is defined as the cardinality of set X

Conceptually, one is searching the adjacency matrix for

objects with a high connectivity whose core sets do not interfere with each other. Once identified, these objects form the Kernel of subsets of objects whose elements are strongly related. As determined by Andreu [18] once the number of Kernel subsets has been identified, the remaining nodes can be assigned to the subsets in which they best fit; where the measure of best fit is as previously defined by the overall measure (M). The actual procedure used to identify the subsets is presented in Appendix B.

The procedure requires the "a priori" specification of the parameter (K) which is related to the number of subgraphs expected to result from the decomposition. Andreu's experiences indicated that obviously $1 < K < N$ where N equals the number of nodes or objects subject to decomposition. Andreu stated more strongly that K should be set at a value somewhat higher than the expected number of subgraphs, yet the lower the value of K, the more conservative is the result since fewer subgraphs will be identified considering the interferences among many core sets. In order to normalize the selection process and to make the facility more robust, the selection process for K was redefined as follows;

K = percentage of the maximum value of connectivity, C_i, for the entire graph.

Note that the value of K has been redefined as a percentage value of C_i; this implies that K should be initially

[18] *Andreu, p.125.*

selected as a high value (80%) in order to yield a conservative result.

Andreu then investigated the possibility of generalizing the definition of the core set as follows:

$$CS_i : \{O_i, O_j \text{ such that the minimum path } O_i \rightarrow O_j \leq P \text{ ; where } P \leq 1$$

Note that in the case where P = 1, this is equivalent to the previous definition of the core set. The definition of P=1 is required in order to specify a minimum path. A more complete explanation is offered by Andreu;[19] briefly the point is that the minimum path among objects O_i, O_j, O_K is as follows:

When minimum path $(O_i \rightarrow O_j) \equiv$ Minimum Path $(O_j \rightarrow O_K)$ then either

1) O_j, O_K are both adjacent to O_i

or 2) O_i is adjacent to neither O_j nor O_k

This is true only in the case where P=1; therefore, Andreu uses P=1 when computing the core sets as previously defined.

A starting point for partitioning cluster analysis has thus been identified, by the calculation and identification of core sets as follows:

1) select K = percentage value of maximum value of of connectivity;

2) select the node with the maximum value of connectivity C_i;

---

[19] *Andreu, p.136.*

3) select the core set, consisting of all objects O_i for which $C_i > K\ (C_iMAX)$.

The final problem involved in generating a partitioning methodology was to develop a method to convert the binary adjacency matrix into a similarity matrix meeting the metric conditions of Euclidean distance; since the single binary coefficients dervied directly from entries in the adjacency matrix fail to meet these properties. Andreu incorporated the "core set" concept previously introduced in order to define entries in the similarity matrix as follows:

$$S_{ij} = i - \frac{|CS_i \quad CS_j|}{|CS_i \quad CS_j|}$$

where S_{ij} = Similarity matrix distance measure between objects O_i and O_j

CS_i = Core set for noe O_i

For the special case, for some pair of nodes O_i and O_k such that $S_{ik}=O$ that is $(CS_i=CS_k)$, then it is true that $S_{ij}=S_{Ck}$ for all j. The nodes O_i and O_k are equivalent with respect to the rest of the graph as described by the matrix S. For cluster analysis purposes, this special case represents the case for which nodes i and k are equivalent. The pair is collapsed to form a single node.

This section has determined that there are several problems which must be solved in order to apply cluster

analysis techniques to set decomposition problems. Andreu has used heuristic graph decomposition techniques in order to:

- Identify the K parameter which represents the maximum value of such nodes for a given graph.
- Convert the adjacency matrix defined by a binary assessment of interrelationships into a similarity matrix meeting the metric properties of Euclidean geometry.

The final section of this chapter will present a stepwise, discussion of the application of these techniques to the decomposition problem.

2.3 Decomposition Methodology

The decomposition problem was analyzed utilizing a software package by Andreu. The package is written in Fortran and runs on the PRIME computer system of the Sloan School of Management.

The features available with this system are as follows:

1) Enter the adjacency matrix developed from the requirements interdependency assessment. This function is performed using the "ENGR" command.
2) Compute a distance matrix for the graph under analysis using the "DIMN" command. The package actually computes the distance matrix P=1 is assumed by the package, also it treats collapsed nodes not as single nodes.

3) Compute the similarity matrix from the distance matrix using the "SIMA" command.

4) Generate an initial partition using the "INPA" command to identify the "core of subgraphs" likely to exhibit high strength. The user must specify value for the K parameter.

5) Use the clustering algorithms to generate clusters and return a value for measure, strength, and coupling.

There are three clustering methods available for use:

Heirarchical Clustering Method 1 - which merges the "closest" pair of clusters measuring the distance between two clusters A and B by the mean of the distance between the nodes of A and the nodes of B. That is,

$$d(A,B) = \frac{1}{N_A N_B} \Sigma = (a,b) \ .$$

where N_A and N_B represent the cardinality of A and B respectively, the summation is over all the elements $a \varepsilon A$ and $b \varepsilon B$.

Heirarchical Clustering Method 2 - which merges the pairs of clusters which lead to a minimum mean of the distance between all pairs of nodes in the cluster resulting from the merge. That is,

$$\text{minimize } X = \frac{1}{N_A^2} \Sigma_s \ (a,a^1)$$

where N_a = the cardinality of the set merger A

$a, a^1 \in A$.

the summation is over all pairs of nodes.

Heirarchical Clustering Method 3 - which merges the two clusters A and B that lead to a minimization of the parameter y.

$$y = \frac{1}{N_A N_B} \sum_s (a,a^1) - \frac{1}{N_A} \sum_s (a,a^1) - \frac{1}{N_B} \sum_s (b,b^1)$$

where the first summation is overall pairs of nodes in A and B, the second overall pairs in A, and the third overall pairs in B. This method evaluates each clustering step as a function of the partition parameters before and after the clustering and, therefore, tends to produce the best partitions; i.e., those with the highest measure.

Additional facilities exist in the software package to perform a number of additions and deletions from the graphs and to print out the results.

The analysis package was designed to recognize a single decomposition problem at a time. Therefore, the package always deals with a current graph; that is, the fundamental working entity that the package is currently working on. In addition, Steps 1 through 4 must be accomplished prior to invoking Step 5. Any change in the order will generate a system error.

The use of the decomposition methodology is presented in Appendices E and H for the first and second iteration of the methodology.

CHAPTER III

SAMPLE OPERATING SYSTEM

The Sample Operating System, developed by Professor Stuart E. Madnick and John J. Donovan[20] as a pedagogical tool to illustrate the basic functions of a computer operating system, was selected as the design problem for analysis. The selection of an existing, well-documented system was dictated by a desire to insulate the decomposition analysis from any problems associated with poor needs analysis.

The Sample Operating System is composed of all the functions normally associated with a computer operating system; however, due to its strictly pedagogical nature, it has some unique features as well. It was conceptually convenient to break the system down with its functional areas for descriptive and requirements definition purposes. The following discussion will highlight the general functions of the Sample Operating System.

3.1 General Characteristics of a Large Scale Computer Operating System

In most general terms any operating system is a group of programs within a computer system which manage the hardware/software resources of the computer, and thereby serve as the interface between the user's programs and the resources of the computer.

---

[20] *Madnick and Donovan, p.381.*

Madnick and Donovan have defined the following entities within a computer system:

user: one who desires to utilize the computer resources.

job: any collection of activities needed to complete the work desired by the user. A job may be further subdivided into steps, tasks, or processes.

job step: units of work which must be done sequentially; namely, compile, load, and execute.

task: a program or job subdivision which is the basic unit or work for the operating system.

process: a complete sequence of instructions that are functionally/computationally independent of other processes.

The normal resource management functions of the operating system may be generalized into the following four functions:

- Keep track of a resource;
- Enforce a policy that determines which user gets a given resource, especially to resolve conflicts arising from competition for the same resource.
- Allocate a resource.
- Reclaim a resource.

The functional resources of any large-scale computer system may be described as follows:

- Memory Management Functions
- Processor Management Functions

. Device Management Functions

. Information Management Functions

The definitions, management functions, and resources previously defined will be adopted in order to fully describe the characteristics of the Sample Operating System.

3.2 Sample Operating System Description

The Sample Operating System, as described by Madnick and Donovan, is conceptually designed around a process, recognizing that a process is the smallest computational entity and, therefore, has certain requirements necessary for its support. Thus, the Sample Operating System implements a basic system nucleus required for a complete system; yet it does not include other capabilities such as language processors or utility programs.

3.2.1 EXTENDED MACHINE CONCEPT:

At the most basic level, a computer processes only specific hardware instructions; such as ADD and LOAD. In the Sample Operating System it was necessary to provide the basic functions for process support as additional hardware - like instructions at a level above the basic machine instructions. These instructions are called extended instructions and are implemented by means of the Supervisor Call Instruction. These instructions are conceptually similar to subroutine calls which enable the user to perform certain resource management functions at a higher level than the bare machine. For example, SVC 'H' is used to halt a job and

signal the supervisor process. Each extended machine instruction calls a handler routine and may be user callable. The basic hardware instructions of the machine combined with the operating system provided "supervisor instructions" comprise the instruction set of the extended machine. The Kernel of this operating system runs on the bare machine, the user's programs run on the extended machine. Figure 3.1 represents the extended machine concept.

3.2.2 HEIRARCHICAL MACHINE STRUCTURE:

Since the Sample Operating System was intended to be primarily a pedagogical tool, a layered system architecture called heirarchical operating system structure was selected as the basis for system design. Basically, the methodology allows the segregation of major functions of the operating system into a heirarchy of capabilities. Its major advantages include:

- It is a powerful means of proving the correctness and maintaining the operational integrity of the operating system.
- Lower layers of the system provide services to higher layers only via well-defined interfaces.
- The modular structure enables the easy identification of the major functions of the operating system.

In order to implement the heirarchical concept in conjunction with the extended machine concept, it was necessary to define the following:

- Certain key functions needed by many of the system

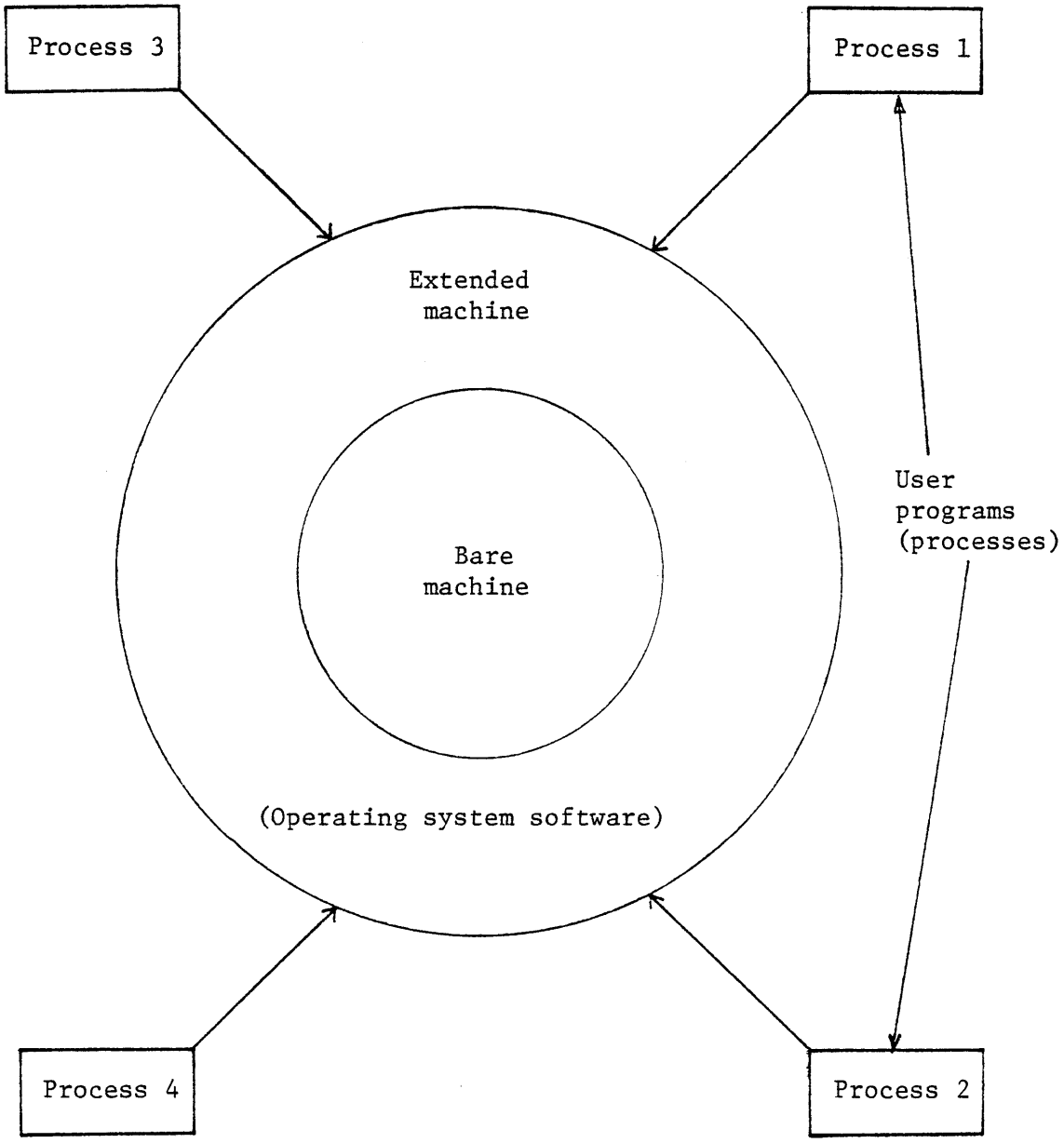


FIGURE 3.1 Extended Machine Concept of a Generalized Operating System

modules could be separated into an "inner extended machine".

. Certain modules, which were not utilized as key functions yet still operating system modules, could be separated out and run on the extended machine in essentially the same way as a user's process.

It is, therefore, apparant that each module of the operating system must be identified as running either in the inner extended machine, the outer extended machine, or as a process.

For further clarification, Madnick and Donovan have generalized the inner/outer extended machine concept into levels of the extended machine, and all operating system functions that run as processes can interrelate and are generalized into layers of processes. The Kernel of the operating system then is all these modules that reside in the extended machine and, therefore, do not include operating system processes.

For purposes of the Sample Operating System design, the basic functions of the operating system have been placed in the Kernel, and as many tasks of the operating system as possible have been placed into separate system processes. In this heirarchical implementation, we impose the following restriction: a given level is allowed to call upon the services of lower levels only; i.e., those levels closer to the bare machine. This restriction requires well-defined

interfaces and synchronization schemes throughout the Sample Operating System.

Figure 3.2 graphically portrays the heirarchical operating system structure.

The three concepts implemented for the design of the Sample Operating System design (that is, process focus, extended machine concept, and heirarchical structure) have evolved into a system with the following features:

- process synchronization semaphore, used extensively for resource allocation synchronization;
- message system for interprocess communication;
- five levels and layers of the Sample Operating System:

  Levels - Process Management, lower module
           Memory Management module
           Process Management, upper module

  Layers - Device Management module
           Supervisor Process module

A brief description of the function of the levels and layers is provided to further clarify the structure of the Sample Operating System.

3.2.3 PROCESS MANAGEMENT, LOWER MODULE:

This module enables the Sample Operating System to support multiprogramming and the basic system primitive operations required for interprocess synchronization.

The basic primitives as previously described, are the so-called P-V operations. Both operations act on a semaphore

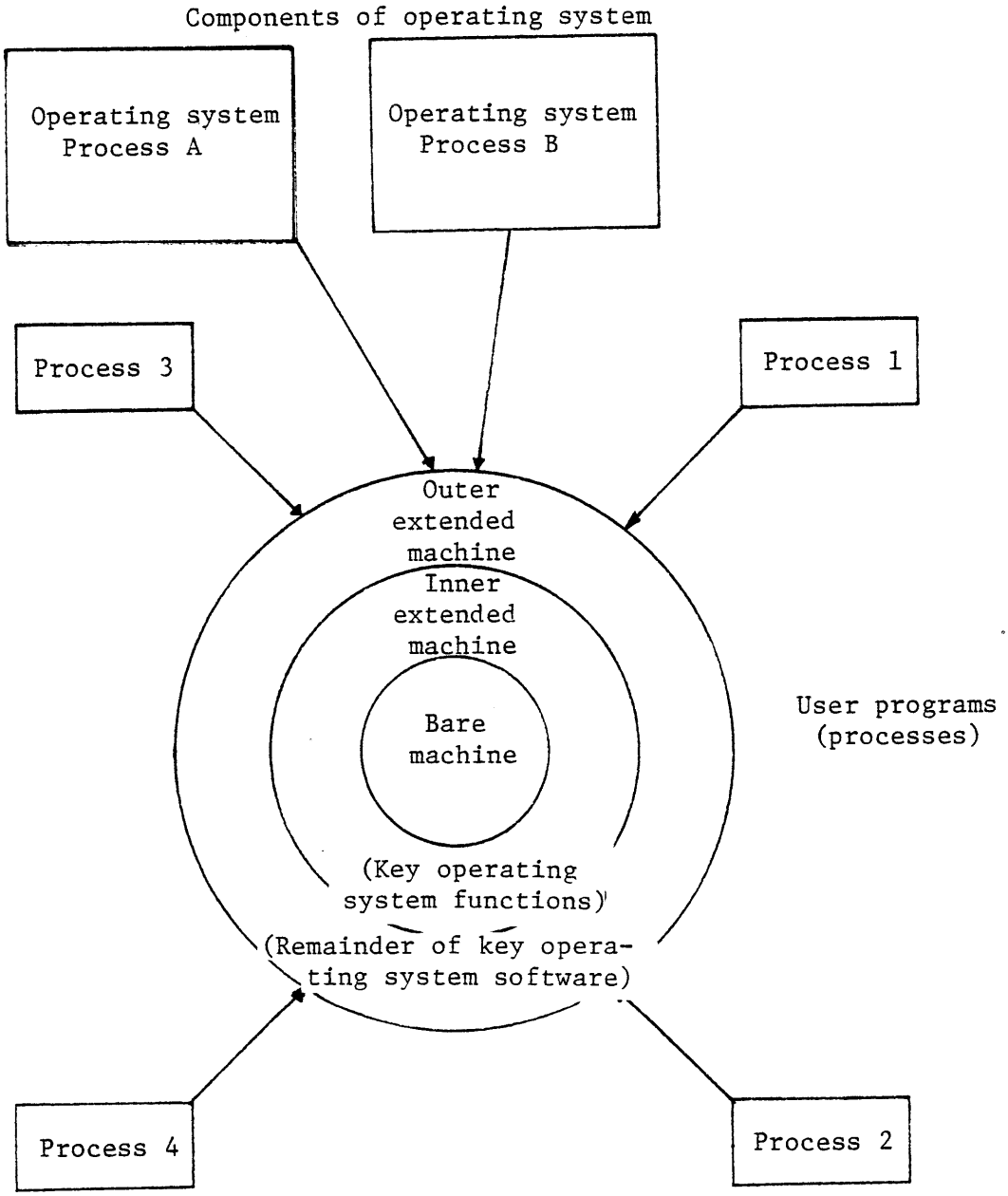


FIGURE 3.2 Heirarchical Design Structure of a Generalized Operating System

which has an associated integer value and serves as a counting lock as follows:

P-operation: IF Semaphore Value > 0 then
Value = Value-1
IF Semaphore Value ≤ 0 then
Value = Value-1 and the process is ineligible to allocate the given resource.

V-operation: IF Semaphore Value > 0 then
Value = Value+1 and no process is ineligible to allocate the given resource.
IF Semaphore Value ≤ 0 then
Value = Value+1 and there is a process waiting to allocate given resource.

Since there is a semaphore associated with each resource the P-V operations can serve as a lock where semaphore value initially = 1. By requiring a P-operating before accessing and a V-operating after completion, the integrity of the resource is ensured.

3.2.4 MEMORY MANAGEMENT MODULE:

This module performs the operations necessary for dynamic allocation and freeing of memory for job partition allocation and for allocating space for use by the operating system.

3.2.5 PROCESS MANAGEMENT, UPPER MODULE:

This module provides the routines for the control of processes; i.e., process creation and deletion. The module also provides for interprocess communication with buffered messages. This module was split from the Process Management, lower module since it depends on the functions of memory management to allocate or free memory areas to store system information concerning each process and to provide temporary buffers to store interprocess communication messages.

3.2.6 DEVICE MANAGEMENT MODULE:

This module runs as a separate process; hence, it is considered a layer of the operating system. There is one device management module per device which provides the routines necessary to issue the appropriate input/output commands to external devices. This module depends heavily upon the interprocess communication message facility to issue I/O and to interpret the status information for a return message. Device management for this service is simple since all devices are dedicated and consist only of card readers and line printers.

3.2.7 SUPERVISOR MODULE:

The supervisor module, also runs as a separate process of the Sample Operating System; specifically, one per job stream. The supervisor provides interfacing for all the routines needed to run a job. In particular, the supervisor process is responsible for coordinating the following:

1) Reads in a job stream.
2) Allocates a partition of memory for each job in sequence.
3) Creates and starts the appropriate device management process.
4) Loads the user's object deck into the partition.
5) Creates and starts a process in the given partition. Since the supervisor process is not needed until the user's job ends, it stops running and waits for a message signalling completion of the user's job.
6) Finally, when completion is signalled, the supervisor cleans up by destroying the allocated partition of memory, and goes to the next job input stream.

3.2.8 USER'S PROGRAMS AND PROCESSES:

Initially the Sample Operating System creates a single process for each job; however, the user is free to create additional processes to run in parallel. The user's job runs in problem state with non-zero protection key assigned; thereby, restricting user access (to privileged instructions and memory areas external to the user's allocated partition).

The nucleus routines, such as P-V operations, are restricted from the user and cannot be accessed by the user's job. However, the interprocess communication message facility is available to the user and can be utilized for interprocess synchronization of user processes.

3.3 Summary

The basic design philosophy of the Sample Operating System and a functional description of the major modules has been presented as the system is currently configured. The next two chapters will first define the requirements as they exist for the Sample Operating System; and second, assess the interrelationships among these requirements.

CHAPTER IV

REQUIREMENTS DEFINITION

The purpose of this chapter is to describe the methodology of the requirements definition for the Sample Operating System. The requirements were defined from a description of the Sample Operating System and program listings as provided by Madnick and Donovan, subject to certain guidelines established by Andreu to insure that as much as possible the requirements are defined in a clear, correct, and concise manner.

It must be stated at the outset that requirements definition was the most time-consuming portion of this analysis. The definition phase was repetitively iterated as requirements were defined more clearly, make less ambiguous, corrected, discarded, combined, separated, and new requirements added continuously. Since one can become completely embroiled in the problem, it is essential that the requirements be reviewed periodically by an interested third party.

The initial methodology for requirements definition was proposed by Andreu[23] and was based on his experience with the problem. Andreu began with a set of requirements for a database management system and sought to refine those requirements as they existed. For the Sample Operating System, however, no such precise list of requirements

---

[23] *Raphael Andreu, "An Exercise in Software Design: From Requirements to Design Problem Structure", MIT Sloan School unpublished report (June, 1977), pp.3-15.*

existed. Therefore, it was necessary to draft a set of requirements from a textual description of what the system does using program listings to resolve unclear issues. Consequently, the Andreu methodology was supplemented with additional guidelines based on these experiences in defining requirements.

The following section will define the methodology and by way of example, demonstrate what constitutes good or poor definitions of requirements.

4.1 Requirement Definition Methodology

4.1.1 DEFINITION CLARITY:

Requirements should be stated clearly and concisely. It is conceptually difficult to deal with requirements which are verbose or deal with more than one specific issue. In addition, requirements interdependencies are assessed on a one-for-one basis. Therefore, each requirement for the Sample Operating System was limited to a single sentence, covering only one issue. The requirements for the Sample Operating System are presented in Appendix G, and each requirement statement is followed by a definition of the requirement and a statement of implications of that requirement for the design of the system. This format was valuable for it enabled a single sentence requirement definition statement, yet it facilitated further amplification of the design requirement which was very helpful in the inter-

dependency assessment phase.

4.1.2 SCOPE OF DEFINITION:

Requirements must not be stated in very general terms, or in terms dealing with issues beyond the scope of design.

For example: The operating system must be capable of maintaining memory resources. This is a general statement characteristic of all operating systems by definition. This statement does nothing to further define characteristics of the Sample Operating System. In addition, no requirements were defined for the following functions: system reliability, documentation, and system security, simply because none of these issues were addressed as needs of the Sample Operating System, and, therefore, were beyond the scope of the design.

4.1.3 IMPLEMENTATION INDEPENDENCE:

As stated by Andreu and Madnick[24] requirements should not specify an implementation scheme that may be used in the design of the system. Clearly a requirement which specifies how a requirement is to be implemented biases the design process. Specifically, such a procedure precludes the design from considering alternative solutions to a given design problem. The specific implementation scheme may be appropriate within its limited realm of consideration, but may not be optimal in the context of the overall design problem. Finally, any implementation scheme, specified

[24] *Andreu and Madnick, p.42.*

"a priori" inevitably affects other requirements in other stages of the design process. The criteria for requirements definition is simply that the requirement definition must state only what is to be done and not how.

For example: the statement, "A process can issue a call to read the text and name of the message sender"; this violates the guidelines since the statement defines the means of implementation. The requirement focuses on how a process recognizes the text and name of a message sender, rather than what was intended.

Therefore, the requirement was re-written as: "The receiving process may read the name and text from the originator".

4.1.4 SYSTEM STRUCTURE INDEPENDENCE:

Any definition of requirements should avoid biases toward pre-established assumptions about the structure of the final design according to Andreu. [25]

This guideline is very subtle in its application, and represented the most difficult guideline to fulfill since the Sample Operating System had been designed and was described in terms of its final structure. Conceptually, anyone seeking to define a non-trivial system must organize his thoughts in some manner to avoid total confusion. The most logical framework for organization is in terms of the functional requirements of the system. The most general

---

[25] *Andreu, "An Exercise in Software Design", p.46.*

functional requirements for an operating system focus upon the role as a resource manager of memory, processors, devices, and files. Therefore, one tends to define requirements in the framework, and the trivial decomposition solution would define four distinct subproblems which correspond to those functional requirements. Clearly, such a solution would offer no new insights into the structure of the design problem.

> For example, the requirement "This operating system must be pedagogical and modularly structured", was considered to violate the guideline. The Sample Operating System was designed to be pedagogical. Although it is generally recognized that the most effective method of achieving pedagogical clarity is through modular design, such a statement is constraining upon the system designers and, therefore, was re-written as follows: "The operating system must be designed as a pedagogical tool". The resulting decomposition of the design requirements should indicate what degree of modularity was achieved in the actual design.

4.1.5 INDEPENDENCE AMONG REQUIREMENTS:

This guideline implies that all requirements must be semantically independent; namely, that redundant requirements must be eliminated.

> For example: the two requirements "Basic system primitives and certain routines are restricted from the

user, the use of which will generate an error" and "The operating system shall protect itself from the use of supervisor routines by the user" are redundant; the former being implied by the latter. The former requirement was, therefore, eliminated.

4.1.6 SIMPLICITY:

Each requirement should address one well-defined capability that the final design is to demonstrate. The purpose of the decomposition methodology is to assess inter-dependencies among individual requirements, and to group similar requirements together. Therefore, grouping requirements by definition masks the decomposition.

Many requirements were originally defined with multiple capabilities. It was necessary, therefore, to separate each capability with a separate requirement.

For example: the following requirement, originally written as a single requirement, was separated into four distinct requirements: "A process synchronization mechanism must be provided:

1) to serve as a lock on a database.
2) for timing of synchronous processes.
3) for synchronization of the message facility.
4) to lock a device.

4.1.7 NO STAND-ALONE REQUIREMENTS:

Requirements which are only remotely concerned with the final design should be avoided; for example, features which

may be added to an operational system at a later time illustrate this point.

For example: The requirement, "The supervisor process must be modularized so that improvements to the system can be easily accomplished", satisfies this guideline. The requirement indicates that improvements to the system are anticipated, yet it does not limit the requirement by specifying what improvements will be made later.

4.1.8 PLAUSABILITY:

Naturally, a requirement should avoid the impossible; therefore, statements shall be eliminated which imply requirements which are:

. not available with current technology;

. in violation of fundamental physical requirements;

. clearly violating other requirements.

For example: The requirement, "The input/output devices are limited to card readers for input job streams and line printers for output", implies that no spooling system is available. This in turn dictates that job scheduling be accomplished on a first-come, first-served basis.

Initially, it was felt that such non-capabilities (i.e., lack of spooling capability and lack of file system) should be explicitly stated as a requirement rather than inferred. However, the assessment of

requirements for facilities which do not exist would have been difficult to accomplish. Therefore, the lack of a certain capability was not addressed in requirements definition.

In addition to the previous guidelines established by Andreu, the following additional guidelines were developed.

4.1.9 SEMANTIC INTERPRETATION:

The requirements should be defined in a manner that limits semantic interpretation. This guideline resulted from an examination of the various "problem statement languages" which are currently being investigated. Stating requirements formally, in a problem language statement, could not only reduce the ambiguity of the requirement, but aid in the interdependency assessment phase. Although no specific language was employed for requirement definition, the basic structure and intent of a rigorous definition language was used to define the requirements; specifically, the requirements were defined as follows:

1. Utilize generally understood terminology; for example, "reclaim memory resources" versus "garbage collection". Reference to functions was by formal terminology job scheduler.
2. Avoid terms which are not commital; for example, "operating system must supply ... " instead of, "operating system may or should be capable of..."
3. Recognize the distinction between existence

statements and performance statements. For example, the requirement, "The process scheduler must time-slice CPU usage among ready processes to achieve multi-programming", implies the existence of some time quantum.

The actual performance requirement is stated separately as "A process must be blocked, and control released to the process scheduler when a time quantum of 50 ms is exceeded".

Limitations implied by existence statements must be made explicit in a performance statement.

4.1.10 SCOPE OF REQUIREMENT DEFINITION:

The requirements must be defined at the same level of scope. The customer, in this case being the person for whom a system is designed, must have a macro-level objective which the system must be designed to satisfy. P. Mandel and C. Chryssostomidis state:

> "The objective of most problems that man is capable of conceiving or is interested in solving is that of choosing the course of action which subject to prevailing constraints, optimizes the 'well being' of all concerned."[26]

The following concepts have been identified at the outset of the design process.

. An objective function to be optimized for the design process.

---

[26] *Mandel and Chryssostomidis, p.85.*

. Prevailing constraints, which impose limitations upon the designer.

. Requirements flow directly from the customer in response to the overall objective of the system design.

The objective function usually takes the form of a multi-matrical expression to be optimized and for most large-scale computer systems, consists of the maximization of throughput or minimization of response time.

The objective function of the Sample Operating System is pedagogical clarity and, therefore, it is very difficult to state that the objective function has not been fulfilled. For the purposes of the design of the Sample Operating System, a design philosophy has been identified which defines the design criteria for the system on a macro-level. The requirements that comprise the design philosophy influence each of the remaining requirements and, therefore, were not incorporated into the assessment process.

The design constraints usually serve to limit the permissable range of solutions of the problem. The constraints, then, impose limitations on the designer which affect the global design problem. In the Sample Operating System, certain hardware constraints were imposed "a priori" upon the design problem. Specifically, the operating system must be designed to run on IBM/360 hardware. The implications of this constraint affect certain basic functions of the operating system. Since the design constraints have been specified "a priori", such constraints

have been separated from the remaining system requirements and were not incorporated into the assessment process since the constraints represent limitations on system design.

Finally, the system requirements are defined in direct response to the customer's objectives. The system level requirements must be defined at a level below the most general of system level statements, yet remain above the level which begins to limit the options of the designer.

4.2 Summary

The requirements for the Sample Operating System were defined in two iterations. The preliminary set of requirements was defined initially and are presented in Appendix C. The second or final requirements set was defined after the initial application of the decomposition methodology and are presented as Appendix G.

CHAPTER V

INTERDEPENDENCY ASSESSMENT METHODOLOGY

The purpose of this chapter is to establish the guidelines that were used for the assessment of interdependencies between pairs of requirements. The assessment was conducted on a pair-wise basis according to the following definition of interdependence.

Two requirements are termed interdependent of the design decisions made with respect to one requirement constraint, or influence the definition of the second requirement. Thus, the interdependent relationship between two requirements can be viewed in two ways:

Supportive: in the sense that the two requirements are compatible; meeting one requirement will help to satisfy the other as well.

Conflicting: the interdependency is such that some trade-offs must be established between the two requirements in the later stages of the design process.

The result of the assessment process is the decomposition of the global system requirements into a number of sub-problems, which ideally will be a collection of highly dependent requirements.

5.1 Interdependency Assessment Methodology

The methodology for interdependency assessment proposed

by Andreu consists of a pair-wise assessment of the inter-dependencies between requirements by the generation of "conceptual models" within whose context the assessment can be made. The purpose of generating a conceptual model for the assessment process is to have a specific mental framework so that the process is consistent and conceptually rigorous. The following guidelines have been proposed by Andreu, for the generation of conceptual models:

- Scan the requirements in order to develop loose conceptual models of the system.
- Supportive requirements can be identified by visualizing a conceptual model in which a possible implementation would allow for common processing in the final system in order to meet the two requirements involved.
- Supportive requirements can be identified in cases where two distinct requirements call for similar functions to be performed in different circumstances in the final design.
- Conflicting requirements can be identified by:
  - searching for deadlocks
  - identifying when a given requirement imposes limitations or constraints in other requirements.
  - identifying the need for "symmetric" processing; that is, additional processing to meet a given requirement is necessary.

The procedure of assessing all the interdependencies for a large system can become burdensom. Therefore, Andreu, has proposed a set of procedural guidelines, based on his experiences, which were helpful in avoiding some of the pitfalls of this time consuming process.

- Establish an order for the assessment to be made.
- Write down conceptual models as they occur.
- Avoid going backwards to renew an assessment made previously. Finish the assessment process and then return.
- If the assessment of similar requirements becomes confusing change to a different set.
- If no conceptual models are apparent skip the assessment until one is available.
- If one feels uncertain or lacks confidence in the assessment process; stop, and come back to it later.
- A second assessment pass is useful, since it enables one to employ new conceptual models and to review the assessments which have been previously established.

In addition to the guidelines established by Andreu the following additional guidelines were identified.

In the case where the assessor's experience is lacking, the results of the assessment process should be reviewed by another interested third party in order to:

- verify the conceptual model.
- verify the nature of the interdependency.
- verify the resulting adjacency matrix.

In order to define a more rigorous conceptual model for the assessment process, the following assessment template was imposed upon each assessment:

1. Does the first requirement conflict with the implementation of second requirement, causing deadlocks, symmetric processing, or imposing limitations? For example, the first requirement, "System resources must be allocated to a job prior to being allocated a processor". The user resources (i.e., processor) are allocated at the user level, and the system resources are allocated at job level, which requires symmetric processing.
2. Does the first requirement support the implementation of the second requirement by common processing: or do they call for similar functions to be performed in different circumstances? For example, the first requirement, "System resources must be allocated to a job prior to the job being made eligible to run" is supported by "the supervisor process must schedule jobs and prepare the jobs for execution". In this case, the supervisor process controls the allocation of resources for each job, preparing them for execution.

5.2 Summary

A pair-wise interdependency assessment was conducted

according to the guidelines previously established for each requirement defined. Since an interdependency is symmetric, in the sense that an interdependency between requirement #8 and #30 implies an interdependency between #30 and #8. Therefore, each requirement was assessed with the requirements that followed it. At the time of assessment, an indication was made whether the interdependency was supportive or conflicting, and a brief statement of the rationale for the interdependency was made.

As was the case for requirements definition, the interdependency assessment process took place in two iterations. The preliminary interdependency assessment is presented in Appendix D, and the final interdependency assessment is presented in Appendix H.

CHAPTER VI

FIRST ITERATION OF THE DESIGN PROBLEM

The interdependencies assessed between pairs of requirements were formed into an adjacency matrix and input into the software analysis package developed by Andreu. This chapter will present an analysis and discussion of the resulting problem structure. The analysis and discussion will consist of the following sections:

- An analysis of the resulting problem structure for the first iteration.
- Discussion of the main subproblems.
- Discussion of the subproblems generated by a second decomposition.
- Relationships among the main subproblems.
- Motivation for a second iteration.

6.1 Analysis of Problem Structures

A total of sixty-five requirements were input with the software analysis package for decomposition. Appendix E presents a copy of the output of the analysis and will be frequently referred to during the analysis. The analysis provided as follows:

First, the data, in the form of links (interdependencies) between nodes (requirements) was verified by checking that all assessed interdependencies were, in fact, present. This was accomplished using the "NOLK" command.

Second, all isolated nodes: those nodes with no interconnecting links were identified. As a procedural convenience, the initial graph was input with several extra nodes. It is possible to delete nodes during the analysis, but no new nodes may be added. Therefore, in order to enter new nodes, one must redefine the entire graph. To avoid this time-consuming process, extra nodes were padded into the graph, and the graph saved with the padded nodes. A working copy of the graph was generated by identifying and deleting isolated nodes and then saving the temporary working copy.

6.1.1 MAIN SUBPROBLEMS:

The adjacency matrix was decomposed utilizing the steps outlined in section 2.3 and the results including a heirarchical tree are presented in Appendix E. The design requirements decomposed into six clusters or main sub-problems (abbreviated at MS), of twenty to four members each. The decomposition was generated using the so-called heirarchical clustering method -3, the following evaluation parameters resulted:

Strength: 1.9864

Coupling: .8674

Measure: 1.1119

Strength was defined as a normalized evaluation of subset internal coherence; that is, how tightly coupled the nodes in a given subgraph are. Coupling is defined as an

evaluation of the extent to which two subgraphs are interdependent. Measure equals strength minus coupling. The evaluation parameters obtained are important in a relative sense, since there is no absolute value of any parameter which indicates a good decomposition. A comparison of strength and coupling was made to make some statements of the decomposition. A coupling/strength ratio = .43 was determined indicating that the coupling between subgraphs was nearly half the measure of internal coherence. This indicates that the subgraphs are internally coherent (high strength value) and still have a reasonable degree of coupling. Whereas a small coupling value would indicate that the subgraphs were relatively decoupled.

In order to further investigate the coupling evaluation Andreu[27] suggests a second decomposition in which each main subproblem is treated as an entire graph and decomposed into subproblems. Since the coupling parameter increases as subproblems are defined, and strength remains constant, the decomposition of the main subproblems into subproblems decreases the overall partition measure. If the coupling parameter between main subproblems is low originally, the main subproblems are fairly disjoint and a second decomposition should be investigated. In the ideal case, a main subproblem may exhibit such internal coherence (high strength) that it does not decompose into subproblems. This

[27] *Andreu, "A systematic Approach to the Design and Structuring of Complex Software Systems", p.277.*

became the motivation for a second decomposition; that is, a main subproblem was considered well-defined if no decomposition resulted. Therefore, a second decomposition of each main subproblem was performed; the results are presented in Appendix E.

6.2.2 SECOND DECOMPOSITION STEP:

Each main subproblem resulting from the original decomposition was individually decomposed as follows:

1. A separate graph was defined for each main subproblem by eliminating all nodes external to the MS under analysis. This was accomplished using the "DEMO" command. The nodes of the current are renumbered at this point, which required rather awkward collating schemes to keep the original set of requirements synchronized with each new subgraph.
2. Each MS was decomposed according to the methodology of section 2.3.

   Of the six MS originally defined only two, MS 2 and MS 3 further decomposed.

6.1.3 ANALYSIS METHODOLOGY.

In order to analyze the structure of the design problem implied by the decomposition technique, it is useful to investigate the following entities:

. Elements: requirements contained within a given subset.

. External Interdependencies: links that exist among the elements of different subsets. The command "PRLK" was useful in identifying the external links.

Recalling the core set identification process, certain nodes were identified as seed nodes, about which other nodes were clustered. In order to identify the main focus of each cluster, one examines the requirements involved in the largest number of interdependencies; i.e., the node in an MS with the largest number of links. It is assumed that this is the seed node for the MS and is, therefore, related to the main focus of that MS. It is also noted that a given main subproblem may have several nodes with nearly the same number of links which could be called equivalent seed nodes. A closer examination of these nodes must be undertaken to determine the nature of such a main subproblem.

6.2 Main Subproblems

The problem structure resulting from the application of the decomposition methodology is presented in Figure 6.1. The problem structure was interpreted as composed of the main subproblems, depicted as blocks in Figure 6.1 and the subproblems, generated by a second decomposition, depicted as circles within the parent MS. The interdependencies among the elements of different subproblems were generalized into interfaces which are required in design between subproblems.

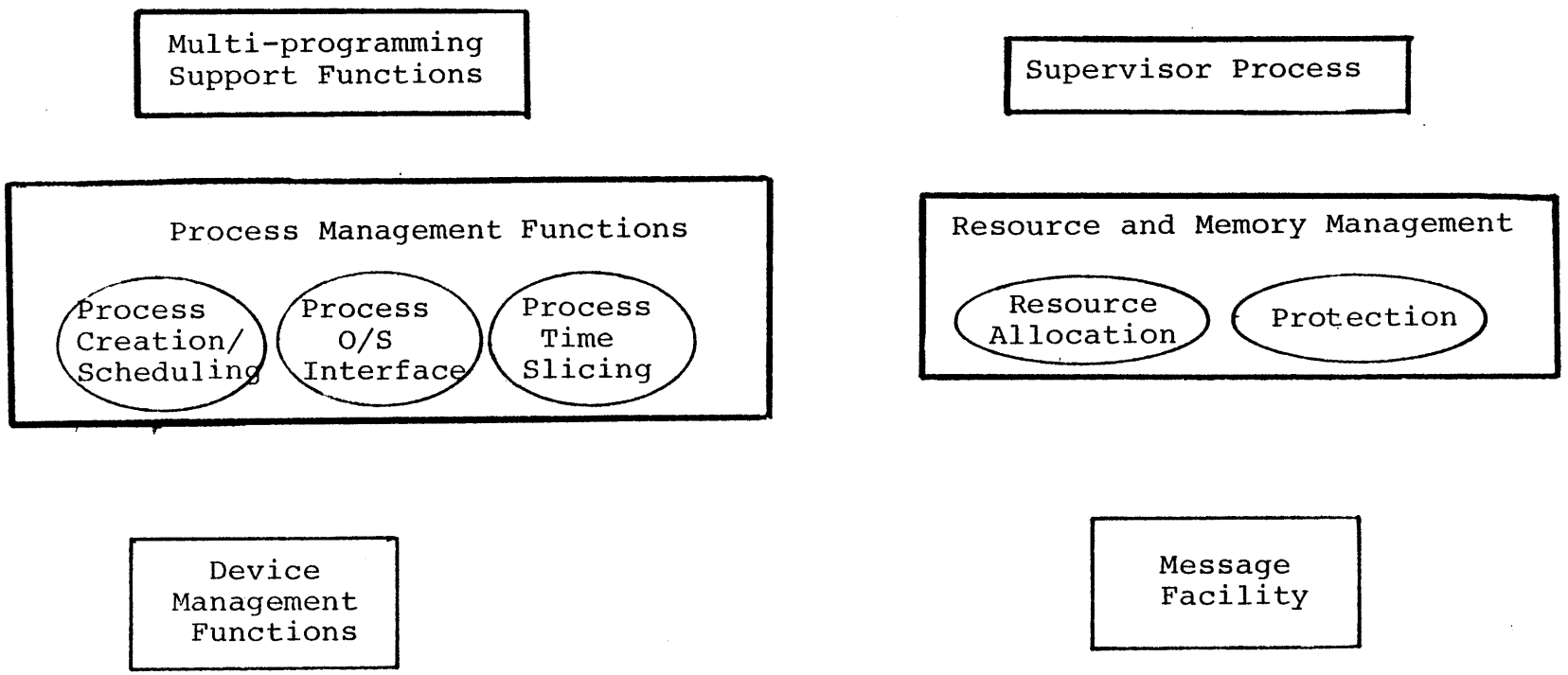


FIGURE 6.1 Problem Structure Implied by the First Iteration of the Decomposition Methodology

The interpretation of main subproblems should be intuitive if the MS are well-defined. Since each MS was built around a certain seed node, interpretation became a problem of identifying the node and understanding how the other member nodes were built around it.

The more interesting part of the interpretation was to identify the counter-intuitive or non-obvious results. These results suggested a structure of the design that was not apparent at the outset or perhaps errors in the analysis. In either case, it was the identification of non-intuitive results that represent the value of the process.

The following discussion will highlight the general and specific characteristics of the design structure indicated by the decomposition methodology.

The decomposition methodology generated six main subproblems at the end of the first decomposition. The six main subproblems have been generalized into the following groups:

1) Multi-programming support functions
2) Process management functions
3) Resource and memory management functions
4) Supervisor process functions
5) Device management functions
6) Message facility

The requirements statement are decomposed into these six main subproblems and are presented in Appendix F.

The following discussion will highlight the characteristics and discrepancies discovered in each main subproblem.

6.2.1 MULTI-PROGRAMMING SUPPORT FUNCTIONS:

The requirements which decomposed into the multi-programming support main subproblems were all concerned with the features and facilities that must be provided by the operating system in a multi-programming environment. These features include:

- a multi-programming environment must exist
- job scheduling
- re-entrant and pure code
- supervisor process support
- synchronization techniques
- protection among jobs

The seed node was requirement 5, "The operating system must provide for a multi-programming environment".

However, it must be noted that requirement 43, "P-V mechanisms must be provided", had a larger number of links than requirement 5. The P-V mechanism provides the basic multi-programming support by synchronizing operating system functions, but it represents a specific tool rather than a focus for the main subproblems. In addition, the P-V mechanism is used for four distinct functions. Therefore, it was decided to further investigate this requirement and attempt to redefine it.

Some of the requirements have a dual function, and,

therefore, decomposed into this MS, which at first appeared counter-intuitive; for instance, the requirement, "Device handler routines must support multiple job streams from card readers". Intuitively, one would have expected this requirement to fall squarely in the device management MS. However, the issue is the requirement to support multi-programming by providing input from multiple job streams. It is expected that such dual requirements will also have interfaces linkages between the two MS in which they seem to belong. This will be investigated later.

This MS did not decompose upon the second decomposition.

6.2.2 PROCESS MANAGEMENT:

The smallest computation entity defined by the operating system is the process; therefore, the operating system must recognize this feature and provide the necessary functions for support of the process. The requirements which decomposed into this MS constitute the largest set of requirements in a given MS and deal with those basic functions required for process support. These features include:

- Process creation/destruction
- Allocation/De-allocation of a processor to a process
- Time-slicing
- Extended machine instruction environment
- Process Scheduling

The seed node for the MS was requirement 6, "The operating

system must be process oriented", which is indeed the focus of the MS.

Requirement 44, "An interrupt handler must be provided" had nearly the same number of linkages as requirement 6. Time slicing CPU usage requires an interrupt handler; however, this is not the only function of the interrupt handler. This requirement presented problems later in the interface analysis. Therefore, it was decided to redefine the requirement by separating it into a number of distinct interrupt handlers.

The decomposition included one counter-intuitive requirement.

"Message facility must be accessible to all processes." It was expected that this requirement would decompose in the message facility MS. Upon examination of the interdependency assessment and the conceptual models used for this requirement, it was noted that the requirement is defined as being interrelated with three requirements in the MS and only one in the message facility MS. The issue there is one of process accessibility to the message facility, which is the primary means for interprocess communication. Therefore, this requirement related more closely to process support than to the message facility.

This main subproblem decomposed into three subproblems in the second decomposition.

6.2.3 RESOURCE AND MEMORY MANAGEMENT FUNCTIONS:

This main subproblem is composed of requirements which deal with resource allocation in general, and memory management in particular. The functions concerning resource allocated include:

- Resources are requested through the supervisor.
- Information tables are utilized to monitor resource allocation.
- Operating system can dynamically allocate memory for its own use.

The requirements dealing with memory management functions include:

- Operating system must allocate memory.
- The mechanisms by which memory is allocated, protected, and reclaimed.

This main subproblem essentially has three nodes of similar linkage value. The three requirements all deal in general terms with resource and memory allocation, but no clear definition is apparent. It can be argued that memory management is a subset of the general resource management function of the operating system. It is noted that this MS has the largest number of interfacing linkages with other main subproblems. This was expected since the members of the MS seem to cover such a broad area of responsibility.

This requirement decomposed into two subproblems in the second decomposition.

6.2.4 SUPERVISOR PROCESS:

The requirements which decomposed into this main subproblem all deal with the functions of the supervisor process. The supervisor process is that process which schedules jobs and prepares them for execution. Many of the functions normally performed by the supervisor were decomposed into the multi-programming support main subproblems, particularly the job scheduling function. The supervisor process is a subset of the functions required for multi-programming support and, therefore, this result seems to make sense. It is also noted that there are a large number of linkages between the supervisor process main subproblem and the multi-programming support module.

The existence of a supervisor process module distinct from the multi-programming support module is considered a significant insight into the problem structure. The design problem structure dictates that both the supervisor process and multi-programming support main subproblems are distinctly separate at the same level of comparison and deserve equal design concern.

This module did not decompose on the second decomposition.

6.2.5 DEVICE MANAGEMENT FUNCTIONS:

The members of this module clearly are concerned with the functions required for device management. These functions include:

. A device management routine.

. Devices and protocols required to support multi-programming.

The seed node for this main subproblem was requirement 36. "The operating system must supply a device management routine." This main subproblem decomposed very clearly; that is, it had the highest strength value for all the main subproblems which did not decompose on the second decomposition.

6.2.6 MESSAGE FACILITY:

All of the requirements in the module directly address the needs for a message facility, which is an interprocess communication technique in the Sample Operating System which enables user processes to communicate and synchronize execution.

The seed node for this main subproblem was requirement 46. "A message facility has many requirements since there are many features defined for use of the facility." Although the message facility may seem to be a relatively less important function of the operating system, the decomposition methodology implies that it constitutes a complete main subproblem. It may be that one may generate an entire main subproblem just by defining a large number of requirements for a relatively insignificant feature; or conversely, this facility may be of greater significance to the operating system than previously anticipated.

This module did not decompose in the second decomposition.

6.3 Subproblems Generated in a Second Decomposition

A second decomposition was conducted as described in section 6.1 and resulted in the decomposition of MS 2 and MS 3 into three and two subproblems respectively. The term subproblem will be used to describe the clusters which resulted from a second decomposition of the main subproblem.

6.3.1 MS 2 - PROCESS MANAGEMENT FUNCTIONS:

MS 2 decomposed into three subproblems as follows:

1) MS 2A - Subproblem A: Process Creation and Scheduling

   This subproblem is designated MS 2A. All of the requirements in the subproblem were concerned with process creation and scheduling. These functions included such features as initial process creation, process identification, process blockage, scheduling, and message facility accessibility.

2) MS 2B - Subproblem B: Process/Operating System Interface

   This subproblem is designated as MS 2B. All of the requirements in this subproblem were concerned with the extended machine instructions which are the means by which processes communicate with the operating system.

3) MS 2C - Subproblem C: Process Time-Slicing

This subproblem is designated MS 2C. All of the requirements in this subproblem are concerned with process time-slicing, the process scheduler's role and the interrupt mechanism required to handle timer interrupts. As pointed out previously, the interrupt handler includes many more functions than time interrupts. This caused some problems in interfaces descovered in the later stages; therefore, it was decided to redefine this requirement to explicitly define all of its functions.

6.3.2 MS 3 RESOURCE AND MEMORY MANAGEMENT FUNCTIONS:

MS 3 decomposed into two subproblems as follows:

1) MS 3A - Subproblem A: Resource Allocation

This subproblem is designated as MS 3A. This subproblem was concerned with the allocation of resources in general, and the mechanism for memory allocation in particular. As before, this subproblem is not clearly defined since it concerns both issues. First, the subproblem deals with some broad issues of how resources are allocated, to whom and when are they allocated. Second, the subproblem deals with the protocols for memory allocation and de-allocation; specifically, only the operating system may dynamically allocate memory. It was decided to further investigage the

issues of resource allocation and memory allocation in the next iteration to determine if the requirements or the conceptual models were ill-defined or improperly assessed.

2) MS 3B - Subproblem B: Protection

This subproblem was designated MS 3B. This subproblem is concerned with the protection mechanisms for both memory and user processes.

6.4 Relationships Among the Main Subproblems

The relationships among the main subproblems are best explained by examining the focus of each main subproblem and the conceptual models used in the interdependency assessment phase which motivated the linkages. The software package makes the linkages explicit through the "PRLK" command the results are presented in Appendix E.

The linkage between main subproblems were generalized into interfaces between the main subproblems. The following discussion will note the general characteristics of these relationships.

6.4.1 LINKAGES BETWEEN MS 1 MULTI-PROGRAMMING AND MS 2 PROCESS MANAGEMENT FUNCTIONS:

1) MS 1 Multi-programming Support;

MS 2A Process Creation/Scheduling:

This interface between these two subproblems consisted of the mechanisms for providing multi-

programming by process creation, blockage, and synchronization. Processes are created by the system and scheduled in a round-robin fashion to achieve multi-programming of user's jobs.

2) MS 1 Multi-programming Support;
MS 2B Process/Operating System Interface:

This interface between these two subproblems was concerned with signaling processing completion to the operating system, so that the next process could begin.

3) MS 1 Multi-programming Support;
MS 2C Process Time-Slicing:

The interface between these two subproblems was concerned with the mechanism of time-slicing CPU usage to achieve multi-programming. The interrupt handler requirement was included in this interface; when it seemed to belong more properly in the MS 2B subproblem. This problem supported the need to re-examine the interrupt handler requirement.

6.4.2 MS 1 MULTI-PROGRAMMING - MS 3 MEMORY MANAGEMENT FUNCTIONS:

1) MS 1 Multi-programming - MS 3A Resource and Memory Allocation

The interface between these two subproblems was concerned with the mechanisms for user and

system allocation of memory. Dynamic allocation of memory is restricted to the system processes.

2) MS 1 Multi-programming - MS 3B Protection

The interface between these two subproblems was concerned with protection of user jobs and memory. The interface did not deal with the mechanisms of protection, but the fact that protection mechanisms must exist to support multi-programming.

6.4.3 MS 1 MULTI-PROGRAMMING - MS 4 SUPERVISOR PROCESS:

The interface between these two subproblems was concerned with the mechanisms for the protection of user jobs and system processes. Protection here is defined at the job level controlled by the supervisor.

6.4.4 MS 1 MULTI-PROGRAMMING SUPPORT - MS 6 DEVICE MANAGEMENT:

The interface between these two subproblems is concerned with the procedural mechanisms by which devices support multi-programming; especially the existence of a device handler routine and the dedication of devices to user jobs.

6.4.5 MS 2 PROCESS MANAGEMENT - MS 3 MEMORY MANAGEMENT FUNCTIONS:

1) MS 2A Process Creation and Scheduling;
MS 3A Resource Allocation:

The interface between these two subproblems

was concerned with the use of information tables to enable the operating system to monitor processes and resources. This interface explicitly points out that since processes and resources must be monitored, the operating system should attempt to use the same mechanism to accomplish this task.

2) MS 2A Process Creation and Scheduling;
MS 3B Protection:

The interface between the subproblems is concerned with identification of processes by symbolic name for protection purposes.

3) MS 2B Process/Operating System Interface;
MS 3A Resource Allocation:

The interface between these two subproblems was concerned with freeing memory upon completion of a job.

4) MS 2B Process/Operating System Interface;
MS 3B Protection:

The interface between these two subproblems was concerned with the two state machine concept. A process is required to run in the problem state, all resource requests must pass through a supervisor. Therefore, protection is afforded by limiting the scope of system functions available to the user.

5) MS 2C Process Time-Slicing; MS 3B Protection:

The interface between these two subproblems is concerned with an interrupt handler to deal with unauthorized memory access requests. This interface seemes distinctly out of place, until one recalls that the requirement for all interrupt handlers regardless of purpose, is located in MS 2C. The lack of definition of the interrupt handler has been a persistent problem; therefore, it was redefined.

6.4.6 MS 2 PROCESS MANAGEMENT - MS 4 SUPERVISOR PROCESS:

1) MS 2A Process Creation/Scheduling;
MS 4 Supervisor Process:

The interface between these two subproblems was concerned with the protocols for user process creation. The supervisor process creates one process per user, initially; all others are dynamically created by the user.

2) MS 2B Process/Operating System Interface;
MS 4 Supervisor Process:

The interface between these two subproblems is concerned with the generation of an end-of-job signal from the final user process to the supervisor.

3) MS 2C Process Tim-Slicing; MS 4 Supervisor Process:

The interface between these two subproblems

is concerned with the interrupt handler which terminates user processing. Again, this is the same persistent problem of a poor interrupt handler requirement, since time-runout is just one of the interrupts for which a handler is required.

6.4.7 MS 2 PROCESS MANAGEMENT - MS 5 MESSAGE FACILITY:

1) MS 2A Process Creation and Scheduling;

MS 5 Message Facility:

The interface between these two subproblems is concerned with the usage of a message facility by user processes as a synchronization technique. This enables user processes to synchronize processing by starting and blocking each other using messages.

2) MS 2B Process/Operating System Interface;

MS 5 Message Facility:

The interface between these two subproblems was concerned with the mechanisms for message generation by the user processes.

6.4.8 MS 2 PROCESS MANAGEMENT - MS 6 DEVICE MANAGEMENT:

MS 2C Process Time-Slicing; MS 6 Device Management

The interface between these two subproblems was concerned with the generation of an I/O interrupt. Once again, this seems to be misplaced since the process time-slicing function is in no way concerned with I/O interrupt handling.

6.4.9 MS 3 MEMORY MANAGEMENT FUNCTIONS - MS 4 SUPERVISOR PROCESS:

1) MS 3A Resource Allocation; MS 4 Supervisor Process:

The interface between these two requirements deals with the issue of the timing of resource allocation and de-allocation. The supervisor process coordinates all resource allocation and de-allocation for the operating system.

2) MS 3B Protection; MS 4 Supervisor:

This interface is concerned with establishing protocols for the user destruction of user processes only. The supervisor sets up a memory partition and user processes are restricted to that memory area; therefore, they may create and destroy processes only within that memory area.

6.4.10 MS 3 MEMORY MANAGEMENT - MS 5 MESSAGE FACILITY:

MS 3A Resource Allocation; MS 5 Message Facility:

The interface between these two processes is concerned with the queuing requirements for the message facility. In order for the message facility to enqueue itself, it must be able to dynamically allocate a buffer area.

6.4.11 MS 3 MEMORY ALLOCATION - MS 6 DEVICE MANAGEMENT:

MS 3A Resource Allocation; MS 6 Device Management:

The interface between these two subproblems is concerned with the use of job control language

statements and information tables to specify and monitor resource allocations.

6.4.12 MS 4 SUPERVISOR PROCESS - MS 6 DEVICE MANAGEMENT:

The interface between these two subproblems is concerned with the reclamation of device resources upon completion of a job. It is interesting to note that allocation is not an issue, because that is controlled in MS 3 Resource and Memory Allocation.

6.5 Summary

The analysis of interfaces between subproblems is a verification procedure which supports the initial main subproblem analysis. Given two subproblems, the nature of the interface could be intuitively derived based on one's knowledge of the way in which various functions of the operating system are supposed to interface. An examination of the links, which the decomposition methodology has implied, verifies the expected result. In cases where the expected result was not verified, or if counter-intuitive interfaces were implied, one could go back to the main subproblem and find misplaced or ill-defined requirements. The second iteration of the decomposition methodology focused on a redefinition of problem requirements and a re-assessment of interdependencies for the entire requirements set.

CHAPTER VII

SECOND ITERATION OF THE DESIGN PROBLEM

The entire process of requirements definition and interdependency assessment is very much a learning process. As one continues to iterate upon the process, the requirements become more well-defined, and the assessment of interdependencies more consistent through the application of better conceptual models. The second iteration is a cumulation of a series of smaller iterations and reflects a flattening of the learning curve.

The analysis of the first iteration of the design problem highlighted a number of discrepencies in the resulting decomposition. The requirements which were identified as being problematic were re-examined from the perspective of their role in the Sample Operating System. Where warranted, these requirements were re-defined. At this point, the entire requirements set was reviewed by two graduate students familiar with operating systems in general; namely, Sid Huff and Chat-Yu Lam. Based on their analysis and recommendations, certain requirements were re-defined or re-written. The entire requirements set, in its final form as contained in Appendix G, was subjected to the interdependency assessment process. This chapter will point out the changes made to the requirements set, and present an analysis and discussion of the resulting problem structure. The chapter is organized as follows:

. Requirements re-definition.

. An analysis of the resulting problem structure for the second iteration.

. Discussion of the main subproblems.

. Discussion of the subproblems generated by a second decomposition.

. Relationships among the main subproblems.

. Comparison of the first and second interations.

7.1 Requirements Definition

The following changes were made to the preliminary set of requirements, Appendix C, based on the results of analysis of the first iteration and examination by an interested third party.

7.1.1 PRELIMINARY REQUIREMENT 6:

"The operating system must be process oriented." This requirement was considered to violate the guideline that all requirements be defined at the same level of scope. This requirement defines in very general terms that there are certain basic functions that the operating system must provide at a process level. The implications of this requirement have been made explicit in other requirements which are defined at a level more consistent with the remaining requirements set. Therefore, the requirement was changed to a design philosophy and appears as requirement 3 in the final requirements set.

7.1.2 FINAL REQUIREMENT 6:

"Input/output devices are limited to card readers for input job streams and line printers for output." I/O devices were limited by the designers of the Sample Operating System to card readers and printers. This was not made explicit in the preliminary requirements set and, therefore, is included in the final requirements set as a design constraint.

7.1.3 PRELIMINARY REQUIREMENT 11:

"User communication with the operating system is via SVC instruction." This requirement was considered to violate the implementation independence guideline for requirement definition. The specification of "SVC instruction" constrains the viewpoint of the designer unnecessarily. Therefore, the requirement was re-written and appears as requirement 12 in the final set: "User communication with the operating system is via special call".

7.1.4 PRELIMINARY REQUIREMENT 13:

"The supervisor process must create and delete the environment in which a job runs." This requirement was awkward and unclear. Therefore, it was re-written as requirement 19 in the final set: "The supervisor process must schedule jobs and prepare the job for execution".

7.1.5 PRELIMINARY REQUIREMENT 24:

"A process shall be blocked, and control released to the traffic controller, when a timer runout trap is detected."

This requirement states that there is a time limit established for processes; yet, it does not explicitly state the time limit. Therefore, the requirement was re-written making the time limit explicit, and is presented as requirement 25 of the final set: "A process must be blocked and control released to the process scheduler when a time quantum of 50 ms is exceeded".

7.1.6 PRELIMINARY REQUIREMENT 23:

"The supervisor process must reclaim all system resources when an error condition abnormally terminates a job." This requirement was unclear, since a user process is created for each job. Also the user may create additional processes, any one of which may create an error which terminates an entire job. Therefore, the requirement was re-defined and is presented as requirement 29 in the final set: "The supervisor process must reclaim all system resources when an error condition is raised by a process".

7.1.7 PRELIMINARY REQUIREMENT 41:

"Input/output devices operate via multiplexor channel." This requirement violates the implementation independence guideline for requirement definition, and is in fact redundant in the case where devices are dedicated. The requirement was, therefore, eliminated.

7.1.8 PRELIMINARY REQUIREMENT 43:

"The name of the sending process must be prefixed to a message." This requirement violated the implementation independence guideline for requirement definition, since the

real issue is the fact that the receiving process must be able to determine which process sent the message. Therefore, the requirement was re-written and is presented as requirement 53 in the final set: "The process receiving a message must be able to determine the originator of the message".

7.1.9 PRELIMINARY REQUIREMENT 43:

"A process synchronization mechanism must be provided." This requirement was the source of a number of inconsistencies in the first iteration of the design problem. Upon closer examination, it was determined that the process synchronization mechanism has a number of specific uses. The requirement was re-defined to clarify the use of the process synchronization mechanism and hopefully, reduce the inconsistencies in the design problem. The requirement was re-defined as follows:

Final Requirement 43

"A process synchronization mechanism must be provided to serve as a lock on a database."

Final Requirement 44

"A process synchronization mechanism must be provided for the timing of synchronization processes."

Final Requirement 45

"A process synchronization mechanism must be provided for synchronization between the send and receiver in message processing."

Final Requirement 46

"A process synchronization mechanism must be provided to lock a device."

7.1.10 PRELIMINARY REQUIREMENT 44:

"An interrupt mechanism must be provided." This requirement was identified as being poorly defined and leading to inconsistencies in the first iteration of the design problem. An interrupt handler is provided by the operating system for a number of specific interrupt mechanisms. Therefore, this requirement was re-defined to explicitly define each of the interrupt handlers as follows:

Final Requirement 47

"An interrupt handler routine must be provided for I/O interrupts."

Final Requirement 48

"An interrupt handler routine must be provided for program interrupts."

Final Requirement 49

"An interrupt handler must be provided for supervisor call interrupts."

Final Requirement 50

"An interrupt handler must be provided to handle external interrupts."

7.1.11

The following requirements were found to be missing from the original requirements set and, therefore, added:

Final Requirement 71

"The I/O interrupt handler routine must provide for a synchronous scheduling of a process requiring fast processing."

Final Requirement 72

"The operating system must include a task which loads the O/S into the computer and defines the processing environment."

These changes were incorporated into the final requirements set, and the interdependencies between requirements were assessed. The next section presents an analysis of the resulting problem structure after the application of the decomposition methodology.

7.2 Analysis of the Resulting Problem Structure for the Second Iteration

A total of seventy-two requirements were input into the software analysis package for decomposition. Appendix I contains a copy of the output of the decomposition and will be referred to during the analysis. As before, the analysis proceeded in the following manner.

First, the input data was verified.

Second, all isolated nodes were identified. In this decomposition, requirement 72, "The operating system must include a non-system resident task which loads the O/S into the computer and defines the processing environment" was

identified as being isolated. Although certainly a consideration for design, the initial program load routine is tailored to the final operating system design. The IPL routine may call routines provided by the operating system, but the requirements for IPL are not usually considered in the design of the operating system. As before, all padded nodes were deleted at the time.

The adjacency matrix was decomposed according to the procedure outlined in section 2.3 and the results, including a heirarchical tree are presented in Appendix I. The design requirements decomposed into eight clusters or main subproblems. The heirarchical clustering method -3 was used to generate the evaluation parameters. The evaluation parameters resulting from the second iteration are presented with those from the first iteration for comparison:

	First Iteration	Second Iteration	Change
Strength	1.9864	2.733	27% increase
Coupling	.8674	1.32	34% increase
Measure	1.1119	1.411	20% increase
Coupling/ Strength	.43	.48	10% increase
Average Main Subproblem Size	10.16	8.125	25% decrease

An examination of the evaluation parameters indicates that all have increased from the first to second iteration, with the coupling parameter showing the largest increase.

Note also that the strength has increased as well from the first to the second iteration. An increase in strength, which is the normalized evaluation of subproblem internal coherence, indicated that the main subproblems which have been identified focus closely on the general subject of each main subproblem.

Thus, an increase in the strength and coupling parameters has resulted in an increased measure for the "goodness" of the main subproblem decompositions. This measure is strictly relative from the first iteration to the second iteration. The real value of the second iteration lies in the increased understanding of the problem structure which is generalized by the decomposition methodology.

The remaining sections will analyze and describe the resulting problem structure. The final section of the chapter will present a comparison of the similarities and differences of the design structure implied by the first and second iterations.

7.3 Main Subproblems

The problem structure resulting from the application of the decomposition methodology is presented in Figure 7.1. The problem structure was interpreted as being composed of the main subproblems depicted as blocks in Figure 7.1, the subproblems, generated by a second decomposition, depicted as circles within the parent main subproblem. The inter-

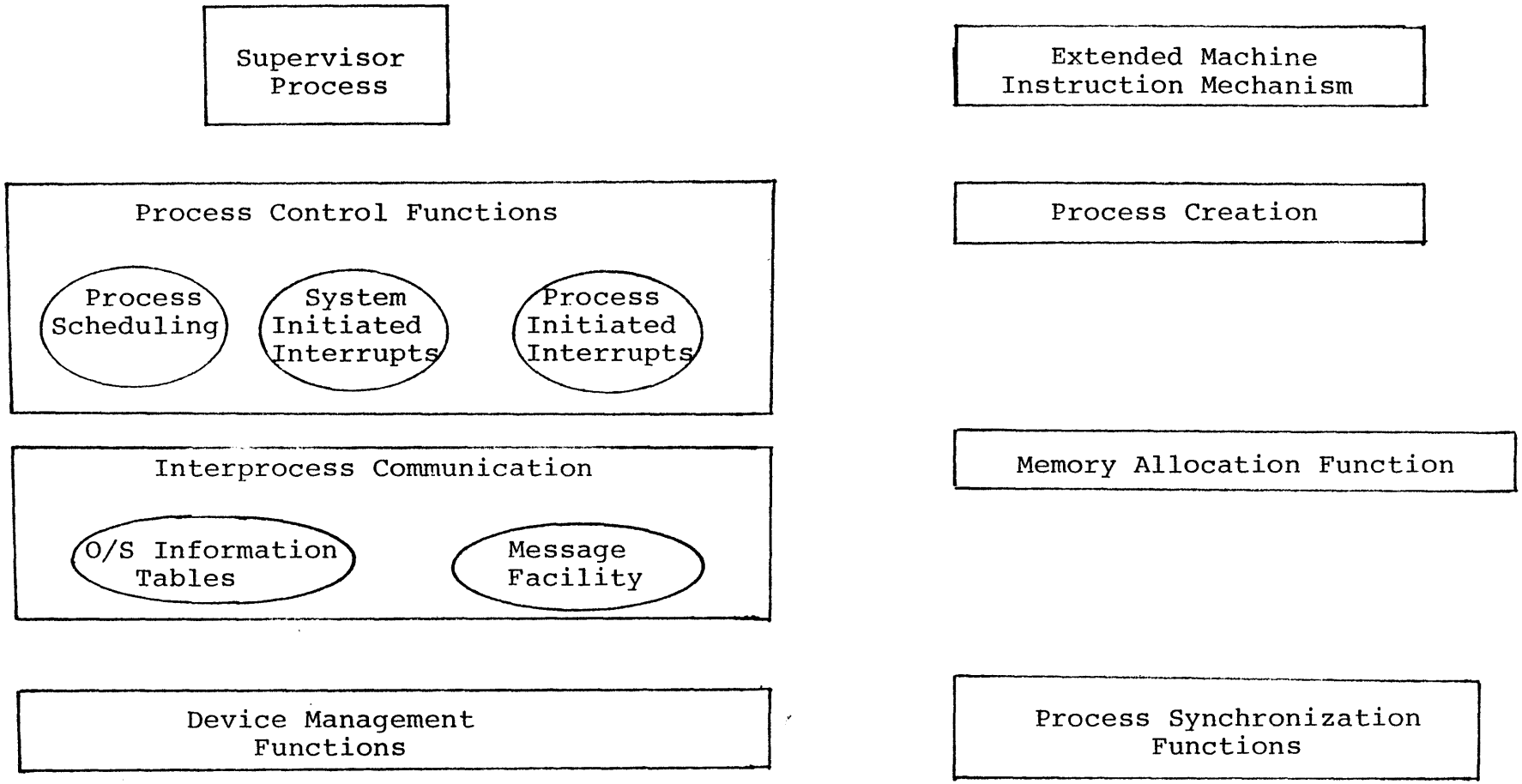


FIGURE 7.1 Problem Structure Implied by the Second Iteration of the Decomposition Methodology

dependencies among the elements of different subproblems were generalized into interfaces which are required in the design process between subproblems.

The eight main subproblems have been generalized into the following groups:

1) Supervisor Process.
2) Extended Machine Instruction Mechanism.
3) Process Control Functions.
4) Process Creation Functions.
5) Interprocess Communication.
6) Memory Allocation Functions.
7) Device Management Functions.
8) Process Synchronization Function.

The requirement statements have been separated into these eight main subproblems and are presented in Appendix J.

The following discussion will highlight the general and specific characteristics of the design structure implied by the decomposition methodology.

7.3.1 SUPERVISOR PROCESS:

The requirements which decomposed into the supervisor process main subproblem were all concerned with the generation of a multi-programming environment through the supervisor process. The supervisor process is that process which prepares and schedules the user jobs for execution. The supervisor process then consists of a number of specific tasks which must be performed for each job entering the

system. As contained in main subproblem 1, these tasks include:

- Resource allocation: system resources must be allocated to each job as it enters the system. These resources consist of memory and devices.
- Job scheduling: the supervisor schedules each job for execution. This system uses a very simplified algorithm (first-come, first-served).
- Loading: the supervisor process must load each user job into a specific memory area.
- Characteristics of the supervisor process: the supervisor process must be modularized and all system processes are written in re-entrant and shared code.

This main subproblem had the lowest individual strength parameter for all the main subproblems, indicating that the requirements are not exceptionally cohesive; or conversely that the requirements in the main subproblem cover a wider scope.

The main subproblem did not decompose on the second decomposition.

7.3.2 EXTENDED MACHINE INSTRUCTION MECHANISM:

The requirements in this main subproblem are all concerned with the extended machine instruction mechanism. The description of the Sample Operating System in section 3.2 included a brief explanation of the purpose of the extended machine instructions. Basically, the extended machine instructions were provided to enable the user to

perform certain resource management functions and hardware-like instructions.

This main subproblem contains the requirements which deal with the characteristics and protocols for the use of the extended machine instructions.

This main subproblem did not decompose on the second decomposition.

7.3.3 MS 3 PROCESS CONTROL FUNCTIONS:

The requirements in the subproblem are all concerned with the functions necessary to control processes in the operating system. Once created, a process may be "blocked" or ready to run. When "ready to run", a process may be "running" or "waiting". This main subproblem identifies the states of blocked, running, or waiting. This main subproblem also identifies what conditions may change a process state and how resource allocation is state-dependent.

The redefinition of the interrupt handler requirement is clearly apparent in this main subproblem. The control of processes in the operating system is interrupt-driven; that is, once a process becomes eligible to run, its execution is dependent upon a number of interrupts which are generated in response to an asynchronous or an exceptional event in the program. This main subproblem includes all of the interrupt handler routines and, therefore, provides for the control of processes.

The main subproblem contained two requirements which did not seem to fit into the classification of process

control. They are:

Requirement 23

"Supervisor routine must reclaim all system resources when a job is completed."

Requirement 29

"Supervisor routine must reclaim all resources when an error condition is raised."

These requirements seem to belong in the supervisor process main subproblem. The supervisor process is initially created, one per input job stream. It performs its functions of resource allocation, scheduling, and loading as a separate process. After all this has been done, the supervisor process is no longer needed until the user's job ends. It stops running and waits for a message, "success" or "failure", signalling completion to come from the user's program.

According to this scheme, the supervisor is dependent upon an interrupt signal generated by the user for successful completion, or by the system in the way of an error, to restrict and reclaim all the resources of the current user. Therefore, the mechanism by which the supervisor process is signalled to restart is contained in the interrupt handler. This is a case in which the implementation scheme of the interrupt handler and supervisor process restart ought to be considered simultaneously. When viewed from this perspective, it makes sense that requirements 23 and 29 were decomposed into main subproblem 3.

This main subproblem decomposed into three subproblems during the second decomposition.

7.3.4 MS 4 PROCESS CREATION:

The requirements in this main subproblem were all concerned with the protocols for process creation. Initially the operating system creates a single process for each user's job. The user may then create additional processes dynamically during execution. Naturally the system imposes certain constraints and procedures upon the dynamic creation of processes. These constraints and procedures are the focus of this main subproblem and deal with:

- When the user may create additional processes?
- How or by what mechanisms may these processes by created?
- How are user processes identified?
- What restrictions are imposed upon dynamically created processes?

It is noted that dynamic creation of user processes is one of the main functions necessary for multi-programming since the processes are time-sliced for CPU usage.

This main subproblem did not decompose any further.

7.3.5 MS 5 INTERPROCESS COMMUNICATION:

The requirements in this main subproblem were concerned with the tables and features provided by the operating system for interprocess communication. The main mechanism for interprocess communication has been previously identified

as the message facility. This main subproblem contains all the requirements for the message facility, as well as the requirements for system tables required to monitor and control processing. These two groups of requirements have been generalized under the heading of interprocess communication since the operating system communicates internally with information tables and user processes communicate via the message facility.

This main subproblem had the highest strength parameter of all main subproblems, indicating that this main subproblem had the greatest internal cohesiveness among requirements. This main subproblem decomposed into two well-defined subproblems in the second decomposition.

7.3.6 MS 6 MEMORY ALLOCATION FUNCTIONS:

The requirements in this main subproblem were all concerned with the protocols for memory allocation within the Sample Operating System. This main subproblem represents a distinct change from the first iteration in which numerous resource allocation procedures were also contained in this main subproblem. The second iteration has resulted in a very well-defined main subproblem; its strength parameter was the second highest, which did not decompose upon the second decomposition.

7.3.7 DEVICE MANAGEMENT FUNCTIONS:

The requirements in the main subproblem were all concerned with the functions required for device management.

This main subproblem was virtually unchanged from the previous iteration.

The main subproblem contains the requirements which deal with the following issues:

. The existence and functions of a device management system.

. Procedures for requesting resources and I/O by the user.

This main subproblem did not decompose upon the second decomposition.

7.3.8 PROCESS SYNCHRONIZATION FUNCTIONS:

The requirements in the main subproblem are specifically concerned with the process synchronization mechanism provided by the Sample Operating System. This main subproblem resulted from the redefinition of the global process synchronization requirement contained in the first iteration of the decomposition process. The requirements were redefined and analyzed independently from each other. The process synchronization mechanism is used extensively throughout the Sample Operating System to provide a linked list for the sequential locking of resources.

The existence of a main subproblem dealing exclusively with the process synchronization mechanism indicates that the implementation of this mechanism warrants the equivalent amount of design consideration given to the other main subproblems.

This main subproblem did not decompose any further.

7.4 Subproblems Generated by a Second Decomposition

A second decomposition was conducted as described in section 7.2, and resulted in the decomposition of MS 3 and MS 5 into three and two subproblems respectively. The following discussion will describe the resulting subproblems.

7.4.1 MS 3 PROCESS CONTROL FUNCTIONS:

MS 3 decomposed into three subproblems as follows:

1) MS 3A Process Scheduling:

   All of the requirements in this subproblem were concerned with the procedures necessary to schedule a process in the Sample Operating System. This subproblem was similar to MS 2A Process Creation and Scheduling, except that the functions of process creation have now been separated into an entire main subproblem.

2) MS 3B System Initiated Interrupts:

   The requirements in this subproblem define the types of interrupts that are system generated to control processing. These interrupts are centered around the time-slicing of CPU usage to achieve multi-programming. The system may also supply interrupt handler routines for supervisor calls and for external interrupts.

3) MS 3C Process Initiated Interrupts:

In contrast to MS 3B, the requirements of this subproblem are concerned with the means by which user processes may signal the operating system via interrupts to control processing. The user process must signal completion to the operating system so that resources may be reclaimed and other processes scheduled. Therefore, the subproblem is primarily concerned with user signalling completion to the operating system. As previously pointed out in section 7.3.3, this subproblem also contains the requirements that the supervisor process be restricted upon completion of the user's job.

7.4.2 MAIN SUBPROBLEM 5: INTERPROCESS COMMUNICATION:

MS 5 decomposed into two subproblems as follows:

1) MS 5A Operating System Information Tables:

The requirements in this subproblem are concerned with the operating system's use of information tables to monitor and control processing. The requirements deal with the existence of such tables and the fact that the tables must be dynamically allocated and released by the operating system.

2) MS 5B Message Facility:

The requirements of this subproblem are concerned with the existence of a message facility

for user process communication. The message facility is the primary means of user process communication and, like information tables, must be a dynamically allocated table to enable queuing of messages. The requirements deal with the procedures and constraints for sending and receiving messages.

7.5 Relationships Among the Main Subproblems

The relationships among the main subproblems were investigated as previously explained in section 6.4. The linkages between main subproblems were generalized into interfaces between the main subproblems. It is noted that the second iteration resulted in a large number of main subproblems and a larger coupling parameter. Therefore, the number of linkages between main subproblems was expected to be much greater than in the first iteration. A comparison was made of those subproblems actually having linkages in the first and second iteration.

	First Iteration	Second Iteration
Average number of linkages between subproblems for which linkages exist	3.52 links / subproblem	2.52 links / subproblem
Number of existing linkages	27	36

Although the number of subproblems having linkages is greater in the second iteration (36 vs. 27), the average

number of linkages between subproblems is more than a third less. This indicates that the interface between two given subproblems may be more highly defined since the linkages will focus on a fewer number of issues. The following discussion will attempt to make the definition of interfaces between subproblems more explicit.

7.5.1 LINKAGES BETWEEN MS 1 SUPERVISOR PROCESS AND MS 2 EXTENDED MACHINE INSTRUCTION MECHANISMS:

The interface between these two subproblems is formed due to the use of a special call instruction to request resources from the supervisor.

7.5.2 MS 1 SUPERVISOR PROCESS AND MS 3 PROCESS CONTROL FUNCTIONS:

MS 1 and MS 3A Process Scheduling:

The interfaces between these two subproblems consists of conflicting implementation. First, the memory and device resources are allocated on a job level by the supervisor. The processor is assigned on a process level only when a process is runnable. Second, jobs are scheduled strictly first-come, first-served, but there is an I/O fast processing scheme that enables asynchronous scheduling of a process requiring frequent update.

MS 1 and MS 3C Process Initiated Interrupts:

The interface between these two subproblems consists of the mechanism by which the supervisor process is re-

started after a user job terminates. MS 3C contained the seemingly misplaced requirements that the supervisor must reclaim resources when a job terminates. This interface makes the association between the supervisor process and user initiated job termination explicit, and verifies the decomposition of subproblem MS 3C.

7.5.3 MS 1 SUPERVISOR PROCESS AND MS 4 PROCESS CREATION FUNCTIONS:

The interface between these two subproblems consists of the protection mechanisms employed by the supervisor process to insure that jobs are isolated from each other. The mechanism is the creation of a single user process initially for each job, which runs exclusively in the user's partition.

7.5.4 MS 1 SUPERVISOR PROCESS AND MS 5 INTERPROCESS COMMUNICATION:

MS 1 and MS 5A Operating System Information Tables

This interface between these two subproblems deals with the fact that the supervisor process must utilize information tables to determine what resources are free or in use to support multi-programming.

7.5.5 MS 1 SUPERVISOR PROCESS AND MS 6 MEMORY ALLOCATION:

The interface between these two subproblems obviously concerns the fact that memory is a resource which must be allocated by the supervisor process.

7.5.6 MS 1 SUPERVISOR PROCESS AND MS 7 DEVICE MANAGEMENT:

The interface between these two subproblems is of a

dual nature. First the device handler routine directly supports multi-programming by providing multiple job streams from multiple sources to the system. Second, devices are resources which must be allocated to jobs by the supervisor process.

7.5.7 MS 1 SUPERVISOR PROCESS AND MS 8 PROCESS SYNCHRONIZATION:

The interface between these two subproblems is formed when the supervisor process uses process synchronization mechanism as a lock for resource allocation.

7.5.8 MS 2 EXTENDED INSTRUCTION MECHANISM AND MS 3 PROCESS CONTROL FUNCTIONS:

MS 2 and MS 3B System Initiated Interrupts

The interface between these two subproblems concerns the fact that the use of extended machine instructions generates a supervisor call interrupt. A handler routine must be provided which interprets the interrupt and performs the intended instruction.

MS 2 and MS 3C Process Initiated Interrupts

The user signals process completion by a special extended machine instruction which is the interface between these two subproblems.

7.5.9 MS 2 EXTENDED MACHINE INSTRUCTION MECHANISM AND MS 4 PROCESS CREATION FUNCTIONS:

The processes are restricted in their use of extended machine instructions; therefore, this interface is concerned

with the fact that dynamically created processes run in the problem state while extended machine instructions are executed in the supervisor state.

7.5.10 MS 2 EXTENDED MACHINE INSTRUCTION MECHANISM AND MS 5 INTERPROCESS COMMUNICATION:

MS 2 and MS 5B Message Facility

The message facility is available to all processes via extended machine instructions. The interface is concerned with the use of extended machine instructions in support of the message facility.

7.5.11 MS 2 EXTENDED MACHINE INSTRUCTION MECHANISM AND MS 8 PROCESS SYNCHRONIZATION:

The interface between these two subproblems is concerned with the protocols for use of the process synchronization mechanism. The synchronization mechanism is available via extended machine instruction; but since it serves to lock resources, it is restricted and cannot be called by user processes.

7.5.12 MS 3 PROCESS CONTROL FUNCTIONS:

MS 3A Process Scheduling and MS 4 Process Creation Functions

The interface between these two subproblems is concerned with the scheduling of dynamically created user processes. Since the scheduling is strictly round-robin, a dynamically created process is scheduled upon creation.

MS 3A and MS 5 Interprocess Communication/
MS 3A and MS 5A Operating System Information Tables

The interface between these two subproblems is concerned with the fact that ready process control blocks may be chained together to facilitate round-robin scheduling.

MS 3A and MS 5B Message Facility

The interface between these two subproblems is concerned with use of the message facility as a means of providing process synchronization.

MS 3A and MS 8 Process Synchronization

The interface between these two subproblems is concerned with the use, by the operating system, of the process synchronization mechanism to schedule or synchronize its own system processes.

MS 3B System Initiated Interrupts and MS 7 Device Management Functions

The interface between these subproblems is the I/O interrupt handler. The user process must request I/O through the operating system and the I/O interrupt handler is provided to service the user's request.

MS 3C Process Initiated Interrupts and MS 5B Message Facility

The interface between these two subproblems is concerned with the fact that when a process signals completion, all messages waiting to be read by that process are destroyed.

MS 3C and MS 6 Memory Allocation Functions

The interface between these two subproblems is concerned with memory reclaimation once the user job has completed.

MS 3C and MS 7 Device Management Functions

The interface between these two subproblems is concerned with the fact that the device handler routine must be terminated when a job is terminated.

MS 3C and MS 8 Process Synchronization

The interface between these two subproblems is concerned with the fact that all locks set by the operating system in consideration of a particular job must be released when that job terminates.

7.5.13 MS 4 PROCESS CREATION FUNCTIONS AND MS 5 INTER-PROCESS COMMUNICATION FUNCTIONS:

MS 4 and MS 5A Operating System Information Tables

The interface between these two subproblems is concerned with protection mechanisms employed by the operating system to protect dynamically created processes. The operating system utilizes information stored in tables to protect user processes.

MS 4 and MS 5B Message Facility

The interface between these two subproblems consists of the identification of user processes so that message originators and destinations may be defined.

7.5.14 MS 5 INTERPROCESS COMMUNICATION AND MS 6 MEMORY ALLOCATION:

MS 5A Operating System Information Tables and MS 6 Memory Allocation

The interface between these two subproblems is concerned with the use of information tables to allocate memory. Memory allocation is heavily dependent upon information tables to identify free areas and to enforce protection rights for certain memory areas.

MS 5A and MS 7 Device Management Functions

The interface between these two subproblems is concerned with device management functions which require dynamic system tables to monitor and control the allocation of device resources.

MS 5A and MS 8 Process Synchronization

The interface between these two subproblems is concerned with the extensive use of process synchronization mechanism with a semaphore to serve as a lock on a database. The counting semaphore may be used as a prioritized list of processes waiting for a particular resource.

MS 5B Message Facility and MS 8 Process Synchronization

The interface between these two subproblems is concerned with the use of the process synchronization mechanism to establish an ordered queue for the message facility.

7.5.16 MS 6 MEMORY ALLOCATION FUNCTIONS AND MS 7 DEVICE MANAGEMENT:

The interface between these two subproblems is concerned with the use of job control language statements to specify memory resource requirements. The JCL statement requirement is decomposed into MS 7; therefore, all resource requests must interface with MS 7 to specify the desired resources.

7.5.17 MS 6 MEMORY ALLOCATION FUNCTIONS AND MS 8 PROCESS SYNCHRONIZATION:

The interface between these two subproblems is concerned with the use of the process synchronization mechanism to serve as a lock on system tables to prevent unauthorized access or modification.

7.5.18 MS 7 DEVICE MANAGEMENT AND MS 8 PROCESS SYCHRONIZATION MECHANISM:

The interface between these two subproblems is concerned with the use of the process synchronization mechanism to lock devices in the device management function.

7.5.19 SUMMARY:

An investigation of the interfaces between pairs of subproblems identified the most obvious relationships among the main subproblems. In one case, described in section 7.5.2, the examination of interfaces has verified a seemingly misplaced decomposition of requirements. The number of interfaces among subproblems has decreased significantly in

the second iteration, resulting in interfaces which are more clearly defined.

The final section of this chapter will compare the problem structures which were implied by the first and second iterations of the decomposition methodology.

7.6 Comparison of the Design Structure Implied by the First and Second Iterations

The method for analyzing the similarities and differences in the design structure implied by the first and second iterations of the decomposition methodology was to compare the subproblems which resulted from each iteration. Each iteration was analyzed in isolation from the other; therefore, the title of each subproblem will not reveal any more than a general similarity. The comparison must include an analysis of the functions or issues involved in each subproblem to determine how the nature of each subproblem has changed from the first iteration to the second.

The first iteration resulted in the decomposition of sixty-five requirements into six main subproblems. Two of the main subproblems decomposed a second time into two and three subproblems. Therefore, the first iteration resulted in a total of nine distinct subproblems for comparison. The second iteration likewise originally resulted in a decomposition of eight main subproblems, again two of which further decomposed into two and three subproblems. Therefore, the

second iteration resulted in eleven distinct subproblems for comparison.

7.6.1 GENERAL FUNCTIONAL COMPARISON:

The general function of each subproblem was investigated from the first iteration and compared to the function of the subproblem resulting from the second iteration. A pair-wise subproblem comparison was suggested of the following form:

Subproblems From First Iteration	Subproblems From Second Iteration
1. Supervisor Process	1. Supervisor Process
2. Device Management Functions	2. Device Management Functions
3. Message Facility	3. Message Facility
4. Resource and Memory Management	4. Operating System Information Tables
5. Process Creation and Scheduling	5. Memory Allocation Functions
6. Process/Operating System Interface	6. Process Creation
7. Process Time-Slicing	7. Process Scheduling
8. Multi-programming Support Functions	8. Extended Machine Instruction Mechanism
	9. System Initiated Interrupt
	10. Process Synchronization Mechanism
	11. User Initiated Interrupts

7.6.2 COMPARISON OF SPECIFIC SUBPROBLEM FUNCTIONS:

Supervisor Process: Both iterations identified the need for a supervisor process, which prepares and schedules jobs

for execution. The supervisor process in the second iteration is more well-defined since it incorporates many of the requirements which previously had been decomposed into the multi-programming support subproblem. The requirements which shifted deal specifically with the functions of the supervisor process in the support of multi-programming.

Device management functions: The subproblems generated for the device management functions were nearly identical for the first and second iterations. The subproblem resulting from the second iteration included the requirement for job control language statements. In the previous iteration this requirement had been contained in the resource and memory allocation function subproblem.

Message facility subproblems: The subproblems generated for the message facility were identical from the first to the second iteration. However, in the first iteration the message facility constituted an entire main subproblem; whereas in the second iteration, it was a subproblem generated after a second decomposition.

Resource and memory management functions and operating system information tables and memory allocation functions:

The first iteration of the design requirements generated a main subproblem which was concerned with the allocation of resources and specifically memory by the operating system. This main subproblem was better defined in the second iteration; in that two subproblems were generated which

separated the functions of the previous main subproblem. Memory allocation subproblem, in the second iteration, is specifically concerned with these requirements for memory. The operating system information tables subproblem deals with the protocols and information requirements which had been associated with the general resource management functions of the first iteration. In addition, the mechanics of resource allocation were decomposed into the supervisor process subproblem of the second iteration which has resulted in more well-defined subproblems.

Process creation and scheduling functions and process creation and process scheduling functions: The single subproblem, Process Creation and Scheduling Functions, of the first iteration, was decomposed into one main subproblem, Process Creation and one subproblem, Process Scheduling Functions in the second iteration. The requirements involved in both iterations are identical. The functional separation achieved in the second iteration has resulted in more clearly defined subproblems.

Process/operating system interface and extended machine instruction mechanism: The requirements contained in each of these two subproblems are nearly identical from the first to the second iteration. However, in the first iteration, the process/operating system interface was a subproblem; whereas in the second iteration, the extended machine instruction mechanisms constituted an entire main subproblem.

Process time-slicing and system initiated interrupts: The requirements for process time-slicing decomposed in the first iteration into a single subproblem entitled "Process time-slicing". In the second iteration, the definition of the interrupt handler requirement had been considerably expanded. One result was the definition of a subproblem dealing with system initiated interrupts. The main focus of system initiated interrupt handler was with time runout however, it also included the requirements external and I/O interrupt handler routines as well.

Multi-programming support functions and process sychronization mechanism: The multi-programming support function main subproblem, generated in the first iteration was eliminated in the second iteration, being replaced by the process synchronization mechanism main subproblem. The multi-programming support functions included many functions which belong to the supervisor process. In fact, it was previously argued that the supervisor process main subproblem could have been considered a subset of the multi-programming main subproblem. In the second iteration, all of the requirements representing supervisor process functions have been decomposed into that main subproblem.

The process synchronization mechanism requirement was re-defined from the first to the second iteration. Since this mechanism provides basic multi-programming support, it had decomposed into the multi-programming support main sub-

problem in the first iteration. After the redefinition in the second iteration, the process synchronization mechanism had decomposed into a distinct and separate main subproblem.

User initiated interrupts: The redefinition of the interrupt handler routine requirements from the first to the second iteration resulted in the decomposition of a subproblem dealing with user initiated interrupt handler. Since the main focus of this subproblem involves the user signaling completion of a job, it had no similar subproblem counterpart from the first iteration.

7.7 Summary

The comparison of the problem structure implied by the first and second iterations yields the following results:

- The second decomposition resulted in a greater number of subproblems.
- The subproblems resulting from the second decomposition were more well-defined than those resulting from the first iteration.
- The changes in subproblems from the first iteration to the second were inituitive and seemed to result in a better problem structure.
- The interfaces between subproblems in the second iteration were also more clearly defined.

The next chapter will analyze the implications of the second iteration problem structure on the design of the Sample Operating System.

CHAPTER VIII

IMPLICATIONS OF THE DECOMPOSITION PROCESS FOR THE DESIGN OF THE SAMPLE OPERATING SYSTEM

The motivation for applying the decomposition methodology was to generate a framework upon the design requirements of the Sample Operating System to provide insight and understanding of the relationships among the system requirements. The framework resulted in the identification of subproblems of system requirements and the establishment of relationships between pairs of subproblems. The framework then constitutes a better basis for the subsequent detailed design stage, than the original disjoint set of requirements. Better in the sense that a design team now has a framework; i.e., design subproblems, in which alternative implementation schemes may be thoroughly investigated.

The prupose of this chapter is to examine the subproblems, which resulted from the second iteration of the decomposition methodology, from the perspective of the completed Sample Operating System to determine if the completed design is verified by the results of the decomposition methodology. The verification procedure was first to determine if the Sample Operating System was designed in a manner consistent with the intuitive results of the decomposition methodology by a comparison of the specific functions identified for the heirarchically structured Sample

Operating System with the functions generalized for each subproblem by the decomposition methodology. The procedure attempted to determine if the decomposition methodology indeed provided a framework for design; yet was sufficiently unconstraining so that a designer was free to investigate alternative implementations and still arrive at the final Sample Operating System design as it exists.

Since the design process for the Sample Operating System is not documented in a manner that would elucidate the decisions made by the designers in the early stages of the design process, a description of the final system was, therefore, used extensively as the only documentation aid for the system.

The second part of the verification procedure was to identify inconsistencies, non-intuitive design features, or contentions that were made obvious through the application of the decomposition methodology.

8.1 Design Overview of the Sample Operating System

The description of the Sample Operating System by Madnick and Donovan included a design overview, which closely represented the major design decisions. The design overview is presented to highlight both the design philosophy and the intent of the system designers.

> "The design of the Sample Operating System follows closely the framework presented in (Fig. 8.1).[28]"

[28] *Madnick and Donovan, p.19.*

"We build our concept of an operating system around a process. We recognize that there are certain requirements necessary to support processes. A process in the proper environment could call certain basic functions. Unfortunately, most present-day hardware does not provide these basic functions."

"Thus, our first design task is to build basic functions (extended machine for process support). These comprise the nucleus or Kernel of the operating system. Examples of these basic functions are the P-V operations, basic multiprocessing support, and traffic controlling. The reader can think of these software functions as being executed in the same way as hardware instructions.

"It is best to think of the Kernel as being an extended machine that consists of a number of extended instructions. In this implementation, the extended instructions are accomplished by means of the supervisor call instruction....."

"....Certain operating system functions can be provided in the form of special system processes rather than system primitives. In this sample operating system, there are several such processes, including the supervisor processes (job stream handlers) and the device handler processes.....The hierarchical construction of the Kernel is such that each successive level, from the bottom up, depends only on the

existence of those levels below it, and not on those above it. This approach has the advantage of pedagogical clarity, offers debugging ease, and may be relevant to the development of new theory."[29]

From Figure 8.1 one may discern five levels and layers (or modules) of the Sample Operating System.

Levels
- Process Management, lower module (lowest)
- Memory Management Module
- Process Management, upper module

Layers
- Device Management Module
- Supervisor Process Module (highest)

The functions of process management have been split into a lower module and an upper module because certain functions of process management (upper module) depend upon memory management functions, but memory management itself depends on certain process management routines that must be in a module below memory management. Clearly this step increases the pedagogical clarity of operating system. It is also noted that the Sample Operating System has no spooling process nor information management (file) system.

An examination was conducted of the functions of each level and layer in the heirarchical operating system structure of the Sample Operating System to determine if they correspond to the functions of the subproblems identified in the decomposition methodology.

[29] *Madnick and Donovan, pp. 383-385.*

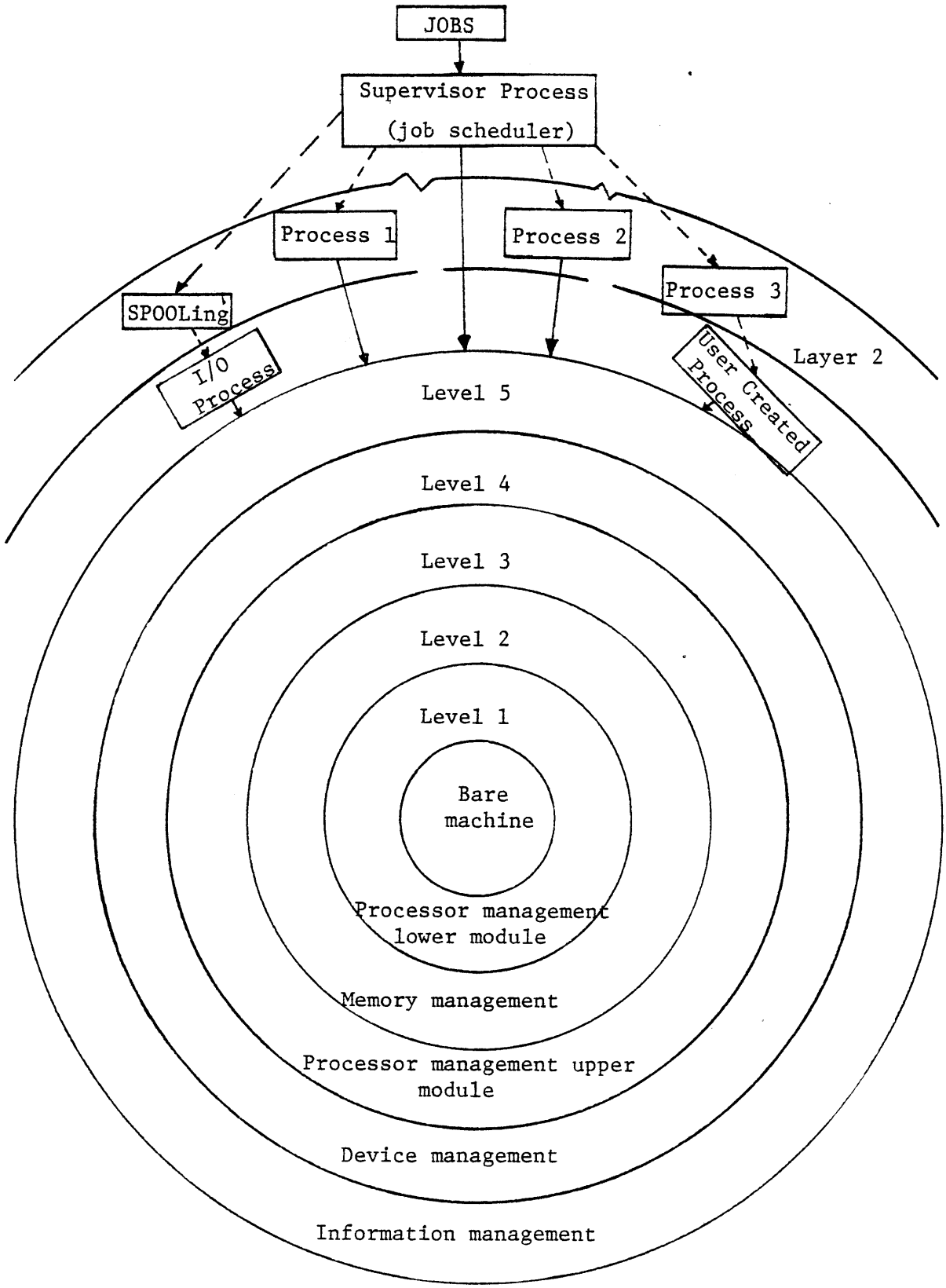


FIGURE 8.1 Heirarchical Design Structure of the Sample Operating System

8.2 Functional Comparison of the Levels and Layers of the Sample Operating System with the Subproblems Generated By the Decomposition Methodology

8.2.1 PROCESS MANAGEMENT (LOWER) MODULE COMPARED WITH PROCESS CONTROL AND PROCESS SYNCHRONIZATION MECHANISM SUBPROBLEMS:

The functional description of the process management (lower) module is as follows:

> The module schedules and runs processes that are eligible to run and provides the basic primitives for synchronization of processes.

These functions are wholly contained in the two main subproblems of Process Control and Process Synchronization mechanism. The process control main subproblem as described in section 7.3.3 is concerned with the functions necessary to control all processes in the operating system. This main subproblem decomposed into three subproblems; specifically, process scheduling, system initiated interrupt handler, and user initiated interrupt handler. The process scheduler is concerned with the procedure for scheduling eligible processes and corresponds to the process scheduler of the Sample Operating System. The system and user initiated interrupt handlers define the functions necessary for process multiplexing by the operating system and the user. These functions essentially distinguish between eligible and ineligible processes.

The second main subproblem included in the comparison is the process synchronization mechanism. As described in section 7.3.8 this main subproblem is concerned with the specific function of the synchronization mechanism and directly corresponds with the "basic primitives for synchronization of processes" described in the process management (lower) module of the Sample Operating System.

8.2.2 MEMORY MANAGEMENT MODULE COMPARED WITH THE MEMORY ALLOCATION MAIN SUBPROBLEM AND OPERATING SYSTEM INFORMATION TABLES SUBPROBLEM:

The functional description of the memory management module is as follows:

> This module performs the operations necessary for the dynamic allocation and freeing of memory for:
>
> a) job allocation.
>
> b) operating system dynamic allocation.

The allocation functions defined for job and system needs correspond to the functions described in the memory allocation main subproblem and operating system information table subproblem as described in section 7.3.6 was concerned with the protocols for memory allocation and directly correspond to the functional description of the memory management module for job partitions.

The operating system information table subproblem was decomposed from the interprocess communication main subproblem. As described in section 7.4.2 this subproblem is

concerned with the use of information tables to monitor and control processing and corresponds to the memory management module allocation functions for memory for operating system dynamic allocation. It is noted that the decomposition methodology defined the functions of operating system dynamic allocation of memory for information tables as a subproblem of interprocess communication; whereas the designers of the Sample Operating System treated the functions as a subproblem of memory management. The conceptual distinction is as follows:

a) Decomposition of the information table requirements as a subproblem of interprocess communication resulted from an assessment of "What" was the function of information table? The function is, of course, to monitor and control processing by communicating the status of resources thru tables shared among the processes of the operating system.

b) The treatment of the dynamic allocation of memory for information tables (operating system dynamic allocation) resulted from an assessment of how is the information table requirement to be implemented? Since the function requires a significant amount of memory allocation, it was considered a subproblem of the memory management module.

The interdependencies among requirements were assessed in an implementation independent environment. The applica-

tion of the decomposition methodology, a framework (i.e., subproblems) in which alternative implementation schemes may be thoroughly investiaged. For the final design, the information table subproblem was combined with the memory allocation subproblem to form the memory management module of the Sample Operating System.

This comparison raised the following issue:

If a main subproblem decomposes upon the second decomposition should one assess the main subproblem as composed of several subproblems or should one assess the subproblems as independent design problems at the same level as main subproblems?

The purpose of decomposition methodology is to provide a framework of subproblems in which the designer is free to optimize the subproblem by investigating alternative implementation schemes. The framework is meant to provide a structure for the designer, but not to impose additional constraints upon the designer's freedom. Since the assessment of subproblems as independent design problems offered more flexibility to the designer, one should, therefore, assess subproblems resulting from a second decomposition at the same level as main subproblems. In terms of the previous comparison, treating the operating system information table subproblem within the structure of the interprocess communication main subproblem would have added a constraint upon the system designer.

8.2.3 PROCESS MANAGEMENT (UPPER) MODULE COMPARED WITH THE PROCESS CREATION SUBPROBLEM AND THE MESSAGE FACILITY SUBPROBLEM:

The functional description of the process management (upper) module is as follows:

"The module provides routines for:

a) the control of processes; specifically, creation and deletion.

b) interprocess communication with buffered messages."[30]

These functions correspond to the functions of the process creation subproblem and the message facility subproblem. The process creation subproblem, as described in section 7.3.4, is concerned with the protocols for process creation, and correspond directly with the functions of this operating system module.

The second subproblem included in this comparison is the message facility subproblem which was decomposed from the interprocess communication main subproblem. For reasons stated in the last section, the message facility subproblem was treated as an independent design problem at the level of a main subproblem. Its functions, as described in section 7.4.2, are concerned with the existence and use of a message facility by all process for interprocess communication. These functions correspond with the functions of "interprocess communication with buffered messages" as

[30] *Madnick and Donovan, p.388.*

specified in the process management (upper) module.

8.2.4 DEVICE MANAGEMENT MODULE COMPARED WITH THE DEVICE MANAGEMENT FUNCTION SUBPROBLEM:

The functional description of the device management module is as follows:

> "This module provides the routines necessary to issue the appropriate input/output commands to extended devices. A special portion of the device management routine handles interrupts."[31]

These functions correspond to the functions contained in the device management function subproblem. As described in section 7.3.7, this subproblem is concerned with the functions required for device management, specifically, the procedures for requesting resources and I/O by the user.

8.2.5 SUPERVISOR PROCESS MODULE COMPARED WITH THE SUPERVISOR PROCESS MAIN SUBPROBLEM:

As implied by the title of this section, both the module and the main subproblem are nearly identical.

The functional description of the supervisor process module is as follows:

> "This module serves as the job scheduler. It can use all the functions provided by the previous modules to create an interface for the process of user jobs."[32]

These functions correspond exactly with the functions of the

---

[31] *Madnick and Donovan, p.389.*

[32] *Madnick and Donovan, p.389.*

supervisor process main subproblem. As described in section 7.3.1, the supervisor process is concerned with the generation of a multi-programming environment for user processes. It is that process which prepares and schedules user jobs for execution.

8.2.6 SUPERVISOR CALL HANDLER COMPARED WITH THE EXTENDED MACHINE INSTRUCTION MECHANISM MAIN SUBPROBLEM:

Madnick and Donovan describe an additional group of routines which are not reflected in the heirarchical operating system structure as follows:

> "Several routines don't conveniently fit our heirarchical level structure. The most notable case is the SVC handler used to activate the extended machine instructions and transfer between levels."

The requirements for these routines are wholly contained in the functional description of the extended machine instruction mechanism main subproblem. As described in section 7.3.2, the main subproblem is concerned with the characteristics and protocols for the use of the extended machine instructions. Since these instructions may be called by any level or layer of the operating system, they cannot be generalized into the heirarchical system structure.

8.2.7 SUMMARY OF THE FUNCTIONAL COMPARISON:

The comparison of the functions of the modules for the Sample Operating System with the functional description of the requirements contained in each design subproblem defined

by the description methodology has yielded several instruction insights:

. The rationale for treating design subproblems, resulting from the second decomposition of a main subproblem, as independent design problems at the level of main subproblems was developed. Since independent design problems provide a framework, yet impose fewer constraints upon the designer, the design process should deal with subproblems as independent design problems to be optimized.

. The decomposition methodology identified a greater number of subproblems, and the subproblems were internally more defined, than the levels and layers of the final operating system design. For instance, the process management (lower) module has three distinct functions:

  a) schedules and run processes;

  b) defines eligible processes;

  c) provides basic system primitives.

  The decomposition methodology identified four subproblems to correspond with the function process management (lower) module; specifically:

  a) process scheduling function;

  b) system initiated interrupt handler;

  c) user initiated interrupt handler;

  d) process synchronization mechanism.

The designer now has at his disposal a framework in which the functions of each subproblem are clearly defined, internally. The designer next investigates alternative implementation schemes to satisfy the requirements of each subproblem. In addition, the interfaces between pairs of subproblems are clearly defined so that, in the case of system and user initiated interrupt handler, common functions or processing may enable concurrent implementation schemes for subproblems so closely related.

Therefore, the designer is presented with a clearly defined framework of subproblems which he may choose to agglomerate into larger modules to satisfy the design problem.

The next section will investigate some of the inconsistencies identified by the decomposition methodology.

8.3 Inconsistencies Identified in the Comparison of the Sample Operating System and the Decomposition Methodology

The inconsistencies identified in the comparison of the Sample Operation System and the decomposition methodology were of two types. First, the final design of the Sample Operating System contained certain features that were not reflected in the results of the decomposition methodology. Second, process of requirements definition, interdependency assessment, and application of the decomposition methodology

identified unresolved contentions or conflicts in the Sample Operating System.

8.3.1 FEATURES OF THE FINAL DESIGN OF THE SAMPLE OPERATING SYSTEM NOT REFLECTED IN THE RESULTS OF THE DECOMPOSITION METHODOLOGY:

The main feature not captured in the decomposition methodology was the heirarchical nature of the Sample Operating System. This is significant because the heirarchical design incorporates a strictly limited interfacing protocol between the levels and layers of the Sample Operating System in which each successive level from the bottom up, depends only on the existence of those levels below it.

It can be argued that the heirarchical construction technique was a design decision made in a later stage of the design process since it satisfies the objective of the design; that is, the modular and heirarchically structured design is pedagogically effective. Yet, the interface protocols are very restricting and the separation of the process management module into an upper and lower module were dictated by existence dependence of upper levels upon lower levels. Therefore, an investigation was made of the linkages between subproblems to determine if the heirarchical nature of the Sample Operating System could be inferred "post facto" from the facilities available in the decomposition methodology.

The results of the previous section were used to

identify when subproblems and modules were equivalent.

Final Design	Decomposition Methodology
Process Management (lower) Module	Process Scheduling
	System Initiated Interrupt Handler
	User Initiated Interrupt Handler
	Process Synchronization Mechanism
Memory Management Module	Memory Allocation
	Operating System Information Tables
Process Management (upper) Module	Process Creation
	Message Facility
Device Management Module	Device Management Functions
Supervisor Process Module	Supervisor Process
Supervisor Call Handler	Extended Machine Instruction Mechanism

Since the linkages between subproblems are assessed in an undirected manner, and are symmetric, the actual direction of the linkages could not be determined. Therefore, no statement could be made in regard to an "upper" module calling a "lower" module.

A comparison was made of the raw number of linkages between subproblems. The tabulation of this comparison is presented in Appendix K. It was expected that some sort of trend might be established with the number of linkages,

cumulated first by subproblem, second by module. Specifically, since the process management (lower) module is the closest to the bare machine, it must be used frequently and, therefore, one would expect the number of linkages to it to be relatively large. Conversely, the device management module is a layer of the operating system; therefore, its level of interfacing in raw numbers, should be considerably less than the previous example.

The results of the comparison are as follows:

- The average number of linkages per subproblem equalled 16.18.
- Process management (lower) module had the greatest number of linkages, yet no trend could be established. That is, the number of linkages exhibited no significant trend as one approached closer to the bare machine.
- The fact that the process management (lower) module had a greater number of linkages was due more to the fact that it was composed of four subproblems, rather than by any existence dependency.

Therefore, the decomposition methodology gave no inferrence of a heirarchically structured operating system.

8.3.2 CONTENTIONS IDENTIFIED DURING THE APPLICATION OF THE DECOMPOSITION METHODOLOGY:

During the process of requirement definition, interdependency assessment, and application of decomposition

methodology, numerous unresolved issues were discovered which could lead to contentions or conflicts during implementations. These issues were involved with the implementation of system requirements and were the result of the application of worst case usage of the system to determine if the requirements set was complete. The unresolved contentions were as follows:

- The operating system must have some finite limit in the number of jobs that it will accept before a critical resource is fully allocated. The limit could involve memory, dedicated devices, or IBM System/360 protection keys. The limit was not established in the requirements, nor was any priority specified to determine which is the critical resource.
- The message requirement number 56 states: "Any number of messages, for a given process, may be queued while waiting to be read by the process." Since the memory area for buffered messages is dynamically allocated, it is conceivable that one process could do nothing but write messages to itself. Carried to an extreme all of memory could be consumed by the process in which event the system would become deadlocked. Therefore, some finite limit should be placed on the number of messages which a process may have enqueued before it is forced to read the messages.
- Requirement 24 states: "Ready processes are scheduled

in simple round-robin fashion by the process scheduler." The process scheduler checks an information table to determine if a given process is ready; if it is not ready, the process scheduler checks the next process in a sequential chain. It is conceivable that all processes in the system may become blocked at the same time. Therefore, the process scheduling function must include some mechanism to first determine that all processes are blocked and second, to attempt to resolve the situation.

The preceeding contentions became obvious during the decomposition methodology. The lack of further contentions was not meant to imply that no further contentions exist in the Sample Operating System. The decomposition methodology contained no rigorous methodology to determine if a complete and consistent set of requirements had been defined.

8.4 Summary

The final design of the Sample Operating System was verified by the results of the second iteration of decomposition methodology. The decomposition methodology identified eleven well-defined subproblems which corresponded in a consistent manner with the functions of the six levels and layers and SVC instructions handler of the final design of the Sample Operating System.

The decomposition methodology did not infer the heir-

archical structure of the final design. However, the identification of linkages between pairs of subproblems explicitly defined the interfaces which were incorporated into the modules of the final design.

The procedures involved in the decomposition methodology; that is, requirements definition, interdependency assessment and decomposition methodology, include no rigorous attempt to ensure that a complete set of requirements was defined for the Sample Operating System.

The next chapter will present recommendations for improvements of the methodology based upon the analysis of the Sample Operating System.

CHAPTER IX

CONCLUDING STATEMENTS CONCERNING THE APPLICABILITY OF THE DECOMPOSITION METHODOLOGY TO THE DESIGN PROCESS AND RECOMMENDATIONS FOR IMPROVEMENT

The purpose of this chapter is to take a retrospective view of the decomposition methodology applied to the Sample Operating System. Based on the experience, conclusions will be discussed concerning the applicability of the decomposition methodology to the design process.

9.1 Objectives of the Methodology

The objective of the application of the decomposition methodology was to support the designer in the architectural design phase by providing the designer with a framework in which the design problem can be studied in a well-defined and organized fashion. The architectural design phase consists of a well-structured series of activities that the design engineer should perform in order to achieve a better understanding of the design problems at hand, as well as to avoid implicit and unwarranged preconceptions that can bias the eventual design significantly. The decomposition methodology supports the architectural design phase by clustering the global system requirements into subproblems. The methodology then does not purport to provide a best answer, since the techniques are satisfying rather than optimizing.

The purpose of this chapter, the methodology must be expanded to include the following stages:

. Requirements definition stage;

. Interdependency Assessment Stage;

. Application of the Decomposition Methodology Developed by Andreu.

The methodology supports the design process by decomposing system requirements into subproblems. The subproblem concept narrows the scope of consideration of the design engineer to more specific well-defined areas of concern. But as pointed out by Leopold, Svendsen, and Kloehn,[33] subproblems create more levels of management and organizations produce designs which are copies of the communication structure of the organization. The result can be that the solution to the design problem becomes a series of compromises based on political expediency rather than on technical objectivity. Any methodology must provide for better communication based on technical objectivity to satisfy the design problem.

The decomposition methodology facilitates consideration and discussion of the system requirements, system objectives and constraints early in the design process. In fact, the methodology forces the user to conduct a pair-wise assessment of the interdependencies of all requirements. This is

[33] *Reuven Leopold, Edward C. Svendsen, and Harvey Kloehn, "Warship Design/Combat Subsystem Integration - A Complex Problem, Unnecessarily Overcomplicated", Naval Engineers Journal (August 1972) p.44.*

significant in two ways:

- An exhaustive pair-wise assessment of interdependencies executed in a top-down manner, forces one to think in terms of conceptual models freeing the designer of his dependencies upon tradition-bound designs.
- The decomposition methodology causes the elimination of prevalent misconception or traditional design practices by displaying the complex interrelationships which heretofore were unavailable to the designer.

The usefulness of the methodology was verified by the results of the application to the design of the Sample Operating System as stated in section 8.4. Yet the experience gained in the application of the methodology suggested improvements to all three phases of this methodology to improve both its effectiveness and to increase the scope of the applicability. The next section of the chapter will present those recommendations for improvement.

9.2 Recommendations for Improvement

9.2.1 SUGGESTIONS TO IMPROVE COMMUNICATION:

The decomposition methodology is but a small supporting tool in the overall design process; specifically, in the architectural design phase. The most time-consuming stages of the methodology were the requirements definition stage

and interdependency assessment stage. The functions required in each stage were hand-written and the analysis was performed off-line. The lack of any text facility precluded an on-line assessment of design problems. The time required to perform the stages of methodology could be reduced, and the methodology improved if the three stages could be made completely interactive by the addition of a facility for limited documentation statements. The specific documentation statements needed are defined in the supplement sections.

9.2.2 REQUIREMENTS DEFINITION:

The problems associated with generating well-defined requirements statements, even for an existing system, are well-known. This stage of the decomposition methodology represented the greatest expenditure of time and energy for this thesis. As described in section 2.1, the functional specification phase of the design process is receiving considerable attention from researchers. Sid Huff[34] has described a template format for requirements definition which recognizes six distinct statements built upon three basic language constructs. The basic constraints consist of:

objects: which are items or activities such as item - memory activity - allocated.

modifiers: which are strings of English adjectives that describe the object.

---

[34] *Sidney Huff, "An Approach to Constructing Functional Requirement Statements for Preliminary System Design"; unpublished report, MIT Sloan School, April, 1978, pp.6-7.*

Imperatives: which indicate the nature of relationships. Only two imperatives are recognized - can: implying conditional capability. will: must be fulfilled.

These constructs are used to generate six templates which are generic types of requirement statements. They consist of the following:

Properties: a feature of the system.

Treatments: an operation that is done to an object.

Timing Relationship: objects may be temporarily related.

Order Statements: order relation, such as, equal to.

Measure: consisting of a parameter and a unit.

For example,

Memory	will be	allocated	in 2K blocks
item object	imp	activity object	modifier

The template format is a useful structuring tool for requirements definition which may serve to identify ambiguities or errors. The primary benefits of the template format to the decomposition methodology are:

. It would provide a concise, well-defined requirements statement which could be generated and stored on-line using a menu of constructs.

. It would be useful for determining interrelationships

since the statements consist of well-defined key words.

. Completeness of requirements set could be verified through the use of simple algorithms which would check for the existence of capabilities clearly defined in the property statements.

9.2.3 ASSESSMENT METHODOLOGY:

The greatest weakness of the decomposition methodology is the fact that the binary assessment procedure is simplistic and, therefore, constraining. The binary assessment procedure does not allow any sort of sensitivity analysis or weighting of the interdependencies and does not allow for the representation or solution of an objective function.

The lack of an ability to represent an objective function resulted in the separation of the design philosophy and constraint statements from the requirements set that was analyzed for interdependencies. All that one could say was that the design philosophy and constraint statements must apply to every other requirement in a global sense or they apply not at all. If the interdependencies could be weighted then it would be possible to assess the relative level of impact and to establish an objective function to be satisfied. This objective function could be satisfied by a facility to describe conceptual models or mathematical relationships on-line. The mathematical relationship would be of the form of an expression interrelating different indices or measures

used to measure the degree of satisfaction of an objective function provided by each interrelationship between requirements. The constraints upon the design must also be represented as limits on certain criteria within which the final values selected for a system must fall. Ideally, all indices used to measure satisfaction of an objective function must be reduced to a common denominator. For instance, the objective function may be stated in terms of response time (T_{total}). The response time is related to CPU time for execution (T_{CPU}), Input/Output time ($T_{I/O}$), waiting or blocked time (T_w). Therefore, the objective function could be stated in terms of $T_{total} = T_{CPU} + T_{I/O} + T_w$. Interrelationships among requirements would be assessed according to a conceptual model involving a time index. The decomposition methodology could then provide a relative measure of the satisfaction of the objective function by each decomposition.

9.2.4 ADDITIONAL FEATURES:

Design is essentially an art, which is heavily dependent upon one's background and biases. It would be interesting although not necessary, to implement a facility in the decomposition methodology which would enable a user to input his own idea of the "best" decomposition in the form of subproblems. The decomposition package should then generate a measure for the proposed decomposition and would serve as a relative grade to the designer vis-a-vis the system-generated "best" decomposition.

9.3 Summary

This study has demonstrated that the decomposition methodology proposed by Dr. Andreu is a useful technique providing a framework for the designer for use in the architectural design stage. It is recognized that this methodology is a first step in the right direction. The usefulness of the first step was recognized by Mandel and Chryssostomidis:[35]

> "Unfortunately, the direct contribution of the computer to design methodology is small because the capabilities provided by the computer do not augment the user's ability as a designer but rather as an analyst. For this reason, it is felt that research leading to documentation of an improved large system design methodology that also takes advantage of today's tools is both timely and worthwhile."

The value of the decomposition methodology will improve as the results of its application are verified through similar research and improvements to the facilities are implemented by designers in search of a better world.

---

[35] *Mandel and Chryssostomidis, p.85.*

BIBLIOGRAPHY

1. Alexander, Christopher; Notes on the Synthetic Form; Cambridge, MA, 1966.

2. Alford, Mack W.; "A Requirements Engineering Methodology for Real-Time Processing Requirements"; I.E.E.E. Transactions on Software Engineering, Vol. SE-3, Number 1, (Jan. 1977), 60-69.

3. Anderberg, Michael R.; Cluster Analysis for Applications; New York, 1973.

4. Andreu, Raphael C.; "An Exercise in Software Architectural Design: From Requirements to Design Problem Structure"; Unpublished report, MIT Sloan School, June, 1977.

5. Andreu, Raphael C.; "A Systematic Approach to the Design and Structuring of Complex Software Systems"; Unpublished Doctoral thesis; MIT Sloan School, February, 1978.

6. Andreu, R. C. and Madnick, Stuart E.; "A Systematic Approach to the Design of Complex Systems: Application to DBMS Design and Evaluation"; Center For Information Systems Research Report 32, MIT Sloan School, March, 1977.

7. Bell, Thomas E., Bixler, David C., and Dyer, Margret E.; "An Extendable Approach to Computer-Aided Software Requirements Engineering"; I.E.E.E. Transactions on Software Engineering; Vol. SE-3, Number 1 (Jan. 1977) 49-60.

8. Bell, T. E. and Thayer, T. A.; "Software Requirements: Are They Really a Problem?"; Proceedings, 2nd Int'l. Conference on Software Engineering (October, 1976) 61-63.

9. Bitjen, E. J.; Cluster Analysis; Groningen, Netherlands, 1973.

10. Blashfield, Rojer K. and Aldenderfer, Mark S.; "A Consumer Report on Cluster Analysis Software"; Unpublished report, Pennsylvania State University, 1978.

11. Chu, Y.; "A Methodology for Software Engineering"; I.E.E.E. Transactions on Software Engineering, Vol. SE-1, Number 3 (Sept. 1975), 262-270.

12. Dahl, O. J., Dijkstra, E. W., and Hoare, C. A. R.; Structural Programming, London, 1972.

13. Daly, E. B.; "Management of Software Development"; I.E.E.E. Transactions on Software Engineering, Vol.SE-3, Number 3 (May, 1977), 230-243.

14. Davis, Carl G. and Vick, Charles R.; "The Software Development System"; I.E.E.E. Transactions on Software Engineering, Vol. SE-3, Number 1 (Jan, 1977), 69-85.

15. Defranco, Steven J.; "Use of Heirarchical Decomposition in Computer Systems Design"; Unpublished Master's thesis, MIT Sloan School, June, 1977.

16. Dijkstra, Edsger W.; A Discipline of Programming; Englewood Cliffs, New Jersey, 1976.

17. Donovan, John J.; Systems Programming, New York, 1972.

18. Evans, J. Harvey; "Basic Design Concepts"; A.S.N.E. JOURNAL (November, 1969), 671-678.

19. Friedman, Jerome H.; "A Recurrsive Partitioning Decision Rule for the Parametric Classification"; Stanford Linear Accelerator Center Report #CS-75-487, Stanford, Calif., Jan. 1976.

20. Huff, Sidney; "An Approach to Constructing Functional Requirement Statements for Preliminary System Design"; Unpublished report, MIT Sloan School, April, 1978.

21. Leopold, Reuven; Svendsen, CAPT Edward C.; and Kloehn, Harvey G.; "Warship Design/Combat System Integration: A Complex Problem Unnecessarily Overcomplicated"; NAVAL ENGINEER'S JOURNAL, August 1972, 28-54.

22. Liskov, Barbara H. and Berzins, Valdis; "An Appraisal of Program Specifications"; Computation Structures Group Memo 141-1, MIT Laboratory for Computer Science, April, 1977.

23. Madnick, Stuart E. and Donovan, John J.; Operating Systems, New York, 1974.

24. Mandel, P. and Chryssostomidis, C.; "A Design Methodology for Ships and Other Complex Systems"; Phil. Trans. R. Soc. Lond., A .273, (1972), 85-98.

25. Martin-Marietta Corp; TFCC System Engineering/Software Development, Preliminary TFCC Data Base Design Document, DBD-6020605, December, 1976.

26. Martin-Marietta Corp; "TFCC System Engineering/Software

Development, TFCC Program Performance Specification for a Data Management System; DDS-6020 DMS, October, 1976.

27. Mills, Harlan D.; "Software Development"; I.E.E.E. Transactions on Software Engineering, Vol. SE-2, Number 4 (Dec. 1976), 265-274.

28. Noonan, R. E.;"Structural Programming and Formal Specification"; I.E.E.E. Transactions on Software Engineering, Vo. SE-1, Number 4 (Dec. 1975), 421-425.

29. Parnas, David L.; "A Techniques of Software Module Specification with Example"; Communications of the ACM, Vol. 15, Number 5 (May, 1972), 330-337.

30. Parnas, David L.; "On the Criteria to be Used in Decomposing Systems into Modules"; Communications of the ACM, Vol. 15, Number 12 (Dec. 1972), 1053-1058.

31. Parnas, David L.; "The Use of Precise Specifications in the Development of Software"; Proceedings of Information Processing '77, (1977).

32. Punj, Doreen; Madnick, Stuart E.; and DeTreville, John D.; "A Survey of Navy Tactical Computer Applications and Executions"; Center for Information Systems Research Report 19, MIT Sloan School, October, 1975.

33. RCA, Government and Commercial Systems, Missile and Surface Radar Division; "Real Time Tactical Operating Systems Study, Second Quarterly Report; Moorsetown, New Jersey, 1974.

34. Reinhard, Nicolau; "An Experiment with Software Design Techniques"; Unpublished report, MIT Sloan School, January, 1978.

35. Ross, D. T. and Schoman, K. E., Jr.; "Structural Analysis for Requirements Definition"; I.E.E.E. Transactions on Software Engineering, Vol. SE-3, Number 1 (Jan. 1977), 6-16.

36. Salter, Kenneth G.; "A Methodology for Decomposing System Requirements into Data Processing Requirements"; Proceedings, 2nd Int'l. Conf. on Software Engineering, October, 1976.

37. Slagle, J. R.; Chang, C.-L.; and Lee, R. C. T.; "Experiments with Some Cluster Analysis Algorithms"; I.E.E.E. JOURNAL, Vol. 6, 1974, 181-187.

38. Slagle, J. R.; Chang, C.-L.; and Heller, S. R.; "A Clustering and Data-Reorganizing Algorithm", I.E.E.E., 1975, 125-128.

39. Slagle, J. and Lee, Richard C. T.; "Application of Automatic Clustering to Emitter Identification"; Naval Research Laboratory Memorandum Report 3407, November, 1976.

40. Sokal, Robert R. and Sneath, Peter H. A.; Principle of Numerical Taxonomy; San Francisco, 1965.

41. Spero, J. R.; Hicks, W. F.; and Greene, D. L.; "A Philosophy of Naval Ship Design and Construction"; NAVAL ENGINEER'S JOURNAL, October 1971, 45-52.

APPENDIX A

Formal Specification of Evaluation Parameters

FORMAL SPECIFICATION OF EVALUATION PARAMETERS:[36]

Given a graph as a pair (X,L), where

X : $\{x|x = 1,2,\ldots. |X|\}$, the set of $|X|$ objects,**

and

L : $\{\ell_{ij}|\ell_{ij}$ exists if a link joins objects $\underline{i}$ and $\underline{j} \varepsilon X\}$,

the set of links,

Define

A : a_{ij} $a_{ij} = 1$ if $_{ij}$ exists, 0 otherwise , the adjacency matrix associated with the graph.

Then, the strength S_i of a subset $X_i \underline{C} X$ can be expressed as:

$$S_i = \frac{\sum_{\substack{k,\ell \varepsilon X_i \\ k<\ell}} a_{k\ell} - (|X_i|-1)}{\frac{|X_i|(|X_i|-1)}{2}}$$

while the coupline C_{ij} between the subsets X_i and $X_j \underline{C} X$,

---

** $|X|$ *is used to indicate the cardinality of set X.*

[36] *Andreu, pp. 100-101.*

$X_i \; X_j = \phi$ (the empty set) can be written as:

$$C_{ij} = \frac{\sum\limits_{\substack{k\varepsilon X_i \\ \ell\varepsilon X_j^i}} a_{k\ell}}{|X_i| \cdot |X_j|}$$

A partition P of X,

$$P : \{X_1, X_2, \ldots, X_p\} \; , \quad \bigcup_{i=1}^{p} X_i = X, \quad \bigcap_{\substack{i,j=1 \\ i\neq j}}^{p} X_i, X_j = \phi$$

is then assigned a measure M:

$$M = \sum_{i=1}^{p} S_i \; - \sum_{\substack{i=1 \\ j=i+1}}^{p} C_{ij}$$

The behavior of M is such that the higher its value, the better the associated partition for our purposes, so that we should, in fact, search for the partition with maximum M value over all possible partitions of the set under decomposition.

APPENDIX B

Algorithm for the Identification of Kernel Subsets

ALGORITHM FOR THE IDENTIFICATION OF KERNEL SUBSETS:[37]

Recalling the following definitions:

- The "core set" CS_i associated with a node o_i to be the set CS_i : $\{o_j | o_j \text{ s.t. } a_{ij} = 1\}$; i.e., the set of all nodes related to o_i, including o_i itself, and
- The "connectivity" of node o_i to be $c_i = |CS_i| - 1$, where by $|X|$ we mean the dimension of set X.

The identification of kernel subsets can be done iteratively using the following procedure:

0) Set J = 0.

1) Compute c_i $\forall o_i \in 0$. If $c_i = c_j$ $\forall$ i,j, set J = J+1; KESU(J) = 0; stop.

2) Consider the k (> 1, a number specified a priori; see the end of this section for considerations about its value) nodes with highest c_i. Without loss of generality, assume that these are the nodes $o_1,\ldots o_k$.

3) Determine CS_i for $o_i \in \{o_1,\ldots,o_k\}$.

4) Compute $KS_i = (CS_i \left[\bigcup_{\substack{j=1 \\ i\neq j}}^{k} CS_j\right])$ $o_i \in \{o_i,\ldots,o_k\}$.

5) Select $o_p \in \{o_1,\ldots,o_k\}$ such that $KS_p = \min_{i=1,\ldots,k} (|KS_i|)$

---

[37] *Andreu, pp. 125-126.*

6) Set J = J+1;

If $|KS_p| = |CS_p|$, set DESU(J) = 0 and stop, else

set KESU(J) = o_p $[CS_p - KS_p]$.

7) Set current set to:

0 = 0 - KESU(J); if $|0| = 0$, stop.

8) Recompute A:

$$A : \{a_{ij} | a_{ij} = \begin{cases} \text{old } a_{ij} \text{ if } o_i, o_j \in 0 \\ \text{mark it "nonexistent" otherwise} \end{cases} \}$$

9) k = k - 1;

If $k > |0|$, set $k = |0|$;

Go to 1.

Once the procedure is executed, J Kernel subsets KESU(1),...,KESU J have been identified.

APPENDIX C

Preliminary Set of Requirements

1. The operating system must be simple, implementing a basic system nucleus.
2. The operating system must be designed as a pedagogical tool.
3. The operating sytem must be small; occupying fewer than 2500 cards of assembly language statements.
4. The operating system is to be implemented utilizing IBM/360 hardware.
5. The operating system must provide for a multi-programming environment.
6. The operating system must be process oriented.
7. The operating system must run on a machine that has two distinct states.
8. All resource requests must pass through the supervisor process.
9. System resources must be allocated to a job, prior to the job being made eligible to run.
10. A process must be ready to run prior to being allocated to a processor.
11. User communication with the operating system to VIA SVC Instructions.
12. The operating system must protect the user jobs from each other.
13. The operating system must utilize information tables to monitor and control processing.

14. System tables can be dynamically allocated and released.
15. Certain extended machine instructions are user callable.
16. System processes are re-entrant and shared.
17. Extended machine instructions are executed in the supervisor state.
18. The supervisor process must create and delete the environment in which a job runs.
19. Initially one process is created for each user's job.
20. Jobs are scheduled on a first come, first served basis.
21. The job scheduling function must be modularized so that improvements to the system can be easily accomplished.
22. The process scheduler must time-slice CPU usage to achieve multi-programming.
23. Ready processes are scheduled in simple round-robin fashion by the traffic controller.
24. A process shall be blocked, and control released to the traffic controller when a timer runout trap is detected.
25. A process shall be blocked and control passed to the traffic controller when the process must wait for synchronization with another process.
26. A process is blocked when it relinquishes controller to the traffic controller.

27. The supervisor routine must reclaim all system resources for a job when the job has completed.
28. The supervisor process must reclaim all system resources when an error condition abnormally terminates a job.
29. Reference to processes within a process group is by symbolic name.
30. The operating system must allocate memory for job partitions, the size of which is specified by the user.
31. Memory is allocated to a job in contiguous 2 K blocks.
32. The operating system may dynamically allocate memory to itself for system processes.
33. Memory is allocated using a best-fit algorithm.
34. Memory must be protected to prevent the simultaneous allocation of a partition to multiple jobs.
35. Free storage areas are collapsed into contiguous blocks of memory whenever a partition is freed.
36. Operating system must supply a device management system which runs as a separate process, one per device.
37. Device handler routines must support multiple job streams from card readers.
38. All devices are dedicated.
39. The device handler routine supports one card reader per input stream.
40. Device handler must support one line printer per output stream.

41. Input/output devices operate via multiplexor channel.
42. The user can provide his own routine for non-standard devices.
43. A process synchronization mechanism must be provided.
44. An interrupt mechanism must be provided.
45. P-V operations are available only to system processes.
46. A message facility must be provided for user processes.
47. The message facility is accessible by all processes.
48. The name of the sending process must be prefixed to a message.
49. The receiving process must read the name and text from the originator.
50. Messages are of arbitrary yet specified length.
51. Any number of messages may be queued while waiting to be read by a process.
52. All messages are released when a process terminates.
53. Messages are not receipted for, from receiver to sender.
54. If no messages are available to a process which expects one, it goes blocked.
55. User programs utilize a simplified job control language.
56. The operating system must accept input data from the user's job stream.
57. The supervisor process must load the user's supplied object code deck into the user partition.
58. The user process may dynamically create and destroy additional processes.

59. Dynamically created processes run in the same partition as the parent job.
60. User processes cannot dynamically allocate memory.
61. User processes cannot destroy system processes within the same process group.
62. User processes run in the problem state.
63. The user process must signal completion of the process to the operating system.
64. The user's job can reference one input device, one output device, and one exceptional device.
65. There is only one supervisor processes per job stream.

APPENDIX D

Preliminary Interdependency Assessment Results

Note: (1) (s) Indicates that the requirement indicated supports the implementation of the requirement being assessed.

(c) Indicates that the requirement indicated conflicts with the implementation of the requirement being assessed.

(2) Requirements 1 through 4 were not assessed for the reasons stated in 4.1.10.

5: The operating system must provide for a multi-programming environment.

8(s): The operating system must allocate resources as a job is read into the system.

9(s): Resource allocation is performed as a job is read into the system, except for processor allocation.

16(s): The need for pure procedures is driven by the need to provide for a multi-programming environment.

19(s): The supervisor process creates one process per job initially in support of multi-programming.

20(s): Multi-programming requires that the jobs be scheduled.

22(s): Time slicing CPU usage facilitates multi-programming.

34(s): Multi-programming requires that memory be protected to prevent simultaneous allocations of partitions.

37(s); Device handler routine facilitates the reading of multiple job streams from different sources.

43(s): Process synchronization mechanism is used to coordinate multi-programming.

55(s): JCL facility assists multi-programming by delineating jobs and specifying resource requirements.

65(s): The supervisor process controls multi-programming environment.

6: Operating system must be process oriented.

10(5): The process has certain resource requirements apart from job level requirements.

11(s): The SVC instruction support process requirements.

13(s): Most information is maintained at a process level.

19(s): A user job begins as a process.

22(s): Process environment requires the use of a traffic controller to achieve multi-programming.

23(s): An algorithm is required for process scheduling.

25(s): Multi-process synchronization is a basic function required for a process environment.

26(s): Relinquishing control to the traffic controller is a basic function of a process environment.

29(s): The naming of process is required as a means of identification.

43(s): Process synchronization mechanism is a basic tool for process oriented support.

46(s): The message facility is a basic means of inter-process communication.

47(s): Message facility must be available to all user processes.

58(s): Dynamic process creation is a basic function for a process environment.

7: Operating system must run on a machine that has two distinct states.

11(s): User communication via SVC instruction ensures that the user may be restricted from certain privileged instructions.

15(s): Only certain SVC instructions are user callable.

17(s): SVC instructions explicitly executed in the supervisor state.

62(s): User programs run in the problem state; hence, system processes run in the supervisor state.

8: All resource requests must pass through the supervisor process.

9(s): All resource requests must be made prior to a job being eligible to run.

13(s): Information tables contain the information concerning resource allocation.

27(s): Supervisor also reclaims resources when a job has completed.

28(s): Same as 27.

30(s): Memory requests are user generated.

32(s): Dynamic memory allocation takes place through the supervisor process.

55(s): JCL facility specifies the resources required of a job to the supervisor process.

60(c): The user cannot dynamically allocate memory.

64(s): The user is restricted in the number of I/O devices he may request.

9: System resources must be allocated to a job, prior to the job being made eligible to run.

10:(c): There are user resources; i.e., the processor, which are allocated at the process level.

27:(s): The same process reclaims resources upon completion.

28:(s): Same as 27.

30:(s): Memory allocation must fall within this requirement.

36:(s): Device handler routine is started for each job at this time.

55:(s): JCL facility identifies resources required of a job.

10: A process must be ready to run prior to being allocated a processor.

13:(s): A process's status is maintained in an information table (PCB).

19:(c): Initially the user's job is a process.

20:(s): The traffic controller may select a ready process only.

23(s): Ready processes must be chained into a list of eligible processes.

25(c): A process is not ready if blocked.

26(c): Same as 25.

54(c): Same as 25.

11: User communication with the operating system is via SVC instruction.

15:(c): Only certain SVC instructions are user callable.

26(s): A process relinquishes control via SVC instruction.

46(s): A request to send a message is via SVC instruction.

49(s): A request to read a message is via SVC instruction.

53(s): Dynamic process creation/destruction is via SVC instruction.

63(s): A process can signal job completion via SVC instruction.

12: The operating system must protect user jobs from each other.

13:(s): Information tables contain information on jobs, processes and resources.

18(s): Supervisor routine creates a separate environment for each job and essentially isolates it from other jobs.

34(s): Memory is also required to be protected from simultaneous user jobs.

36(s): The device management routine runs as a separate process, one per device to isolate jobs.

37(c): The device handler routines deal with many jobs and must isolate each one.

43(s): The P-V operations serve as a locking function and help to insure verifiable access rights.

59(s): Dynamically created process must remain within their process group.

13: Operating system must utilize information table to monitor and control processing.

14(s): Dynamic allocation of system tables is required for multi-programming environment.

23(s): Round-robin scheduling is most effectively accomplished by chaining PCB's.

30(s): Memory allocation requires adjustment to information tables.

32(s): Dynamic allocation of memory by the operating system is used for tables.

35(s): Free storage blocks must be updated each time memory is freed.

36:(s): Unit control blocks are built and maintained by the operating system.

43:(s): P-V operations are used extensively to update semaphores and lock resources.

46:(s): The message facility is a buffered table which is used to pass information between processes.

14: System tables can be dynamically allocated and released.

32(s): Dynamic memory allocation fully supports this requirement.

51(s): The queuing of messages requires a dynamic allocation facility.

60(c): The user is strictly prohibited from dynamic allocation.

15: Certain extended machine constructions are user callable.

26(s): The process may issue an SVC instruction to stop itself.

47(s): Message facility is implemented with user allable SVC's.

58(s): Dynamic process creation is implemented with user callable SVC's.

63(s): User signals completion via an SVC instruction.

16: System process routine an re-entrant and shared.

21(s): Job scheduling is a system process which must be shared.

32:(s): The operating system maintains pure code by dynamically allocating memory for work space for system routines.

36(s): The device management process is a system process which must be shared.

61(c): User processes cannot destroy system processes.

17: Extended machine instructions are executed in the supervisor state.

44(s): The interrupt handler must be provided to service an SVC interrupt.

18: The supervisor process must create and delete the environment in which a job runs.

19:(s): The supervisor initially creates one process per job.

27:(s): This requirement deals with the destruction of processes.

28(s): Same as 27.

58(s): User creation of processes supplements the job environment.

61(c): The user cannot destroy the entire job environment.

19: Initially one process is created for each user's job.

58(s): The user process may create additional processes to create a process group.

20: Jobs are scheduled strictly on a first come, first served basis.

21(s): FCFS scheduling is simplistic; therefore, we can improve system performance at some later time if this is strictly modularized.

39(s): The fact that all input devices are dedicated forces us to use an FCFS algorithm.

21: The job scheduling function must be modularized so that improvements to the system can be easily accomplished.

37(s): In order to improve the sophistication of the job scheduler, it would be necessary to interface to a great extent with the device handler routine.

39(s): Again for the same reason, improvements to the job scheduler are accomplished in conjunction with input stream handler.

55(s): JCL would be affected by improvements to the job scheduler.

22: The process scheduler (traffic controller) must time-slice CPU usage to achieve multi-programming.

24(s): Timer runout trap is the result of CPU usage being exceeded.

25(c): A process may terminate while awaiting synchronization.

26(c): A process may terminate voluntarily.

44(s): The interrupt handler processes a timer runout and returns control to the traffic controller.

23: Ready processes are scheduled in simple round-robin fashion by the traffic controller.

44(s): The interrupt handler gives control to the traffic controller in order to dispatch another process.

58(c): Since processes are scheduled in this fashion a user may desire to create more processes in order to grab a larger time quantum.

63(s): User signals completion so that the next process may start up.

24: A process shall be blocked, and control released to the traffic controller, when a timer runout trap is deleted.

44(s): The interrupt handler is the means by which the traffic controller regains control.

25: A process shall be blocked and control passed to the traffic controller when the process must wait for synchronization with another process.

29(s): Processes must be uniquely identifiable in order to synchronize.

43(s): P-V operations are used system-wide for synchronization, but this is directed towards synchronization of system processes.

46(s): User synchronization can be accomplished via the message facility.

47(s): Message facility is available to users.

54(s): A process, expecting a synchronizing message, is blocked until it receives one.

26: A process is blocked when it relinquishes control to the traffic controller.

63(s): The user must relinquish control by a specific signal to the operating system.

27: The supervisor routine must reclaim all system resources for a job when the job has completed.

28(c): The supervisor must also reclaim resources if a user commits an error.

35(s): When memory is freed by direction of the supervisor it must also re-configure.

36(s): The device handler routine is a resource that must be reclaimed.

38(s): The devices used must be released.

44(s): A program interrupt starts things happening.

61(c): The supervisor routine must destroy all system processes for a job which terminates.

63(s): The user must signal completion.

28: The supervisor process must reclaim all system resources when an error condition abnormally terminates a job.

35:(s): Memory is re-configured when it is reclaimed.

36:(s): The device management routine must be reclaimed.

38(s): Devices resources must be reclaimed at this time.

44(s): The interrupt handler signals that an error has occurred.

61(c): The supervisor must destroy all system processes for a terminated job.

29: Reference to processes within a process group is by symbolic name.

48(s): The sending process must have a name.

49(s): The receiving process must have a name for the message facility to operate.

58(s): A process is given a name at creation time.

59(s): Names are unique with a partition.

30: The operating system must allocate memory for job partitions, the size of which is specified by the user.

31(c): Memory allocation is limited to increments of 2K blocks.

32(c): Memory may also be allocated by the system.

33(s): A list of free areas is updated each time a partition is freed.

55(s): A simplified JCL is available for the user to specify his memory requirements.

59(c): Memory partition requested must be large enough for all dynamically created processed.

60(c): The user cannot dynamically allocate memory.

31: Memory is allocated to a job in contiguous 2K blocks.

32(c): The operating system does not need memory allocated on 2K blocks since it has its own protection scheme.

33(s): Best-Fit algorithm memorizes partition waste.

34(s): Allocation in 2K blocks allows hardware protection by the IBM 360 system.

35(s): Memory is re-configured whenever it is freed.

43(s): P-V operations can serve as a lock on a database.

55(s): The user specifies memory requirements using JCL.

32: Operating system may dynamically allocate memory to itself for system processes.

34(s): System workspaces must be protected the same as user work spaces.

35(s): Free areas are collapsed for system processes.

36(s): Device management system requires memory for its own tables.

51(s): Message queuing facility requires memory.

60(c): The user cannot dynamically allocate memory.

33: Memory is allocated using a best-fit algorithm.

35(s): Memory is reconfigured when deallocated to insure that the largest contiguous blocks are available.

55(s): User must specify memory requirements on JCL.

34: Memory must be protected to prevent the simultaneous allocation of a partition to multiple jobs.

43:(s): The P-V operation is used extensively as a lock on a database.

44:(s): The interrupt handler is provided as a means of detecting out-of-bounds memory requests.

59(s): Dynamically created processes must run in the partition of the parent job which further

protects memory.

60(s): The user is prevented from allocating additional memory.

35: Free storage areas are collapsed into contiguous blocks of memory whenever a partition is freed.

63:(s): The user must signal completion to the operating system so that partition can be freed.

36: Operating system must supply a device management system, which runs as a separate process, one per device.

37(s): Device handler must be included within device management system.

38(s): Since devices are dedicated, only one process per device is required.

39(s): These constitute the specific requirements of the device handler routine.

40(s): Same as above.

41(s): Since I/O devices operate via multiplexor channel there is not need for I/O traffic controller.

42(s): Device management system must enable the user to supply his own handling routines.

43(s): P-V operation is used to lock devices.

44(s): P-V plus limited interrupt facility provide I/O interface.

37: Device handler routines must support multiple job streams from card readers.

38(s): Dedicated devices enable sequential processing and simplify designation of job streams.

39(s): A card reader represents an input stream; hence, multiple card readers represent multiple job streams.

41(s): Multiplexed channels enable simultaneous servicing of multiple devices.

43(s): P-V operations are used to lock devices.

56:(s): The device handler must be able to distinguish among user decks and data cards.

38: All devices are dedicated.

39:(s): Since devices are dedicated, a card reader represents an input stream.

40(s): Since devices are dedicated, a line printer represents an output stream.

41(s): Multiplexed channel is used for dedicated service.

42(c): Non-standard devices may not necessarily be dedicated.

43(s): P-V operations are used to lock devices.

64(s): User must specify which devices are being used by his program.

39: The device handler routine supports one card reader per input stream.

40(c): The output stream conversely supports one line printer.

41(s): Multiplexing eliminates the need for an I/O traffic controller.

42(c): A user may specify his own routine.

64(s): The user must designate the card reader to be used.

40: Device handler must support one line printer per output stream.

41(s): Multiplexing eliminates the need for an I/O traffic controller.

42(s): A user can supply his own routines.

64(s): The user must specify the line printer to be used.

41: Input/output devices operate via multiplexor channel.

42(c): The user may provide his own routines and I/O interface.

43(s): The P-V operation can be used to lock a device.

56(s): Input data for a user's program must be accepted via multiplex or channel.

42: The user can provide his own routine for non-standard devices.

64: The user must specify the use of an exceptional device to the system via JCL.

43: A process synchronization mechanism must be provided.

45(s): The synchronization mechanism is used as the basis for process support and, therefore, is not available to users.

44: An interrupt handler must be provided.

63(s): A user signals completion via SVC interrupt.

46: A message facility must be provided for user processes.

47:(s): The message facility is available to all processes.

48:(s): Requirements for sending a message.

49:(s): Requirements for receiving a message.

50:(s): This contains the message length requirement.

51:(s): Messages may be queued in order to be read by a process.

52:(s): Messages are released when a process terminates.

53:(s): The message facility has no receipt mechanism.

54:(s): Messages can be used for process synchronization.

48: The name of the sending process must be prefixed to a message.

49(s): The receiving message must be able to read from whom the message came.

53(s): The message facility does not receipt for message transfer.

54(s): The message facility can be used for one-to-one process synchronization.

49: The receiving process must read the name and text from the originator.

51(c): The queuing process makes it essential that the message receiver be able to tell from whence the message came.

53(c): Messages are not receipted for.

54(c): A process awaiting synchronization must be able to determine that the message is from the proper source.

50: Messages are of an arbitrary, yet specified length.

51(s): Since messages may vary in length, queuing them is the most simplistic means of dealing with the variable length.

52: All messages are released when a process terminates.

53(s): The sending process may have been terminated before the receiving process read the message.

55: User programs utilize a simplified job control language.

60:(s): User is limited to the amount of memory specified in his JCL.

64(s): The user must specify his input/output device requirements utilizing JCL statements.

56: The operating system must accept input data from the user's job stream.

65:(s): The supervisor process controls the loading of the user's deck into the machine.

57: The supervisor process must load the user-supplied object deck into the user partition.

60:(s): Once the user's deck is loaded, he is stuck with whatever memory partition he requested.

65:(s): The supervisor process handles the loading function.

58: The user process may dynamically create and destroy additional processes.

59(s): Dynamically created processes are limited to the user's partition.

60(c): The user cannot destroy system processes.

61(s): User-created processes are limited to problem state.

59: Dynamically created processes run in the same partition as the parent job.

60:(s): The user process cannot create processes which also expand its memory requirements.

62:(s): User processes all run to problem state.

61: User processes cannot destroy system processes within the same process group.

62:(s): Since all user processes run in the problem state, and system processes in the supervisor

state, we are protected.

63: The user process must signal completion of the process to the operating system.

65: The supervisor process now takes over to reclaim resources or to signal the traffic controller.

APPENDIX E

Results of the Interactive Decomposition Package for the First Iteration

NOLK

RECORDED LINKS.
FROM NODE TO NODE(S):

```
  5   ( 11)  8,  9, 16, 19, 20, 22, 34, 37, 43, 55,
             65,
  6   ( 13) 10, 11, 13, 19, 22, 23, 25, 26, 29, 43,
             46, 47, 58,
  7   (  4) 11, 15, 17, 62,
  8   ( 10)  5,  9, 13, 27, 28, 30, 32, 55, 60, 64,
  9   (  8)  5,  8, 10, 27, 28, 30, 36, 55,
 10   (  9)  6,  9, 13, 19, 20, 23, 25, 26, 54,
 11   (  8)  6,  7, 15, 26, 46, 49, 58, 63,
 12   (  7) 13, 18, 34, 36, 37, 43, 59,
 13   ( 12)  6,  8, 10, 12, 14, 23, 30, 32, 35, 36,
             43, 46,
 14   (  4) 13, 32, 51, 60,
 15   (  6)  7, 11, 26, 47, 58, 63,
 16   (  5)  5, 21, 32, 36, 61,
 17   (  2)  7, 44,
 18   (  6) 12, 19, 27, 28, 58, 61,
 19   (  5)  5,  6, 10, 18, 58,
 20   (  4)  5, 10, 21, 39,
 21   (  5) 16, 20, 37, 39, 55,
 22   (  6)  5,  6, 24, 25, 26, 44,
 23   (  6)  6, 10, 13, 44, 58, 63,
 24   (  2) 22, 44,
 25   (  8)  6, 10, 22, 29, 43, 46, 47, 54,
 26   (  6)  6, 10, 11, 15, 22, 63,
 27   ( 10)  8,  9, 18, 28, 35, 36, 38, 44, 61, 63,
 28   (  9)  8,  9, 18, 27, 35, 36, 38, 44, 61,
 29   (  6)  6, 25, 48, 49, 58, 59,
 30   (  9)  8,  9, 13, 31, 32, 33, 55, 59, 60,
 31   (  7) 30, 32, 33, 34, 35, 43, 55,
 32   ( 11)  8, 13, 14, 16, 30, 31, 34, 35, 36, 51,
             60,
 33   (  4) 30, 31, 35, 55,
 34   (  9)  5, 12, 31, 32, 43, 44, 59, 60, 62,
 35   (  7) 13, 27, 28, 31, 32, 33, 63,
 36   ( 15)  9, 12, 13, 16, 27, 28, 32, 37, 38, 39,
             40, 41, 42, 43, 44,
 37   (  9)  5, 12, 21, 36, 38, 39, 41, 43, 56,
 38   ( 10) 27, 28, 36, 37, 39, 40, 41, 42, 43, 64,
 39   (  9) 20, 21, 36, 37, 38, 40, 41, 42, 64,
 40   (  6) 36, 38, 39, 41, 42, 64,
 41   (  8) 36, 37, 38, 39, 40, 42, 43, 56,
 42   (  6) 36, 38, 39, 40, 41, 64,
 43   ( 12)  5,  6, 12, 13, 25, 31, 34, 36, 37, 38,
             41, 45,
 44   (  9) 17, 22, 23, 24, 27, 28, 34, 36, 63,
 45   (  1) 43,
 46   ( 12)  6, 11, 13, 25, 47, 48, 49, 50, 51, 52,
             53, 54,
```

```
 47    (  4)  6, 15, 25, 46,
 48    (  5) 29, 46, 49, 53, 54,
 49    (  7) 11, 29, 46, 48, 51, 53, 54,
 50    (  2) 46, 51,
 51    (  5) 14, 32, 46, 49, 50,
 52    (  2) 46, 53,
 53    (  4) 46, 48, 49, 52,
 54    (  5) 10, 25, 46, 48, 49,
 55    (  9)  5,  8,  9, 21, 30, 31, 33, 60, 64,
 56    (  3) 37, 41, 65,

 57    (  2) 60, 65,
 58    ( 10)  6, 11, 15, 18, 19, 23, 29, 59, 60, 61,
 59    (  7) 12, 29, 30, 34, 58, 60, 62,
 60    (  9)  8, 14, 30, 32, 34, 55, 57, 58, 59,
 61    (  6) 16, 18, 27, 28, 58, 62,
 62    (  4)  7, 34, 59, 61,
 63    (  8) 11, 15, 23, 26, 27, 35, 44, 65,
 64    (  6)  8, 38, 39, 40, 42, 55,
 65    (  4)  5, 56, 57, 63,

(AVERAGE NO. OF LINKS PER NODE:      6.058).

 REQ:
ISOL

 ISOLATED NODES:
   1
   2
   3
   4
  66
  67
  68
  69

 REQ:
DENO
:
/1,2,3,4,66,67,68,69/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
    1    2    3    4   66   67   68   69
```

NODES HAVE BEEN RENAMED AS FOLLOWS:

OLD NO.	NEW NO.
5	1
6	2
7	3
8	4
9	5
10	6
11	7
12	8
13	9
14	10
15	11
16	12
17	13
18	14
19	15
20	16
21	17
22	18
23	19
24	20
25	21
26	22
27	23
28	24
29	25
30	26
31	27
32	28
33	29
34	30
35	31
36	32
37	33
38	34
39	35
40	36
41	37
42	38
43	39
44	40
45	41
46	42
47	43
48	44
49	45
50	46
51	47
52	48

```
   53       49
   54       50
   55       51
   56       52
   57       53
   58       54
   59       55
   60       56
   61       57
   62       58
   63       59
   64       60
   65       61

 REQ:
SAVE
 ENTER FILE NAME:
SOSA12

 STATUS SAVED IN FILE SOSA12

 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1     ( 1)   1
    2     ( 1)   2
    3     ( 1)   3
    4     ( 1)   4
    5     ( 1)   5
    6     ( 1)   6
    7     ( 1)   7
    8     ( 1)   8
    9     ( 1)   9
   10     ( 1)  10
   11     ( 1)  11
   12     ( 1)  12
   13     ( 1)  13
   14     ( 1)  14
   15     ( 1)  15
   16     ( 1)  16
   17     ( 1)  17
   18     ( 1)  18
   19     ( 1)  19
   20     ( 1)  20
   21     ( 1)  21
   22     ( 1)  22
   23     ( 1)  23
   24     ( 1)  24
   25     ( 1)  25
   26     ( 1)  26
   27     ( 1)  27
   28     ( 1)  28
   29     ( 1)  29
   30     ( 1)  30
   31     ( 1)  31
   32     ( 1)  32
   33     ( 1)  33
   34     ( 1)  34
   35     ( 1)  35
   36     ( 2)  36  38
   37     ( 1)  37
   38     ( 1)  39
   39     ( 1)  40
   40     ( 1)  41
   41     ( 1)  42
   42     ( 1)  43
   43     ( 1)  44
   44     ( 1)  45
   45     ( 1)  46
   46     ( 1)  47
   47     ( 1)  48
   48     ( 1)  49
   49     ( 1)  50
   50     ( 1)  51
   51     ( 1)  52
   52     ( 1)  53
   53     ( 1)  54
   54     ( 1)  55
   55     ( 1)  56
   56     ( 1)  57
   57     ( 1)  58
   58     ( 1)  59
   59     ( 1)  60
   60     ( 1)  61
```

```
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
SIMA

 SIMILARITY MATRIX COMPUTED.

 REQ:
INPA
 ENTER PERCENTAGE PARAMETER:
80.

 INITIAL PARTITION COMPUTED WITH P = 80.00 %.

 REQ:
HCM1
 BEST PARTITION MEASURE:    0.556
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL
```

CLUSTER (NO)		OBJECTS									
1	(15)	1	4	5	9	10	26	27	28	29	30
		31	51	55	56	58					
2	(13)	2	3	6	7	11	13	15	19	21	22
		43	54	59							
3	(15)	8	12	16	17	32	33	34	35	36	38
		37	39	41	52	60					
4	(7)	14	18	20	23	24	40	57			
5	(11)	25	42	44	45	46	47	48	49	50	53
		61									

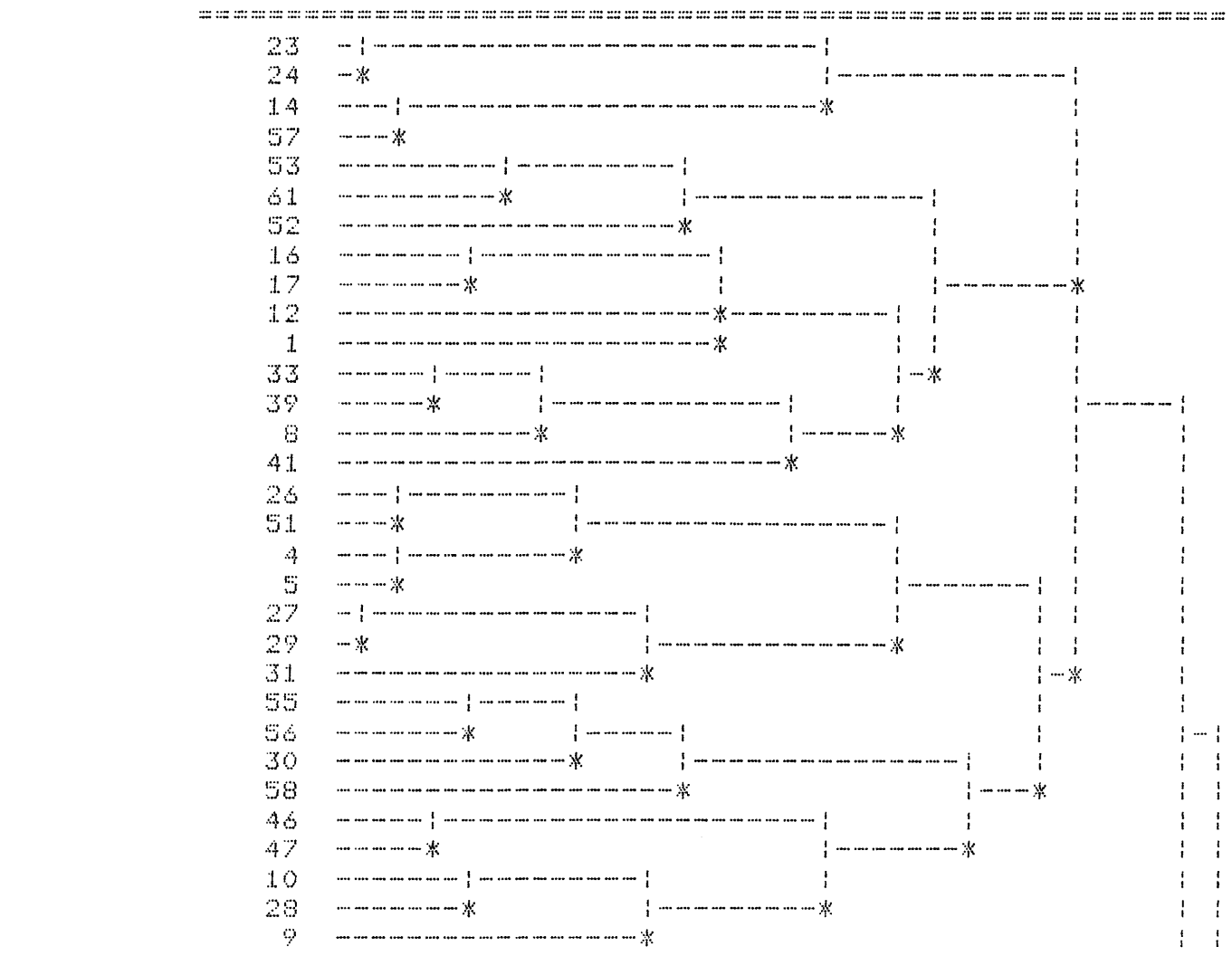
```
 REQ:
HCM2
 BEST PARTITION MEASURE:    0.577
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL
```

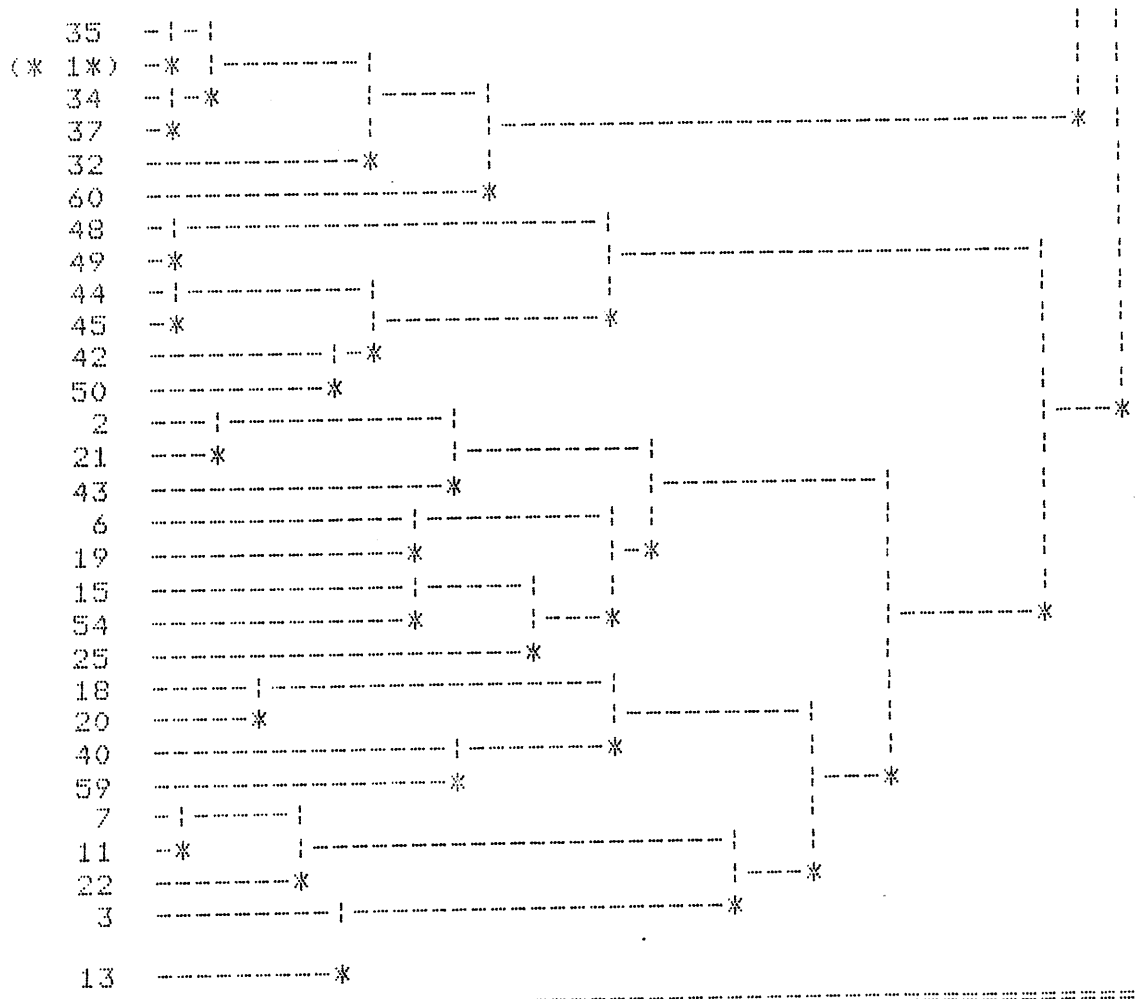
CLUSTER (NO)	OBJECTS									
1	(14)	1	8	12	16	17	32	33	34	35	36
		38	37	39	60						
2	(14)	2	6	7	11	15	18	19	20	21	22
		40	43	54	59						
3	(2)	3	13								
4	(14)	4	5	9	10	26	27	28	29	30	31
		51	55	56	58						
5	(4)	14	23	24	57						
6	(9)	25	42	44	45	46	47	48	49	50	
7	(1)	41									
8	(3)	52	53	61							

```
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
HCM3
 BEST PARTITION MEASURE:    1.119
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL
```

CLUSTER (NO)	OBJECTS									
1	(11)	1	8	12	16	17	33	39	41	52	53
		61									
2	(17)	2	3	6	7	11	13	15	18	19	20
		21	22	25	40	43	54	59			
3	(16)	4	5	9	10	26	27	28	29	30	31
		46	47	51	55	56	58				
4	(4)	14	23	24	57						
5	(7)	32	34	35	36	38	37	60			
6	(6)	42	44	45	48	49	50				

```
 REQ:
HCM3
 BEST PARTITION MEASURE:     1.119
 DO YOU WANT TO PRINT THE TREE?
YES
 SET PAPER AND PRESS RETURN:
```





COLLAPSED OBJECTS:

(* 1*): 36 38

MEASURES:

-193.500	-187.500	-180.167	-173.667	-168.167	-165.667
-158.917	-153.917	-145.917	-138.917	-133.750	-123.500
-120.250	-112.500	-109.000	-102.750	-96.250	-93.050
-88.217	-82.800	-80.050	-77.717	-70.500	-66.333
-61.361	-56.278	-51.236	-45.653	-40.528	-37.375
-32.875	-31.591	-26.806	-23.694	-21.278	-20.155
-18.224	-16.012	-12.858	-10.633	-8.539	-7.770
-7.174	-5.805	-3.277	-2.350	-1.722	-1.070
-0.348	0.160	0.484	0.779	0.911	1.119
0.979	1.051	0.742	0.240	0.081	

```
 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1     (11)   1   8  12  16  17  33  39  41  52  53
                61
    2     (17)   2   3   6   7  11  13  15  18  19  20
               21  22  25  40  43  54  59
    3     (16)   4   5   9  10  26  27  28  29  30  31
               46  47  51  55  56  58
    4     ( 4)  14  23  24  57
    5     ( 7)  32  34  35  36  38  37  60
    6     ( 6)  42  44  45  48  49  50

 REQ:
EVAL

 STRENGTH:  1.9864,
 COUPLING:  0.8674,
 MEASURE:   1.119,

 REQ:
DENO
:
/2,3,6,7,11,13,15,18,19,20,21,22,25,40,43,54,59,4,5,9,10,26,27,28,
;
29,30,31,46,47,51,55,56,58,14,23,24,57,32,34,35,36,38,37,60,42,44,
;
45,48,49,50/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
   2    3    4    5    6    7    9   10   11   13
  14   15   18   19   20   21   22   23   24   25
  26   27   28   29   30   31   32   34   35   36
  37   38   40   42   43   44   45   46   47   48
  49   50   51   54   55   56   57   58   59   60

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO.  NEW NO.
 ----------------
   1        1
   8        2
  12        3
  16        4
  17        5
  33        6
  39        7
  41        8
  52        9
  53       10
  61       11
```

```
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 NO PRECLUSTERING PERFORMED; DISTANCE MATRIX COMPUTED WITH P = 1.

 REQ:
SIMA

 SIMILARITY MATRIX COMPUTED.

 REQ:
INPA
 ENTER PERCENTAGE PARAMETER:
80.

 INITIAL PARTITION COMPUTED WITH P = 80.00 %.

 REQ:
HCM3
 BEST PARTITION MEASURE:    0.091
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1    (11)   1   2   3   4   5   6   7   8   9  10
                11

 REQ:
EVAL

 STRENGTH:  0.0909,
 COUPLING:  0.0000,
 MEASURE:   0.091.
```

```
 REQ:
REST
 ENTER FILE NAME:
SOSA12

 ADJACENCY MATRIX READ FROM FILE SOSA12

 REQ:
DENO
:
/1,8,12,16,17,33,39,41,52,53,61,4,5,9,10,26,27,28,29,30,31,46,47,
:
51,55,56,58,14,23,24,57,32,34,35,36,38,37,60,42,44,45,48,49,50/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
   1    4    5    8    9   10   12   14   16   17
  23   24   26   27   28   29   30   31   32   33
  34   35   36   37   38   39   41   42   44   45
  46   47   48   49   50   51   52   53   55   56
  57   58   60   61

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO.  NEW NO.
 ---------------
    2        1
    3        2
    6        3
    7        4
   11        5
   13        6
   15        7
   18        8
   19        9
   20       10
   21       11
   22       12
   25       13
   40       14
   43       15
   54       16
   59       17
```

```
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 NO PRECLUSTERING PERFORMED; DISTANCE MATRIX COMPUTED WITH P = 1.

 REQ:
SIMA

 SIMILARITY MATRIX COMPUTED.

 REQ:
INPA
 ENTER PERCENTAGE PARAMETER:
80.

 INITIAL PARTITION COMPUTED WITH P = 80.00 %.

 REQ:
HCM3
 BEST PARTITION MEASURE:    0.448
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1     ( 8)   1   3   7   9  11  13  15  16
    2     ( 6)   2   4   5   6  12  17
    3     ( 3)   8  10  14

 REQ:
EVAL

 STRENGTH:  0.8857,
 COUPLING:  0.4375,
 MEASURE:   0.448.
```

```
 REQ:
REST
 ENTER FILE NAME:
SOSA12

 ADJACENCY MATRIX READ FROM FILE SOSA12

 REQ:
DENO
:
/1,8,12,16,17,33,39,41,52,53,61,2,3,6,7,11,13,15,18,19,20,21,22,25,
:
40,43,54,59,14,23,24,57,32,34,35,36,38,37,60,42,44,45,48,49,50/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
    1     2     3     6     7     8    11    12    13    14
   15    16    17    18    19    20    21    22    23    24
   25    32    33    34    35    36    37    38    39    40
   41    42    43    44    45    48    49    50    52    53
   54    57    59    60    61

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO.  NEW NO.
 ---------------
    4        1
    5        2
    9        3
   10        4
   26        5
   27        6
   28        7
   29        8
   30        9
   31       10
   46       11
   47       12
   51       13
   55       14
   56       15
   58       16
```

```
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 NO PRECLUSTERING PERFORMED; DISTANCE MATRIX COMPUTED WITH P = 1.

 REQ:
SIMA

 SIMILARITY MATRIX COMPUTED.

 REQ:
INPA
 ENTER PERCENTAGE PARAMETER:
80.

 INITIAL PARTITION COMPUTED WITH P = 80.00 %.

 REQ:
HCM3
 BEST PARTITION MEASURE:    0.449
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1    (13)   1   2   3   4   5   6   7   8  10  11
                12  13  15
    2    ( 3)   9  14  16

 REQ:
EVAL

 STRENGTH:  0.5769,
 COUPLING:  0.1282,
 MEASURE:   0.449.
```

```
 REQ:
!EST
 ENTER FILE NAME:
SOSA12

 ADJACENCY MATRIX READ FROM FILE SOSA12

 REQ:
DENO
:
/1,8,12,16,17,33,39,41,52,53,61,2,3,6,7,11,13,15,18,19,20,21,22,25,
:
40,43,54,59,4,5,9,10,26,27,28,29,30,31,46,47,51,55,56,58,32,34,35,36,38,
:
37,60,42,44,45,48,49,50/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
    1     2     3     4     5     6     7     8     9    10
   11    12    13    15    16    17    18    19    20    21
   22    25    26    27    28    29    30    31    32    33
   34    35    36    37    38    39    40    41    42    43
   44    45    46    47    48    49    50    51    52    53
   54    55    56    58    59    60    61

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO. NEW NO.
 ---------------
   14        1
   23        2
   24        3
   57        4

 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
SIMA

 SIMILARITY MATRIX COMPUTED.

 REQ:
INPA
 ENTER PERCENTAGE PARAMETER:
80.

 INITIAL PARTITION COMPUTED WITH P = 80.00 %.

 REQ:
MCM3
 CURRENT PRCLUSTERING HAS ONLY ONE CLUSTER.
 UNABLE TO DO IT.
```

```
 REQ:
REST
 ENTER FILE NAME:
SOSA12

 ADJACENCY MATRIX READ FROM FILE SOSA12

 REQ:
DENO
;
/1,8,12,16,17,33,39,41,52,53,61,2,3,6,7,11,13,15,
*
   18,19,20,21,22,25,40,43,54,59,4,5,9,
    ;
    10,26,27,28,29,30,31,46,47,51,55,56,58,

   14,23,24,57,42,44,45,48,49,50/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
   1    2    3    4    5    6    7    8    9   10
  11   12   13   14   15   16   17   18   19   20
  21   22   23   24   25   26   27   28   29   30
  31   33   39   40   41   42   43   44   45   46
  47   48   49   50   51   52   53   54   55   56
  57   58   59   61

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO. NEW NO.
 ---------------
   32       1
   34       2
   35       3
   36       4
   37       5
   38       6
   60       7
```

```
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
SIMA

 SIMILARITY MATRIX COMPUTED.

 REQ:
INPA
 ENTER PERCENTAGE PARAMETER:
80.

 INITIAL PARTITION COMPUTED WITH P = 80.00 %.

 REQ:
HCM3
 BEST PARTITION MEASURE:    0.619
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1    ( 7)   1   5   2   3   4   6   7

 REQ:
EVAL

 STRENGTH:  0.6190,
 COUPLING:  0.0000,
 MEASURE:   0.619.
```

```
 REQ:
REST
 ENTER FILE NAME:
SOSA12

 ADJACENCY MATRIX READ FROM FILE SOSA12

 REQ:
DENO
;
/1,8,12,16,17,33,39,41,52,53,61,2,3,6,7,11,13,15,18,19,20,21,22,
;
25,40,43,54,59,4,5,9,10,26,27,28,29,30,31,46,47,51,55,56,58,

 14,23,24,57,32,34,35,36,38,37,60/

   THE FOLLOWING NODES HAVE BEEN REMOVED:
     1    2    3    4    5    6    7    8    9   10
    11   12   13   14   15   16   17   18   19   20
    21   22   23   24   25   26   27   28   29   30
    31   32   33   34   35   36   37   38   39   40
    41   43   46   47   51   52   53   54   55   56
    57   58   59   60   61

   NODES HAVE BEEN RENAMED AS FOLLOWS:
   OLD NO.  NEW NO.
   ---------------
     42        1
     44        2
     45        3
     48        4
     49        5
     50        6
```

```
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
SIMA

 SIMILARITY MATRIX COMPUTED.

 REQ:
INPA
 ENTER PERCENTAGE PARAMETER:
80.

 INITIAL PARTITION COMPUTED WITH P = 80.00 %.

 REQ:
HCM3
 BEST PARTITION MEASURE:    0.400
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1    ( 6)   1   2   3   4   5   6
```

```
 REQ:
REST
 ENTER FILE NAME:
SOSA12

 ADJACENCY MATRIX READ FROM FILE SOSA12

 REQ:
NNPA
:
/1,8,12,14,17,33,39,41,52,53,61/
:
2,6,15,19,21,25,43,54/
:
3,7,11,13,22,59/
:
18,20,40/
:
14,23,24,57/
:
32,34,35,36,37,38,60/
:
42,44,45,48,49,50/
:
4,5,9,10,26,27,28,29,31,46,47,51,56/
:
30,55,58/$

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 ____________________

    1     (11)    1    8   12   16   17   33   39   41   52   53
                 61
    2     ( 8)    2    6   15   19   21   25   43   54
    3     ( 6)    3    7   11   13   22   59
    4     ( 3)   18   20   40
    5     ( 4)   14   23   24   57
    6     ( 7)   32   34   35   36   37   38   60
    7     ( 6)   42   44   45   48   49   50
    8     (13)    4    5    9   10   26   27   28   29   31   46
                 47   51   56
    9     ( 3)   30   55   58
```

```
 REQ:
FRLK

 LINKS  BETWEEN  CLUSTERS    1 &    2 :
                  1  -   15
                 16  -    6
                 39  -    2
                 39  -   21

 LINKS  BETWEEN  CLUSTERS    1 &    3 :
                 61  -   59

 LINKS  BETWEEN  CLUSTERS    1 &    4 :
                  1  -   18

 LINKS  BETWEEN  CLUSTERS    1 &    5 :
                  8  -   14
                 12  -   57

 LINKS  BETWEEN  CLUSTERS    1 &    6 :
                  8  -   32
                 12  -   32
                 16  -   35
                 17  -   35
                 33  -   32
                 33  -   34
                 33  -   35
                 33  -   37
                 39  -   32
                 39  -   34
                 39  -   37
                 52  -   37

 LINKS  BETWEEN  CLUSTERS    1 &    7 :
NONE.

 LINKS  BETWEEN  CLUSTERS    1 &    8 :
                  1  -    4
                  1  -    5
                  1  -   51
                  8  -    9
                 12  -   28
                 17  -   51
                 39  -    9
                 39  -   27
                 53  -   56

 LINKS  BETWEEN  CLUSTERS    1 &    9 :
                  1  -   30
                  8  -   30
                  8  -   55
                 39  -   30
```

LINKS BETWEEN CLUSTERS 2 & 3 :
2 - 7
2 - 22
6 - 22
19 - 59
43 - 11
54 - 7
54 - 11

LINKS BETWEEN CLUSTERS 2 & 4 :
2 - 18
19 - 40
21 - 18

LINKS BETWEEN CLUSTERS 2 & 5 :
15 - 14
54 - 14
54 - 57

LINKS BETWEEN CLUSTERS 2 & 6 :
NONE.

LINKS BETWEEN CLUSTERS 2 & 7 :
2 - 42
6 - 50
21 - 42
21 - 50
25 - 44
25 - 45
43 - 42

LINKS BETWEEN CLUSTERS 2 & 8 :
2 - 9
6 - 5
6 - 9
19 - 9
54 - 56

LINKS BETWEEN CLUSTERS 2 & 9 :
25 - 55
54 - 55

LINKS BETWEEN CLUSTERS 3 & 4 :
13 - 40
22 - 18
59 - 40

LINKS BETWEEN CLUSTERS 3 & 5 :
59 - 23

LINKS BETWEEN CLUSTERS 3 & 6 :
NONE.

```
LINKS BETWEEN CLUSTERS   3 &   7 :
                 7 -  42
                 7 -  45

LINKS BETWEEN CLUSTERS   3 &   8 :
                59 -  31

LINKS BETWEEN CLUSTERS   3 &   9 :
                 3 -  58

LINKS BETWEEN CLUSTERS   4 &   5 :
                40 -  23
                40 -  24

LINKS BETWEEN CLUSTERS   4 &   6 :
                40 -  32

LINKS BETWEEN CLUSTERS   4 &   7 :
NONE.

LINKS BETWEEN CLUSTERS   4 &   8 :
NONE.

LINKS BETWEEN CLUSTERS   4 &   9 :
                40 -  30

LINKS BETWEEN CLUSTERS   5 &   6 :
                23 -  32
                23 -  34
                24 -  32
                24 -  34

LINKS BETWEEN CLUSTERS   5 &   7 :
NONE.

LINKS BETWEEN CLUSTERS   5 &   8 :
                23 -   4
                23 -   5
                23 -  31
                24 -   4
                24 -   5
                24 -  31

LINKS BETWEEN CLUSTERS   5 &   9 :
                57 -  58

LINKS BETWEEN CLUSTERS   6 &   7 :
NONE.
```

```
LINKS BETWEEN CLUSTERS    6 &    8 :
                32 -    5
                32 -    9
                32 -   28
                60 -    4
                60 -   51

LINKS BETWEEN CLUSTERS    6 &    9 :
NONE.

LINKS BETWEEN CLUSTERS    7 &    8 :
                42 -    9
                42 -   46
                42 -   47
                45 -   47

LINKS BETWEEN CLUSTERS    7 &    9 :
NONE.

LINKS BETWEEN CLUSTERS    8 &    9 :
                26 -   55
                27 -   30
                28 -   30
                56 -   30
                56 -   55

REQ:
```

APPENDIX F

Main Subproblems Resulting From The First Iteration of The Decomposition Analysis

Note: (11) The number in parenthesis indicates the number of interdependicies identified for the requirement.

Main Subproblem 1: Multi-programming Support Functions:

5 (11): Operating system must provide a multi-programming environment.

12 (7): Operating system must protect user jobs from each other.

16 (5): System process routines are re-entrant and shared.

20 (4): Jobs are scheduled strictly on a first-come, first-served basis.

21 (5): Job scheduling function must be modularized so that improvements to the system can be easily accomplished.

37 (9): Device handler routines must support multiple job streams from card readers.

43 (12): P-V mechanism must be provided.

45 (1): P-V operations are available only to system processes.

56 (3): Operating system must accept input data from the user's job stream.

57 (2): Supervisor process must load the user's program.

65 (4): There exists one supervisor process per job stream.

Main Subproblem 2: Process Management Functions:

M5 2A: Process Creation and Scheduling.

6 (14): The operating system must be process oriented.

10 (9): A process must be ready to run prior to being allocated to a processor.

19 (5): Initially one process is created for each user's job.

23 (7): Ready processes are scheduled in simple round-robin fashion.

25 (8): A process shall be blacked while awaiting synchronization with another process.

29 (6): Reference to a process is by symbolic name.

47 (4): The message facility must be accessible to all processes.

58 (10): The user process may dynamically create and destroy other user processes.

MS 2-B Process/Operating System Interface:

7 (4): The operating system must run a machine that has two states.

11 (8): User communication with the operating system is via SVC instruction.

15 (6): SVC instructions are user callable.

17 (2): SVC instructions are executed in the supervisor state.

26 (6): A process shall be blocked when it specifically relinquishes control to the process scheduler.

63 (8): User processes must schedule completion.

MS 2-C Process Time-Slicing:

27 (6): The traffic controller must time-slice CPU usage to achieve multi-programming.

24 (2): A process shall be blocked when a timer run-out trap is detected.

44 (10): An interrupt handler must be provided.

Main Subproblem 3: Resource and Memory Management Functions:

MS 3-A Resource Allocation:

8 (10): All resource requests must pass through the supervisor.

9 (8): System resources must be allocated to a job, prior to the job being made eligible to run.

13 (12): The operating system must utilize information tables to monitor and control processing.

14 (4): System tables can be dynamically allocated and released.

30 (9): The operating system must allocate memory for job partitions the size of which is specified by the user.

31 (7): Memory is allocated in 2K blocks.

32 (11): Operating system must dynamically allocate memory for itself.

33 (4): Memory is allocated using a best-fit algorithm.

35 (7): Free storage areas are collapsed into continguous blocks of memory whenever a

partition is freed.

50 (2): Messages are of an arbitrary yet specified length.

51 (5): Any number of messages may be queued.

60 (9): User processes cannot dynamically allocate memory.

MS 3-B Protection:

34 (9): Memory must be protected to prevent the simultaneous allocation of a partition to multiple jobs.

59 (7): Dynamically created processes must run in the same partition as the parent job.

62 (4): The user processes run in the problem state.

Main Subproblem 4: Supervisor Process:

18 (6): Supervisor process must create and delete the environment in which a job runs.

27 (10): Supervisor routine must reclaim all system resources when a job has completed.

28 (9): Supervisor process must reclaim all system resources when an error condition abnormally terminates a job.

61 (6): User cannot destroy system process within the same process group.

Main Subproblem 5: Device Management Functions:

36 (16); Operating system must supply a device manage-

ment routine.

38 (10): All devices are allocated.

39 (9): Device handler routine supports one card reader/input stream.

40 (6): Device handler must support one line printer/ output stream.

41 (8): I/O devices operate via multiplexor channel.

42 (7): The user can provide his own routines for non-standard devices.

64 (6): The user's job can reference 1 input, 1 output, 1 exceptional type of device.

Main Subproblem 6: Message Facility:

46 (12): A message facility must be provided.

48 (5): The name of the sending process must be prefixed to a message.

49 (7): The receiving process must read the name and text of a message.

52 (2): All messages are released when a process terminates.

53 (4): The receiver of a message may destroy the message without acknowledgement.

54 (5): If no messages are available to a process which expects one, it gets blocked.

APPENDIX G

Final Requirements Definition

I. Design: Philosophy

1. The operating system must be simple, implementing a basic system nucleus.

   DEFINITION: The operating system is to be simple in the sense that it is to implement only those features most essential for learning the fundamentals of the operating system. Therefore, the system is to implement a basic system nucleus to include the following features:

   --- Multi-programming;

   --- Basic multi-programming support;

   --- Dynamic memory allocation;

   --- Device management;

   --- Simple top level supervisor; and

   --- Traffic control.

   IMPLICATIONS FOR DESIGN: The nucleus does not include the following:

   --- language processors;

   --- utility programs;

   --- spooling;

   --- file systems;

   --- application packages;

   --- debugging facilities; and

   --- subroutine libraries.

2. The operating system must be designed as a pedagocial tool.

DEFINITION: Since the operating system is to be used as an instructional tool, simplicity and easy identification of the major functions are the objectives of the design. As previously described, the heirarchical operating system structure enables:

--- easy identification of the relevant sections for processor management, memory management, and device management; and

--- identification of the well-defined interfaces between the various functional section.

IMPLICATIONS FOR DESIGN: The design concepts of extended machine instructions and heirarchical operating system structure have been selected; as the optimum method of satisfying the design objective.

Also the pedagocial clarity of the operating system is preferred to performance.

3. The operating system must be process oriented.

DEFINITION: The requirement is vague as it stands, yet it recognizes the fact that there are certain requirements necessary to support a process. The following entities exist within the system:

--- job stream: sequential;

--- job: collection of activities needed to do the work required;

--- process group: processes belonging to the same job; and

--- process: a system-created entity which is the smallest computational entity with which the system must deal.

IMPLICATIONS FOR DESIGN: Therefore, the operating system must provide certain basic functions by the extended machine including:

--- P-V operations;

--- basic multi-processing support; and

--- traffic controlling.

The software functions can be thought of as being executed in the same way as hardware instructions. Again, the basic functions represent what the operating system must accomplish; the extended machine implements the requirements.

II. Design Constraints

4. The operating system must be small; occupying fewer than 2500 cards of assembly language statements.

DEFINITION: It was not clear from the system description that the requirement occurred "post hoc, ergo propter hoc".

If, in fact, this was a design constraint then it must be analyzed in conjunction with the

requirements for simplicity and a basic nucleus. Clearly, adding more simplistic capabilities to the system increases the number of assembly language statements and at some point, would conflict. It was assumed that since the actual operating system deck 2500 that this requirement was not significant.

5. The operating system is to be implemented utilizing IBM System/360 hardware.

   DEFINITION: This simple requirement has far-reaching significance for the design; specifically, the hardware constraint has implications for the following functions:

   --- IBM/360 is a two state machine;
   (problem, supervisor states identified)

   --- Protection is provided in 2K blocks;
   (protection must be provided to match memory, allocation is in 2K blocks)

   --- Interrupt mechanisms are hardware functions which dictate what sort of interrupts are recognized and how they are processed.

   IMPLICATIONS FOR DESIGN: Since the guidelines for defining requirements called for independence among requirements, it was not clear if the implication of the constraint needed to be stated explicitly as requirements.

Since the design constraints were not assessed with the remaining design requirements, it was decided to draft the implications of the design constraint and to include these in the assessment process.

6. The input/output devices are limited to card reader for input job streams, and line printers for output.

   DEFINITION: This was a design constraint, imposed "a priori", which limits both the flexibility and complexity of the operating system.

   IMPLICATIONS FOR DESIGN: This requirement reduces the variety of hardware and, therefore, the scope of the device management functions of the operating system. The impact of the requirement is specifically written into subsequent requirements.

III. Design Requirements

7. The operating system must provide for a multi-programming environment.

   DEFINITION: Multi-programming - multiple job streams from different sources.

   IMPLICATIONS FOR DESIGN: The operating system must have the facilities for:

   --- input stream interpretation - those

functions which delineate jobs and job steps;

--- job control - those functions of the operating system which control the processing of a job in the system; and

--- job scheduling - those functions which prepare a job for execution.

Limitations on Multi-programming: There must be some sort of a limit established for the number of jobs that the operating system can handle. In fact, the system is limited by:

1. 15 protection keys;
2. the number of I/O stream must equal the number of devices; and
3. the amount of memory available.

8. The operating system must run on a machine that has two distinct states.

DEFINITION: The two states are problem state and supervisor state. This requirement implies first that user programs execute in the problem state, and second, a processor can correctly execute privileged instructions only in the supervisor state. Privileged instructions include requests to:

--- change the state of the machine;

--- start I/O;

--- change the protection rights of memory; and

--- change the interrupt states of the machine.

Since the operating system includes the implementation of the extended machine concept, these instructions may take advantage of the dual state machine by making system routines unavailable to the user and, therefore, only certain selected routines are user callable.

IMPLICATIONS FOR DESIGN: Therefore, the operating system must have the capability to:

--- distinguish machine state;

--- identify privileged instructions; and

--- identify user-callable extended machine instructions.

9. All resource requests must pass through the supervisor process.

DEFINITION: The supervisor routine is a top-level process that establishes the environment in which a job will execute. Initially, all resources required by a given job are stated explicitly on JCL cards. The supervisor routine coordinates requests for resources prior to creating a process for the job.

IMPLICATIONS FOR DESIGN: The tasks which the supervisor must perform are as follows:

--- allocate memory;

--- allocate devices required;

--- read the user deck into his partition;

--- start user process; and

--- upon completion, reclaim all resources.

10. System resources must be allocated to a job prior to the job being made eligible to run.

    DEFINITION: The specific resources consist of memory and input/output devices.

    IMPLICATION FOR DESIGN: These resource allocations are made at a job level. There are other resources which are allocated at the process level.

11. A process must be ready to run prior to being allocated a processor.

    DEFINITION: Resources required at the process level consist of only the processor.

    IMPLICATION FOR DESIGN: Since resource allocations are made at the process level, there must be a traffic controller routine to create a process-oriented environment and the system must have some means of determining when a process is not eligible to run.

12. User communication with the operation system is via special call.

    DEFINITION: What need has the user of communicating with the operating system? Once all the

resources are allocated, must there by any communication? These questions require the user to communicate with the operating system:

--- create a process;

--- destroy a process;

--- halt job and signal supervisor;

--- find a PCB given its name;

--- read a message;

--- send a message;

--- start/stop process; and

--- abnormally terminate the job.

IMPLICATIONS FOR DESIGN: The operating system must take action based on the user requests.

13. The operating system must protect user jobs from each other.

DEFINITION: Protect in this sense means to prohibit unauthorized access to memory locations.

IMPLICATIONS FOR DESIGN: For purposes of this system, a separate supervisor process exists in a separate process group for each job stream. There is no communication between process of different jobs; therefore, they essentially are invisible to each other.

14. The operating system must utilize information tables to monitor and control processing.

DEFINITION: The operating system must maintain

information on a varying number of jobs, processes, and resources. This requirement attempts to identify the table and thereby minimize proliferation and redundancy of system information.

IMPLICATIONS FOR DESIGN: The following information tables exist in the sample operating system:

--- nucleus databases;

--- process control block - one per process containing save areas, used by the system routines for storing the status conditions, and semaphores;

--- memory - free storage blocks;

--- processor management - message facility; and

--- device management - unit control block stored in a permanently allocated area for every unit.

Notice that as previously stated, the emphasis for all transactions is at the process level; therefore, the process control block contains most of the system information.

15. System tables can be dynamically allocated and released.

DEFINITION: System tables refer to those tables built and maintained by certain system processes. These tables include:

--- process control block;

--- semaphores;

--- free storage block;

--- message; and

--- unit control block.

IMPLICATIONS FOR DESIGN: The operating system must be capable of dynamically allocating memory to itself for these tables.

A possible deadlock could occur at the point of a user's program which consumed all of memory; namely by continually writing messages. The system has no built-in limiting functions to identify such overrun conditions.

16. Certain system routines are user callable.

DEFINITION: The nucleus routines are the SVC instructions of the extended machine concept. Some of the routines allow unrestricted memory reference and, therefore, are not available to the user.

IMPLICATIONS FOR DESIGN: When an SVC instruction is issued, the handler routine must check to see if the operation requested is, in fact, user-callable.

17. System process routines are re-entrant and shared.

DEFINITION: System process have only one copy resident in the system. Therefore, they must be efficiently shared in a multi-programming environment. Pure procedure operates only on variables in registers or in separate data segments associated with the job.

IMPLICATIONS FOR DESIGN: The need for pure procedure is driven by the need for a multi-programming environment. The system can set locks through the P-V operations to prevent race conditions.

18. Extended machine instructions are executed in the supervisor state.

DEFINITION: The extended machine instructions along with the normal hardware instructions, comprise the nucleus of the system. SVC handler is used to activate the extended machine instruction and transfer between loads.

IMPLICATIONS FOR DESIGN: When an SVC instruction is issued, a supervisor call interrupt occurs and control is transferred to SVC handler routine. Therefore, an SVC handler interrupt must be provided.

19. The supervisor process must schedule jobs and prepare the jobs for execution.

DEFINITION: The supervisor routine initially

creates a process. The user may generate his own processes by SVC instructions during execution of his process group. This requirement deals only with establishing the USER PROG and not with specific resource allocations.

IMPLICATIONS FOR DESIGN: This requirement is an explicit statement of one of the functions of the supervisor process. The following instructions apply:

--- a non-system process cannot stop a system process; and

--- a process must be stopped prior to its being deleted.

20. Initially one process is created for each user's job.

DEFINITION: One process is created by the supervisor process after all job level resources have been allocated to the job.

IMPLICATIONS FOR DESIGN: The user must create any additional processes desired on his own.

21. Jobs are initiated strictly on a first-come, first-served basis.

DEFINITION: Jobs are read into the system in the form of job streams from card readers. Jobs are accepted into the system as long as sufficient resources exist. Since there is no

spooling capability, a job cannot be copied in the system. Once in the system, user jobs are redefined in process groups which contend for the processor in a multi-programming environment.

IMPLICATIONS FOR DESIGN: The supervisor process must determine if it can schedule a job before reading it into the system.

22. The supervisor process must be modularized so that improvements to the system can be easily accomplished.

DEFINITION: The system description indicated that sophistication of job scheduling is limited by the brevity of the implementation. Therefore, the system could easily be extended to provide more advanced features and facilities. Modularization of the function was critical, not for pedagogical clarity, but to provide for system improvements.

IMPLICATIONS FOR DESIGN: Although modularized design was emphasized as a design philosophy for pedagogical clarity, it is now emphasized to allow easy improvement. This function should be designed incorporating interface features easily adaptable to a system which will implement advanced features such as spooling.

23. The process scheduler must time-slice CPU usage among ready processes to achieve multi-programming.

DEFINITION: Traffice controller resides in the process management, lower level, and enables a process to run until a certain time quantum has elapsed; at which time, the process is stopped and another started. A process is ready when it is not blocked or waiting for the completion of some external event such as I/O operation or for a message from another process.

IMPLICATIONS FOR DESIGN: The traffic controller schedules ready process in a round-robin fashion. Interrupts must be enabled to identify when a process:

--- exceeds its time quantum;

--- becomes blocked; and

--- relinquishes control to the traffic controller to await the completion of an external event.

24. Ready processes are scheduled in simple round-robin fashion by the process scheduler.

DEFINITION: Round-robin scheduling means that processors are sequentially scanned until a ready process is found.

IMPLICATIONS FOR DESIGN: The traffic controller must maintain a current list of processes from which to select the next ready process.

25. A process must be blocked and control released to the process scheduler when a time quantum of 50 ms is exceeded.

DEFINITION: Timer runout trap must be indicated when a process exceeds its time quantum. By blocking a process is meant that it is ineligible to run temporarily.

IMPLICATIONS FOR DESIGN: Interrupt mechanism must be provided to detect a time runout.

26. A process shall be blocked and control passed to the process scheduler when the process must wait for synchronization with another process.

DEFINITION: Multiple process creation may require that one process await the completion of a previous process in order to run.

IMPLICATIONS FOR DESIGN: Some mechanism (basic primitives) must be provided for the synchronization of processes.

27. A process shall be blocked and control passed to the traffic controller when the process specifically relinquishes control to the process scheduler.

DEFINITION: A user process may actually finish

execution and relinquish control to the traffic controller.

IMPLICATIONS FOR DESIGN: User process must signal termination stop process instructions, or abnormal termination.

28. The supervisor process must reclaim all system resources from a job when the job has completed.

DEFINITION: Reclaimation of resources is accomplished on a job level, since processes only gain the use of a processor.

IMPLICATIONS FOR DESIGN: This requirement implies successful completion of a job; there are such things as unsuccessful completions. The supervisor must at this point:

--- print a message on the printer;

--- destroy all processes created for or by the user job;

--- free memory partition; and

--- move on.

A message must be available to signal successful completion.

29. The supervisor process must reclaim all system resources when an error condition is caused which terminates processing for a process.

DEFINITION: An error in one user process which reaches the supervisor level, is capable of

terminating processing for the entire process group.

IMPLICATIONS FOR DESIGN:

--- certain error conditions must be defined, remember that this system does not have debugging facilities;

--- the supervisor must perform the same functions as in the previous requirement; and

--- an error message must be provided.

30. Reference to processes within a process group is by symbolic name.

DEFINITION: In order to communicate back and forth user processes must be able to identify each other. Therefore, each process is given a name by the process that creates it.

IMPLICATIONS FOR DESIGN:

--- each process must be named by the process creating it; and

--- each process must have a unique name field in order to identify it.

31. The operating system must allocate memory for a job, the size of which is to be specified by the user.

DEFINITION: The operating system provides routines that will allocate a block of memory of a given size and given address alignment

using a best-fit algorithm.

IMPLICATIONS FOR DESIGN:

--- user must specify the job partition size required by JCL;

--- the operating system must maintain a list of storage areas, accomplished using free storage block list; and

--- a queue is established for those jobs awaiting memory.

32. Memory is allocated to a job in contiguous 2K blocks.

DEFINITION: Partitioned allocation for user's jobs is a simple memory requirement scheme which facilitates multi-programming. A block is a uniquely named group of words whose addresses are contiguous.

IMPLICATIONS FOR DESIGN:

--- the user must specify memory requirements in increments of 2K; and

--- the operating system should allocate memory such that the amount of wasted memory is minimized.

33. The operating system may dynamically allocate memory to itself for temporary work space or traffic areas for system processes.

DEFINITION: All system tables and system processes which do not run in the user's process need temporary memory allocated to them. Dynamic memory allocation means that partitions are created as required during processing. The operating system may use these areas for:

--- work space for system processes; or

--- temporary buffer areas for message storage.

IMPLICATIONS FOR DESIGN: Tables must be maintained, testing free and allocated storage areas, usually using a chaining method to facilitate the dynamic nature of allocation scheme.

34. Memory is allocated using a best-fit algorithm.

DEFINITION: The memory allocation algorithm cycles through a free storage list, which is arranged in ascending order, until it finds a block large enough to contain the requested area. In order to minimize breakage, the allocated area with the specified alignment is selected as close to the beginning of the block as possible.

IMPLICATIONS FOR DESIGN:

--- excess memory is re-linked to a free storage list whenever memory is allocated; and

--- a free storage list, arranged in ascending order, must be available in order to

accomplish the best-fit scheme.

35. Memory must be protected to prevent the simultaneous allocation of a partition to multiple jobs.

DEFINITION: Memory protection is a hardware function in the IBM System 360. Each partition is assigned a protection key (1 through 15). The "0" key is reserved for the operating system. Since the hardware actually associates the keys with each 2K byte block of memory, partitions must be multiples of 2K, and all locks within a partition are set to the same value. Access control functions are those functions which protect an area of storage against unauthorized access by:

--- insuring that all storage references by an executing task for the purpose of writing, executing and/or reading in that storage are are legal; and

--- provides a task from modifying areas of main storage beyond the limits.

IMPLICATIONS FOR DESIGN:

--- protection keys must be assigned and set when memory is allocated; and

--- partition locks must be tested prior to allowing access to memory.

36. Free storage areas are collapsed into contiguous blocks of memory whenever a job partition is freed.

DEFINITION: Since memory is allocated in contiguous blocks, the operating system must recombine memory partitions and update its list of free areas.

IMPLICATIONS FOR DESIGN: Memory is to be reconfigured and the list of free space updated and re-ordered whenever a partition of memory is freed.

37. The operating system must supply a device management system which runs as a separate process, one per device.

DEFINITION: The device management system:

--- provides the routine necessary to issue the I/O commands;

--- monitors the I/O devices; and

--- interprets the status information when an I/O interrupt occurs. It must also maintain interfaces to process management interrupt handlers and event monitoring functions.

IMPLICATIONS FOR DESIGN:

--- management system can use semaphores as locks against two processes simultaneously

attempting to access the same device; and

--- the fielding and handling of input/output interrupts are performance by a special routine that is involved whenever an I/O interrupt occurs. It runs for a very short time, just long enough to store status information and perform a V operation on Wait-Semaphore.

38. Device handler routines must support multiple job streams from card readers.

DEFINITION: Support means that the routines must distinguish among:

--- job control cards;

--- object deck;

--- data cards;

and to delineate jobs and job steps.

IMPLICATIONS FOR DESIGN: Each card reader represents an input job stream.

39. A device is dedicated to a job.

DEFINITION: A dedicated device is allocated to a job for the job's entire duration; this is especially applicable to card readers and printers. Allocation is made by the supervisor during job definition.

IMPLICATIONS FOR DESIGN:

--- a card reader represents an input job stream;

--- a line printer must be allocated to a job prior to the job being made eligible to run.

40. The device handler routine supports one card reader per input stream.

DEFINITION: I/O that can be processed sequentially to terminate an I/O stream. A single card reader then is used to read in an entire job stream.

IMPLICATIONS FOR DESIGN: The system can continue to accept jobs as long as sufficient responses are allocatable. As soon as we reach the resource limit we must stop reading in jobs.

Therefore, the supervisor process must allocate resources as jobs are being used in in order limit the number of jobs at the appropriate time.

The user must specify a name for his input stream on JCL.

41. The device handler routine must support one line printer per output stream.

DEFINITION: The user may specify a certain output device in his JCL.

IMPLICATIONS FOR DESIGN: The device name for output must be specified in JCL.

42. The user must provide his own routines for non-standard devices.

DEFINITION: The user may supply his own routine to issue his own I/O commands.

IMPLICATIONS FOR DESIGN: The user must indicate the use of a non-standard device in his JCL statements. The device handler process must supply a routine to handle the interface for devices wherein the user wishes to provide his own I/O commands.

43. A process synchronization mechanism must be provided to serve as a lock on a database.

DEFINITION: The process synchronization mechanism is the P-V operations used in conjunction with semaphore.

--- P operation - of value >0 then value = Value-1
if value $\leq$ 0 then value=value-1
and the process is ineligible
or blocked.

--- V operation - if No processes are ineligible
then value=value+1
if there is a process ineligible
then value=value+1
and the waiting process is
eligible.

--- Applications - the semaphore when the initial value=1 can serve as a lock by

requiring a P-operation before accessing and a V-operation afterwards, can insure integrity of a resource.

IMPLICATIONS FOR DESIGN: P-V operations can be used to provide protection for databases.

44. A process synchronization mechanism must be provided for the timing of synchronous processes.

DEFINITION: For processes which require synchronous processing, the P-V operations can be used to insure that such synchronization takes place.

IMPLICATIONS FOR DESIGN: Since P-V operations are available only to system process, this technique may be used to insure that system processes run in sequential order.

45. A process synchronization mechanism must be provided for synchronization between the sender and receiver in message processing.

DEFINITION: A message facility is available to all processes for interprocess communication. The P-V operations can be employed by the message facility to insure that messages are synchronized and queued.

IMPLICATIONS FOR DESIGN: The P-V operation can be used to establish a message queue facility.

46. A process synchronization mechanism must be provided to lock a device.

DEFINITION: All devices are dedicated, one per job. The P-V operation can be used to lock each device.

IMPLICATIONS FOR DESIGN: The P-V operation can be used to lock devices.

47. An interrupt handler routine must be provided for I/O interrupts.

DEFINITION: An interrupt is an occurrence that causes the processor to take some immediate action. The IBM System/360 has a mechanism for being interrupted, saving its status, determining what general class of interrupt has occurred, and executing an appropriate interrupt handler routine.

IMPLICATIONS FOR DESIGN: The interrupt handler determines the cause of the following faults and calls the appropriate operating system function. In this case, it calls the I/O interrupt handler.

48. An interrupt handler routine must be provided for program interrupts.

DEFINITION: Program interrupts consist of interrupts employed within the program structure to enable a synchronous processing.

IMPLICATIONS FOR DESIGN: This facility is available only to system processing and must be provided for that purpose.

49. An interrupt handler must be provided for supervisor call interrupts.

DEFINITION: Supervisor call interrupts are required to recognize SVC instructions. This mechanism is used to activate the extended machine instructions and to transfer between levels of the system.

IMPLICATIONS FOR DESIGN: The operating system must include a supervisor call handler.

50. An interrupt handler must be provided to deal with external interrupts.

DEFINITION: External interrupts are generated outside of the operating system due to external conditions; specifically, timer runout trap.

IMPLICATIONS FOR DESIGN: The operating system may utilize the timer function to provide for a multi-programming environment.

51. P-V Operations are available only to system processes.

DEFINITION: Since the P-V operations in effect control the synchronization of the operating system and lock various resources, they are available only to operating system processes for use.

IMPLICATIONS FOR DESIGN: User processes must have another mechanism available to synchronize their processing.

52. A message facility must be provided to all processes.

DEFINITION: The message facility must be available for interprocess communication to all processes in the system.

IMPLICATIONS FOR DESIGN: User processes must be identifiable by name. The message facility must recognize:

--- a sender;

--- a receiver;

--- the size of the message; and

--- the text.

The message facility must be able to queue up messages to a given process, uses memory management for message buffers, uses P-V operations to synchronize message flow.

53. The process receiving a message must be able to determine the originator of the message.

DEFINITION: The receiver of a message must be able to determine from whence it came.

IMPLICATIONS FOR DESIGN: A process may be kept waiting for a message from another process, as a means of synchronization.

54. The receiving process may read the name and text from the originator.

DEFINITION: In order to respond to a message the receiver must be able to verify that it is the correct message from the correct process. In order to take action on a message the receiver must be

able to read the message.

IMPLICATIONS FOR DESIGN: The receiver must have the capability to read the name of the originator and the text of the message, but this does not imply that the message must, in fact, be read.

55. Messages are of an arbitrary, but specified length.

DEFINITION: The message facility must allow for a valuable message size.

IMPLICATIONS FOR DESIGN: The message queue must be dynamically allocated space since the number and size of messages is variable. Note that no limit is specified for the number of messages.

56. Any number of messages for a given process may be queued while waiting to be read by the process.

DEFINITION: A process can have a varying length queue of messages waiting to be read.

IMPLICATIONS FOR DESIGN: Each process has a variable length message queue which is dynamically allocated.

57. All messages, enqueued for a given process to read, are released when that process terminates.

DEFINITION: When a process terminates, all messages waiting to be read are freed.

IMPLICATIONS FOR DESIGN: This is performed within the destroy process SVC by freeing memory used to store the messages.

58. Messages are not receiptable for, from receiver to sender.

DEFINITION: The receiver of a message does not have to acknowledge receipt of any message to the sender.

IMPLICATIONS FOR DESIGN: If the message facility cannot locate the process for which the message was intended an error condition is caused.

59. If no messages are available to a process which expects one, it may go blocked.

DEFINITION: The message facility can be used for process synchronization; therefore, a process is blocked until properly synchronized.

IMPLICATIONS FOR DESIGN: The user has a mechanism for the synchronization of various processes.

60. User programs utilize a job control language statement to specify resource requirements.

DEFINITION: Job Control Language is the means by which a user specifies and quantifies his resource requirements to the operating system. For the purposes of the sample operating system, the simplified JCL must specify:

--- memory size required;

--- name of input device type;

--- name of the output device type; and

--- non-standard device for which the user will supply his own handler routine.

IMPLICATIONS FOR DESIGN:

--- JCL card is used to delineate job boundaries;

--- It must be the first card of the deck so that resource requirements may be determined.

61. The operating system must accept input data from the user's job stream.

DEFINITION: The user may input data to be read and used in execution of the object deck.

IMPLICATIONS FOR DESIGN: The supervisor must be capable of distinguishing among JCL, object deck, and data cards for any job.

62. The supervisor process must load the user-supplied object deck into the user area of memory.

DEFINITION: Once the supervisor has allocated the resources required for the user's job, the user's object deck is read into his partition.

IMPLICATIONS FOR DESIGN: This is a function of the supervisor process.

63. All processes may dynamically create additional processes.

DEFINITION: The user has the SVC instructions available to him which allows the creation of additional processes.

IMPLICATIONS FOR DESIGN: The user processes run in the same partition and state as the initially created user process. The user may destroy only user created processes.

64. Dynamically created processes run in the same memory area as the parent job.

DEFINITION: Dynamically created processes must share the memory partition allocated to the parent job and have the same protection attributes assigned.

IMPLICATIONS FOR DESIGN: Dynamically created user processes must be identifiable and are protected from other jobs in the same manner as is the parent job.

65. User processes cannot dynamically allocate memory.

DEFINITION: This is directly implied by #59. Since user created processes run in the partition of the parent job, no more memory is needed. However, some people will attempt to get more memory than they can use.

IMPLICATIONS FOR DESIGN: The user must specify the memory requirements of the entire job, including dynamically created processes, once and be satisfied with it. Attempting to exceed the user's memory partition will generate an error.

66. User processes can destroy other user processes only within the same process group.

DEFINITION: System processes are created for the use of the operating system and must be maintained. These processes consist of supervisor process and device handler process.

IMPLICATIONS FOR DESIGN: System processes must be identifiable and protected from user destruction. The user destroys a process by unlinking the PCB, system processes do not have a specified PCB.

67. User processes run in the problem state.

DEFINITION: The problem state is one of two states defined by the IBM System/360.

IMPLICATIONS FOR DESIGN: System processes are protected from user violation and/or destruction by the two state machine concept.

68. The user process must signal completion (successful or unsuccessful) to the operating system.

DEFINITION: A completion signal; i.e., stop process, is required so that:

--- traffic controller may schedule a process; and

--- supervisor process may reclaim system resources at the end of a job.

IMPLICATIONS FOR DESIGN:

--- user processes may only stop user processes;

--- a process must be stopped before it is destroyed.

69. The user's job can reference at most: 1 input device, 1 output device, 1 non-standard devices.

DEFINITION: The operating system will allow references to only one each of the three degrees types.

IMPLICATIONS FOR DESIGN: I/O commands operate as streams unless otherwise specified by the user in

the handling of exceptional devices.

70. There is one supervisor process per job stream.

DEFINITION: The supervisor process must schedule all jobs and prepare them for execution by calling other appropriate modules of the system. Functions of the supervisor process include:

--- determines the amount of memory required;

--- set storage protection keys;

--- starts a process in an interface routine for each device;

--- reads in the user's object deck;

--- user process starts to run; and

--- upon completion, the supervisor process destroys all processes created for or by the user frees memory and devices.

IMPLICATIONS FOR DESIGN: The supervisor process acts as the interface between the user and the operating system.

71. The I/O interrupt handler routine must provide for a synchronous scheduling of a process requiring fast processing.

DEFINITION: The interrupt mechanism transfers control to the traffic controller causing the process waiting for the interrupt to start running immediately. It is, therefore, possible to attain very fast processing of exceptional interrupts.

IMPLICATIONS FOR DESIGN: Interrupt routine trans-

fers control directly to the traffic controller in order to run a new process.

72. System Initialization: The operating system must include a non-system resident task which loads the O/S into the computer and defines the processing environment.

DEFINITION: Initial program load routine runs free of most of the rest of the system, and serves to initialize supervisor process and SVC routines, essentially by initializing PCB entries and free storage blocks for memory.

IMPLICATIONS FOR DESIGN: This system is used infrequently and depends heavily upon the final implementation design in order to carry out its functions.

APPENDIX H

Final Interdependency Assessment Results

Note 1: (s) Indicates that the requirement indicated supports the implementation of the requirement being assessed.

(c) Indicates that the requirement indicated conflicts with the implementation of the requirement being assessed.

Note 2: Requirements 1 through 6 were not assessed for the reasons stated in 4.1.10.

7. The operating system must provide for a multi-programming environment.

   10(s): Resource allocation is performed as a job is read into the system, except for process allocation.

   13(s): A multi-programming environment must include job protection mechanism.

   14(s): Information tables are the mechanism by which the operating system monitors and controls the multi-programming environment.

   17(s): The need for pure procedures is driven by a multi-programming environment.

   19(s): The supervisor process creates one process per job initially to support multi-programming.

   21(s): Multi-programming environment requires that multiple jobs be scheduled.

   35(s): Some memory allocation scheme is required to support a multi-programming environment.

   40(s): Device handler routine facilitates the reading of multiple job streams from different sources.

   60(s): JCL assists multi-programming by delineating jobs and specifying resource requirements.

   70(s): The supervisor process controls and synchronize all the functions in a multi-programming environment.

8. The operating system must run on a machine that has two

distinct states; i.e., problem and supervisor.

12(s): User communication with the operating system via a special call ensures that the user may be restricted from certain privileged instructions.

16(s): Only certain special instructions are user callable.

18(s): Special instructions explicitly executed in the supervisor state.

49(s): An interrupt handler must be available in order to change machine states.

67(s): User processes an restricted to the problem state.

9. All resource requests must pass through the supervisor process.

10:(s): All resources, less processor, must be allocated prior to the job being made eligible to run.

12:(s): Resource requests are processed as privileged instructions through the supervisor process.

28:(s): The resources must also be reclaimed by the supervisor.

29:(s): Resources are reclaimed when an error condition terminates job processing.

31:(s): Memory allocation is a resource request.

37:(s): Device management is a resource which must be allocated.

60:(s): JCL specifies the resources required of a job to the supervisor process.

70:(s): The supervisor process controls all resource allocations.

10. System resources must be allocated to a job prior to the job being made eligible to run.

11(c): User resources; i.e., processes are allocated at the process level.

19(s): The supervisor process allocates all resources to a job.

31:(s): Memory is an allocatable resource.

37:(s): The device handler routine is allocated to a job at this time.

60:(s): JCL enables the user to identify his resource needs.

70:(s): The supervisor process controls all resource allocations.

11. A process must be ready to run prior to being allocated a process.

14(s): The status of a process is directly maintained and controlled by information tables.

25(c): A process shall not be ready if it exceeds the time quantum.

26(c): A process shall not be ready if it is waiting to synchronize with another process.

27(c): A process shall not be ready if it specifically relinquishes control to the traffic controller.

59(c): A process shall not be ready if it is waiting to receive a message.

12. User communication with the operating sytem is via special call.

16(c): Only certain of the special calls are available to user processes.

27(s): A process may relinquish control to the operating system via special call.

46(s): The process synchronization mechanism is implemented using a special call.

49(s): The supervisor call interrupt is generated by special call.

52(c): The message facility is another mechanism employed for user communication.

68(s): The user must signal completion using a special call.

13. The operating system must protect user jobs from each other.

14:(s): Information tables contain the information required to protect user's jobs.

20(s): The creation of a single process initially, isolates user jobs from each other.

35(s): The protection of memory partitions can be

accomplished with the same implementation utilized for the requirement.

64(s): As a protection mechanism, dynamically created processes run in the memory area of parent job.

66(s): To protect jobs, a process can destroy only those non-system processes within its process group.

67(s): User processes run in the problem state to prevent access to system level functions.

14. The operating system must utilize information tables to monitor and control processing.

15(s): Dynamic allocation of system tables is required in support of multi-programming environment.

24(s): Round-robin scheduling is accomplished most effectively by chaining the tables together.

32(s): Memory allocation is accounted for in 2K increments.

33(s): The operating system may dynamically allocate memory for information tables.

36(s): Collapsing free storage areas requires that the system tables be updated.

43(s): P-V mechanism is used extensively to restrict access to system tables for protection.

52(s): The message facility requires use of information tables extensively.

15. System tables can be dynamically allocated and released.

    33(s): Dynamic memory allocation facility fully supports this requirement.

    56(s): The queuing of messages requires a dynamic memory allocation facility.

    66(c): The user is strictly prohibited from dynamic memory allocation.

16. Certain system routines are user callable.

    18(s): Extended machine instructions are executed in the supervisor state to provide a system check to determine if use is authorized.

    51(s): P-V operations are specifically restricted from the user since these are used as system locks.

    52(s): The message facility is made available to all users for user communication.

17. System process routines are re-entrant and shared.

    33(s): The operating system maintains pure code by dynamically allocating memory for work space for system routines.

    37(s): The device management process is a system routine which must be shared among many users.

    44(s): The process synchronization mechanism is used as a lock to synchronize usage of certain routines.

70(s): The supervisor process is a system routine which must be shared among many jobs.

18. Extended machine instructions are executed in the supervisor state.

49(s): An interrupt handler must be provided to recognize and handle extended machine instructions.

67(c): User processes must run in the problem state, and generate calls to the operating system via extended machine instructions for resources.

19. The supervisor process must schedule jobs and prepare the jobs for execution.

20(s): The supervisor initially creates one process per job.

21(s): The supervisor schedules jobs strictly on a first-come, first-served basis.

22(s): The functions of the supervisor, and the interfaces must be clearly defined so that improvements may be easily accomplished.

28(s): Another function of the supervisor routine is to reclaim all system resources.

29(c): The supervisor must reclaim resources when a process generates a system level error.

62(s): The supervisor must also load the user's deck in order to prepare a job for execution.

70(s): One supervisor process exists per job stream.

20. Initially, one process is created for each user's job.

    63(s): The user process may create additional processes to form a process group after the user process has been initiated.

21. Jobs are initiated strictly on a first-come, first-served basis.

    22(s): The FCFS scheduling is simplistic; therefore, we can improve system performance at some later time by modularizing this function.

    40(s): The fact that all input devices are dedicated card readers, forces the FCFS implementation.

    71(c): The provision for a fast I/O processing mechanism may preclude a job from being scheduled strictly FCFS.

22. The supervisor process must be modularized so that improvements to the system can be easily accomplished.

    70(s): Modularization of the supervisor process requires that its functions and interfaces be clearly defined so that any change in its implementation be made explicit.

23. The process scheduler must time-slice CPU usage among ready processes to achieve multi-programming.

    24(s): All processes are scheduled round-robin, so that the next sequential ready process is selected for scheduling.

25(s): The specific time-slice quantum equals 50ms.

50(s): An external interrupt is generated when a timer runout is deleted, and a handler must be provided.

25. Ready processes are scheduled in simple round-robin fashion by the process scheduler.

26(c): A process is not scheduled it is is waiting for synchronization with another process.

44(s): A process synchronization mechanism must be provided to enqueue ready processes in a chain.

59(s): A process is not scheduled if it is waiting for message synchronization with another process.

63(s): User processes may create additional processes which must in turn be scheduled.

71(c): The fast I/O processing mechanism allows immediate scheduling of a process, conflicting with the round-robin scheduling.

25. A process must be blocked, when a time quantum of 50ms is exceeded.

50(s): An external interrupt is generated when the time quantum is exceeded, and an interrupt handler must process the interrupt.

26. A process is blocked, when waiting for synchronization with another process.

44(s): A process synchronization mechanism is provided.

48(s): A program interrupt mechanism is provided to enable a process to signal that it is waiting for synchronization.

51(s): Process synchronization mechanism is available only to system processes.

59(s): The user processes utilize the message facility to signal other user processes for synchronization.

27. A process is blocked, when it specifically relinquish control to the process scheduler.

48(s): A program interrupt facility is required so that a process can signal the process scheduler.

68(s): The user must signal completion of a process, and, thereby, relinquish control of the processor to the process scheduler.

28. The supervisor process must reclaim all system resources from a job when the job has completed.

29(c): The supervisor must also reclaim resources if a user process generates a system level error.

36(s): Free storage areas must be collapsed and reconfigured when a job ends.

37(s): The device handler routine for a particular job must be terminated.

43(s): All system locks must be released when a

particular job terminates.

46(s): All devices which are locked by the job must be released.

48(s): The user must signal the end of his job, and an interrupt handler must be provided to deal with the signal.

68(s): The user is required to signal completion.

70(s): The supervisor process is restarted when the job ends just long enough to clean up all the resources.

29. Supervisor must reclaim system resources when a user process generates a system level error.

48(s): Upon generation of a system level error interrupt, a handler must take control and deal with the interrupt.

68(c): Normally the user must signal completion, but this requirement dictates that abnormal ending must be recognized.

30. Reference to processes within a process group is by symbolic name.

53(s): The message sending and receiving recognition mechanism is strictly accomplished by process names.

54(s): Same as 53.

63(s): Dynamically created processes within a process

group must be named as they are initiated.

64(s): Processes of the same process group must run on the same memory area as the parent job.

66(s): User processes may destroy other user processes only within the same process group by symbolic name.

31. The operating system must allocate memory for a job, the size of which is to be supplied by the user.

32(c): Memory allocation is limited to 2K increments.

34(s): Memory must be allocated using a best-fit algorithm.

36(c): Memory is collapsed into contiguous blocks whenever it is freed, which enables reassignment.

43(s): The process synchronization mechanism may be used to lock a database after allocation.

60(s): The user specifies his memory requirements in JCL.

65(c): Once initial memory has been allocated, the user cannot dynamically allocate memory.

32. Memory is allocated in 2K blocks.

34(s): The best-fit algorithm is used to limit the wasted memory space.

35(s): Allocation is 2K blocks allows hardware protection of memory be IBM/360 hardware.

36(s): Memory is configured whenever it is freed.

43(s): The process synchronization mechanism can be used to lock a database once memory has been allocated.

60(s): The user must supply his memory requirements in 2K increments.

65(c): User process cannot dynamically allocate memory whereas system process can.

33. Operating system can dynamically allocate memory to itself for temporary workspace or buffer areas for system processes.

35(s): Once allocated, memory areas must be protected to prevent simultaneous access.

36(s): Memory must be reconfigured by the operating system whenever a block is freed.

37(s): The device management system requires memory for temporary workspaces.

43(s): The process synchronization mechanism can be used to lock databases.

56(s): The message facility requires dynamic memory allocation to enqueue messages.

65(c): User processes are strictly prohibited from dynamically allocating memory.

34. Memory is allocated using a best-fit algorithm.

36(s): Memory is configured when de-allocated to ensure that the largest contiguous blocks are

available to the system.

60(s): The user must specify his memory requirements in a JCL statement.

35. Memory must be protected to prevent the simultaneous allocation of a partition to multiple jobs.

43(s): The process synchronization mechanism is available to lock a database.

64(s): Dynamically created process must run in the same memory partition as the parent job, which further protects memory.

65(s): The user is strictly prohibited from dynamically allocating memory which reduces the protection requirements.

36. Free storage areas are collapsed into contiguous blocks of memory whenever a job partition is freed.

68(s): The user must signal completion of his job, to the operating system so that memory may be reclaimed.

37. The operating system must supply a device management system, which runs as a separate process, one per device.

38(s): A device handler routine must be included in the device management system.

39(s): Since devices are dedicated, only one person per device is required.

40(s): The device handler routine is specifically

required to support only one card reader per input stream.

41(s): The device handler routine is specifically required to support only one printer per output stream.

42(s): The device management system must enable the user to supply his own routine for non-standard devices.

46(s): The process synchronization mechanism is available to lock a dedicated device.

47(s): An interrupt handler routine is provided to process I/O interrupts.

69(s): The user must declare his devices, and is limited to a card reader, a printer, and a non-standard device.

38. Device handler routines must support multiple job streams from card readers.

39(s): Dedicated devices enable sequential processing and simplify the designation of job stream.

40(s): A card reader represents an input stream; hence, multiple card readers represent multiple job streams.

61(s): One aspect of the device handler routine is to distinguish among JCL, object deck, and user's data.

69(s): The user must specify which card reader consti-

tutes his input job stream.

39. A device is dedicated to a job.

    40(s): Since devices are dedicated, a card reader represents an input job stream.

    41(s): Since devices are dedicated, a printer represents an output job stream.

    42(s): Non-standard devices employed by the user must be dedicated to his job.

    46(s): The process synchronization mechanism is available to lock a device.

    60(s): The user must identify the devices used by a JCL statement.

    69(s): The user must explicitly identify which devices he is using.

40. The device handler routine supports one card reader per input stream.

    42(c): The user must specify his own handler routine for any non-standard devices used.

    46(s): The process synchronization mechanism can be used to lock a device to an input stream.

    60(s): The user must identify the devices used by a JCL statement.

    61(s): The device handler must enable the operating system to discern between JCL, object deck, and user's data.

    69(s): The user is limited to one card reader or non-

standard device for input.

41. The device handler routine must support one line printer per output stream.

    42(c): A user must supply his own handler routine for any non-standard devices.

    46(s): The process synchronization mechanism can be used to lock a device for an output stream.

    60(s): The user must specify a printer for use in the JCL statement.

    69(s): The user is limited to one line printer or non-standard device.

42. The user must provide his own routines for non-standard devices.

    47(s): An interrupt handler for I/O interrupts must recognize that a user is providing his own device handler routine.

    60(s): The user must specify the use of a non-standard device in a JCL statement.

    61(s): Any non-standard device handler routine must recognize JCL, object deck, and user's data.

    69(s): The user is limited to a single non-standard device.

43. A process synchronization mechanism must be provided to serve as a lock on a database.

    44(s): The mechanism also may be used for the timing

of synchronous processes.

45(s): The mechanism may also be used for synchronization of the message facility.

46(s): The mechanism may also be used to lock a device.

51(s): The mechanism is restricted to use by system processes only.

44. A process synchronization mechanism must be provided for the timing of synchronous processes.

45(s): The mechanism is also used for synchronization of the message facility.

46(s): The mechanism is also used to lock a device.

51(s): The mechanism is restricted to use by system processes only.

45. A process synchronization mechanism must be provided for synchronization between the sender and receiver in message processing.

46(s): The mechanism is also used to lock a device.

51(s): The mechanism is restricted to use by system processes only.

56(s): The mechanism is used to establish an ordered queue for the message facility.

46. A process synchronization mechanism must be provided to lock a device.

51(s): The mechanism is restricted to use by system processes only.

47. An interrupt handler routine must be provided for I/O interrupts.

    48(s): An interrupt handler routine must also be provided for program interrupts.

    49(s): An interrupt handler routine must also be provided for supervisor call interrupts.

    50(s): An interrupt handler routine must also be provided for external interrupts.

    61(s): The interrupt handler may be utilized to recognize input data from the user's job stream.

    71(s): The interrupt handler must provide a special facility to enable fast processing of I/O requests for non-standard devices requiring frequent updates.

48. An interrupt handler routine must be provided for program interrupts.

    49(s): An interrupt handler routine must also be provided for supervisor call interrupts.

    50(s): An interrupt handler routine must also be provided for external interrupts.

    68(s): The user must signal process completion via a program interrupt.

49. An interrupt handler routine must be provided for supervisor call interrupt.

    50(s): An interrupt handler routine must also be provided for external interrupts.

52. A message facility must be provided to all processes.

53(s): The message facility must enable the process receiving a message to determine the originator of the message.

54(s): The message facility must enable the process to read the name and text from the originator.

55(s): The facility must be able to handle messages of an arbitrary, yet specified length.

56(s): The faculty must use some sort of chaining to queue waiting messages.

57(s): The facility must be able to release messages for a process which terminates.

58(s): There is no need for a receiver of a message to acknowledge to the originator.

59(s): The message facility can be used for process synchronization by blocking processes, expecting messages.

53. The process receiving a message must be able to determine the originator of the message.

54(s): The message determines the originator by reading the name of the originating process, separate from the text.

58(s): As long as the receiver knows from whence the message came, there is no need for receipt.

59(s): A process may be blocked until it receives the message it anticipates from a specific process.

54. The receiving process may read the name and text from the originator.

    56(s): In a queue of multiple messages, a process must be able to determine the name and text of the originator.

    58(s): As long as a process can read the name of the originator, there is no need to receipt a message.

    59(s): A process may be synchronized by blocking it until it receives the proper text from a given process.

55. Messages are of an arbitrary yet specified length.

    56(s): Messages may be of a variable length and number; therefore, a queuing process is required to store all messages dynamically.

56. Any number of messages to a process may be queued while waiting to be used.

    57(s): The queued messages may not necessarily be read by a process; therefore, they must be released when that process terminates.

57. All messages, enqueued for a given process to read, are released when that process terminates.

    58(s): A process may never read the messages addressed to it; therefore, there is no facility required for receipting.

68(s): A user process must signal completion to the operating system so that the enqueued messages for that process may be released.

60. User programs utilize a job control language statement to specify resource requirements.

61(s): The operating system must be capable of discerning among JCL, user's object deck, and user's data.

69(s): The user must specify I/O devices in the JCL statement.

61. The operating system must accept input data from the user's job stream.

70(s): The supervisor process controls the input of the user's job stream and must, therefore, separate all the JCL, user's object deck, and data.

62. The supervisor process must load the user supplied object deck into the user's area of memory.

70(s): The supervisor process explicitly performs this function as it exists, one per job stream.

63. All processes may dynamically create additional process.

64(s): Such processes are limited to the initial user memory area.

66(s): The user processes can also destroy processes but these are limited to user processes only.

64. Dynamically created processes run in the same memory area as the parent job.

    65(s): The user cannot dynamically allocate memory; therefore, all user processes must run in the area of the parent.

    66(s): User processes of different jobs are made invisible to each other and, therefore, can only destroy processes within the same process group.

    67(s): The user processes run in the problem state and, therefore, are not capable of allocating additional memory.

66. User processes can destroy other user processes only within the same process group.

    67(s): User processes run in the problem state and in the same memory area as the parent job; therefore, user processes of different process groups are invisible to each other.

APPENDIX I

Results of the Interactive Decomposition Package for the Second Iteration

```
 REQ:
SAVE
 ENTER FILE NAME:
SOSA5

 STATUS SAVED IN FILE SOSA5

 REQ:
NOLK

 RECORDED LINKS.
 FROM NODE     TO NODE(S):

      7   ( 10) 10, 13, 14, 17, 19, 21, 35, 40, 60, 70,
      8   (  5) 12, 16, 18, 49, 67,
      9   (  8) 10, 12, 28, 29, 31, 37, 60, 70,
     10   (  8)  7,  9, 11, 19, 31, 37, 60, 70,
     11   (  6) 10, 14, 25, 26, 27, 59,
     12   (  8)  8,  9, 16, 27, 46, 49, 52, 68,
     13   (  7)  7, 14, 20, 35, 64, 66, 67,
     14   ( 10)  7, 11, 13, 15, 24, 32, 33, 36, 43, 52,
     15   (  4) 14, 33, 56, 66,
     16   (  5)  8, 12, 18, 51, 52,
     17   (  5)  7, 33, 37, 44, 70,
     18   (  4)  8, 16, 49, 67,
     19   (  9)  7, 10, 20, 21, 22, 28, 29, 62, 70,
     20   (  3) 13, 19, 63,
     21   (  5)  7, 19, 22, 40, 71,
     22   (  3) 19, 21, 70,
     23   (  3) 24, 25, 50,
     24   (  7) 14, 23, 26, 44, 59, 63, 71,
     25   (  3) 11, 23, 50,
     26   (  6) 11, 24, 44, 48, 51, 59,
     27   (  4) 11, 12, 48, 68,
     28   ( 10)  9, 19, 29, 36, 37, 43, 46, 48, 68, 70,
     29   (  5)  9, 19, 28, 48, 68,
     30   (  5) 53, 54, 63, 64, 66,
     31   (  8)  9, 10, 32, 34, 36, 43, 60, 65,
     32   (  8) 14, 31, 34, 35, 36, 43, 60, 65,
     33   (  9) 14, 15, 17, 35, 36, 37, 43, 56, 65,
     34   (  4) 31, 32, 36, 60,
     35   (  7)  7, 13, 32, 33, 43, 64, 65,
     36   (  7) 14, 28, 31, 32, 33, 34, 68,
     37   ( 13)  9, 10, 17, 28, 33, 38, 39, 40, 41, 42,
                46, 47, 69,
     38   (  5) 37, 39, 40, 61, 69,
     39   (  8) 37, 38, 40, 41, 42, 46, 60, 69,
     40   ( 10)  7, 21, 37, 38, 39, 42, 46, 60, 61, 69,
     41   (  6) 37, 39, 42, 46, 60, 69,
     42   (  8) 37, 39, 40, 41, 47, 60, 61, 69,
     43   ( 10) 14, 28, 31, 32, 33, 35, 44, 45, 46, 51,
     44   (  7) 17, 24, 26, 43, 45, 46, 51,
     45   (  5) 43, 44, 46, 51, 56,
```

```
 46    ( 10) 12, 28, 37, 39, 40, 41, 43, 44, 45, 51,
 47    (  7) 37, 42, 48, 49, 50, 61, 71,
 48    (  8) 26, 27, 28, 29, 47, 49, 50, 68,
 49    (  6)  8, 12, 18, 47, 48, 50,
 50    (  5) 23, 25, 47, 48, 49,
 51    (  6) 16, 26, 43, 44, 45, 46,
 52    ( 10) 12, 14, 16, 53, 54, 55, 56, 57, 58, 59,
 53    (  5) 30, 52, 54, 58, 59,
 54    (  6) 30, 52, 53, 56, 58, 59,
 55    (  2) 52, 56,
 56    (  7) 15, 33, 45, 52, 54, 55, 57,
 57    (  4) 52, 56, 58, 68,
 58    (  4) 52, 53, 54, 57,
 59    (  6) 11, 24, 26, 52, 53, 54,
 60    ( 12)  7,  9, 10, 31, 32, 34, 39, 40, 41, 42,
              61, 69,
 61    (  6) 38, 40, 42, 47, 60, 70,
 62    (  2) 19, 70,
 63    (  5) 20, 24, 30, 64, 66,


 64    (  7) 13, 30, 35, 63, 65, 66, 67,
 65    (  5) 31, 32, 33, 35, 64,
 66    (  6) 13, 15, 30, 63, 64, 67,
 67    (  5)  8, 13, 18, 64, 66,
 68    (  7) 12, 27, 28, 29, 36, 48, 57,
 69    (  7) 37, 38, 39, 40, 41, 42, 60,
 70    (  9)  7,  9, 10, 17, 19, 22, 28, 61, 62,
 71    (  3) 21, 24, 47,

 (AVERAGE NO. OF LINKS PER NODE:      5.225).


 REQ:
DENO
:
/1,2,3,4,5,6,73,74,75,76,77,78,79,80/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
    1     2     3     4     5     6    73    74    75    76
   77    78    79    80
```

```
 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO.  NEW NO.
 --------------
```

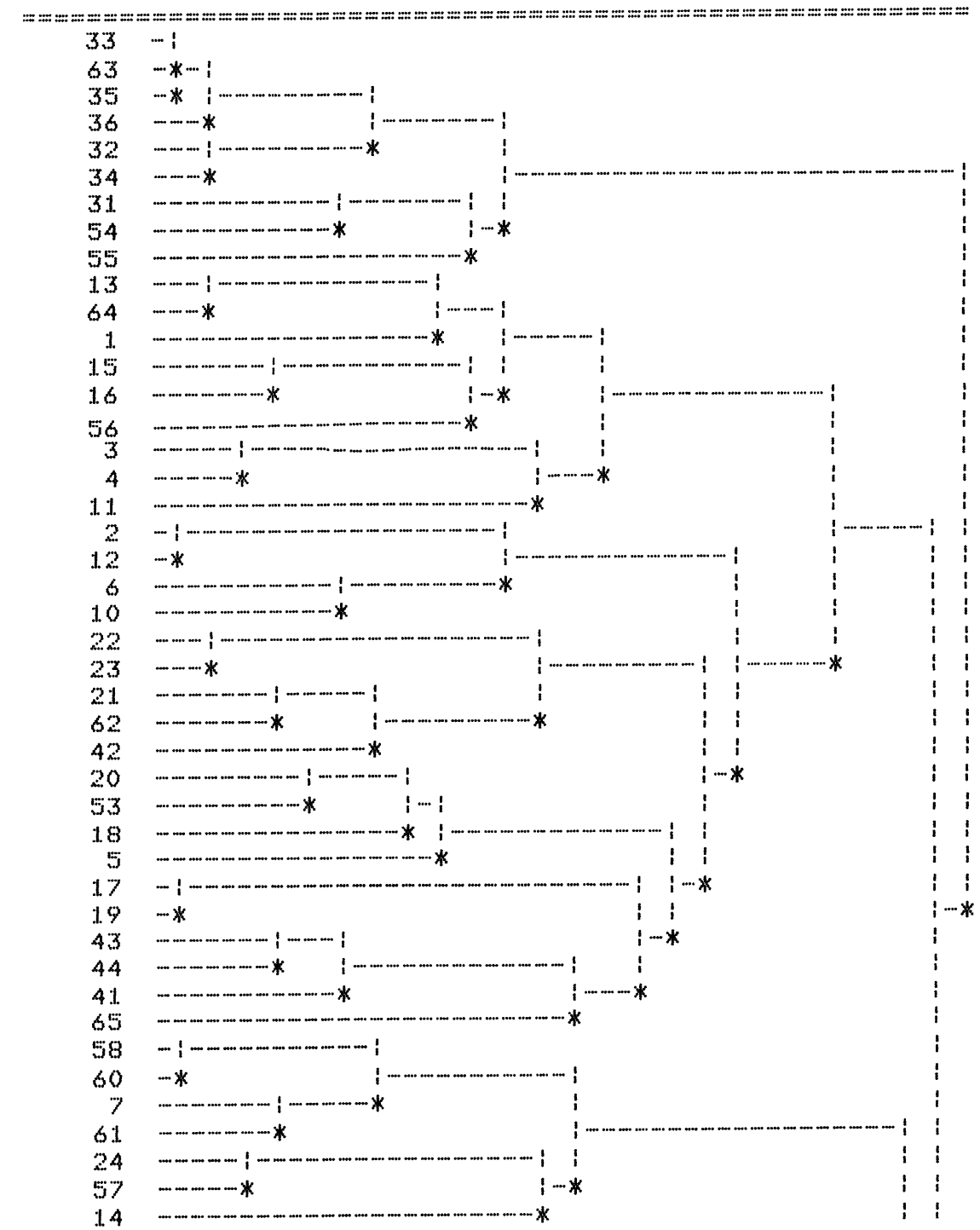
7	1	29	23	51	45
8	2	30	24	52	46
9	3	31	25	53	47
10	4	32	26	54	48
11	5	33	27	55	49
12	6	34	28	56	50
13	7	35	29	57	51
14	8	36	30	58	52
15	9	37	31	59	53
16	10	38	32	60	54
17	11	39	33	61	55
18	12	40	34	62	56
19	13	41	35	63	57
20	14	42	36	64	58
21	15	43	37	65	59
22	16	44	38	66	60
23	17	45	39	67	61
24	18	46	40	68	62
25	19	47	41	69	63
26	20	48	42	70	64
27	21	49	43	71	65
28	22	50	44	72	66

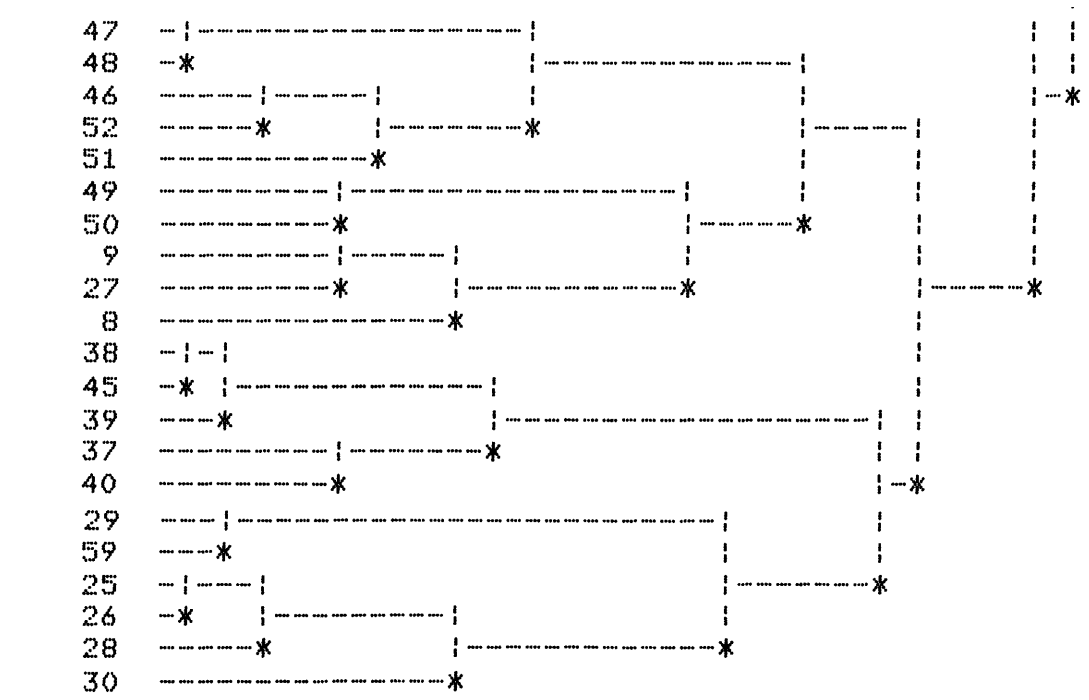
```
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 NO PRECLUSTERING PERFORMED; DISTANCE MATRIX COMPUTED WITH P = 1.

 REQ:
ISOL

 ISOLATED NODES:
  66

 REQ:
DENO
:
/66/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
   66
```

```
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 NO PRECLUSTERING PERFORMED; DISTANCE MATRIX COMPUTED WITH P = 1.

 REQ:
SIMA

 SIMILARITY MATRIX COMPUTED.

 REQ:
INPA
 ENTER PERCENTAGE PARAMETER:
80.


 INITIAL PARTITION COMPUTED WITH P = 80.00 %.

 REQ:
HCM1
 BEST PARTITION MEASURE:    1.263
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------

    1     ( 8)   1   3   4  13  15  16  56  64
    2     ( 5)   2   6  10  12  43
    3     ( 4)   5  17  19  44
    4     ( 7)   7  14  24  57  58  60  61
    5     ( 9)   8   9  25  26  27  28  29  30  59
    6     (12)  11  31  32  33  34  35  36  41  54  55
                63  65
    7     ( 7)  18  20  37  38  39  40  45
    8     ( 5)  21  22  23  42  62
    9     ( 8)  46  47  48  49  50  51  52  53
```

```
 REQ:
HCM2
 BEST PARTITION MEASURE:    1.276
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 ---------------------
    1     (11)   1   3   4  11  13  15  16  22  23  56
                64
    2     (13)   2   6  10  12  17  19  21  41  42  43
                44  62  65
    3     ( 9)   5  18  20  37  38  39  40  45  53
    4     ( 7)   7  14  24  57  58  60  61
    5     ( 9)   8   9  25  26  27  28  29  30  59
    6     ( 9)  31  32  33  34  35  36  54  55  63
    7     ( 7)  46  47  48  49  50  51  52

 REQ:
HCM3
 BEST PARTITION MEASURE:    1.411
 DO YOU WANT TO PRINT THE TREE?
NOP

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 ---------------------
    1     ( 9)   1   3   4  11  13  15  16  56  64
    2     ( 4)   2   6  10  12
    3     (15)   5  17  18  19  20  21  22  23  41  42
                43  44  53  62  65
    4     ( 7)   7  14  24  57  58  60  61
    5     (10)   8   9  27  46  47  48  49  50  51  52
    6     ( 6)  25  26  28  29  30  59
    7     ( 9)  31  32  33  34  35  36  54  55  63
    8     ( 5)  37  38  39  40  45

 REQ:
EVAL

 STRENGTH:  2.7333,
 COUPLING:  1.3228,
 MEASURE:   1.411.
```

```
HCM3
 BEST PARTITION MEASURE:    1.411
 DO YOU WANT TO PRINT THE TREE?
YES
 SET PAPER AND PRESS RETURN:
```





MEASURES:

-201.500	-196.000	-191.500	-185.667	-179.167	-172.667
-164.667	-161.667	-155.750	-150.917	-146.167	-140.542
-133.042	-124.792	-120.375	-116.875	-110.792	-104.792
-99.792	-95.292	-92.292	-88.042	-83.458	-80.042
-74.458	-69.458	-65.792	-60.875	-57.958	-53.292
-49.570	-46.617	-43.117	-39.700	-35.422	-30.728
-25.311	-22.978	-20.094	-19.567	-16.408	-14.619
-12.669	-10.072	-7.544	-6.707	-5.655	-4.016
-3.307	-2.604	-1.605	-0.371	0.182	0.934
1.183	1.359	1.411	1.387	1.053	0.971
0.893	0.618	0.555	0.070		

```
 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 ---------------------
    1     ( 9)   1   3   4  11  13  15  16  56  64
    2     ( 4)   2   6  10  12
    3     (15)   5  17  18  19  20  21  22  23  41  42
                43  44  53  62  65
    4     ( 7)   7  14  24  57  58  60  61
    5     (10)   8   9  27  46  47  48  49  50  51  52
    6     ( 6)  25  26  28  29  30  59
    7     ( 9)  31  32  33  34  35  36  54  55  63
    8     ( 5)  37  38  39  40  45

 REQ:
DENO
:
/2,6,10,12,5,17,18,19,20,21,22,23,41,42,43,44,53,62,65,7,14,24,57,
:
58,60,61,8,9,27,46,47,48,49,50,51,52,25,26,28,29,30,59,31,32,33,
:
34,35,36,54,55,63,37,38,39,40,45/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
    2     5     6     7     8     9    10    12    14    17
   18    19    20    21    22    23    24    25    26    27
   28    29    30    31    32    33    34    35    36    37
   38    39    40    41    42    43    44    45    46    47
   48    49    50    51    52    53    54    55    57    58
   59    60    61    62    63    65

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO.  NEW NO.
 ----------------
    1        1
    3        2
    4        3
   11        4
   13        5
   15        6
   16        7
   56        8
   64        9
```

```
 REQ:
DENO
:
/2,6,10,12,5,17,18,19,20,21,22,23,41,42,31,
:
43,44,53,62,65,7,14,24,57,58,60,61,8,9,27,46,47,48,49,
:
50,51,52,25,26,28,29,30,59,32,33,34,35,36,54,55,63,37,38,39,40,45/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
    2     5     6     7     8     9    10    12    14    17
   18    19    20    21    22    23    24    25    26    27
   28    29    30    31    32    33    34    35    36    37
   38    39    40    41    42    43    44    45    46    47
   48    49    50    51    52    53    54    55    57    58
   59    60    61    62    63    65

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO.  NEW NO.
 ---------------
    1        1
    3        2
    4        3
   11        4
   13        5
   15        6
   16        7
   56        8
   64        9

 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 NO PRECLUSTERING PERFORMED; DISTANCE MATRIX COMPUTED WITH P = 1.

 REQ:
HCM3
 BEST PARTITION MEASURE:    0.250
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1    ( 9)   1   2   3   4   5   6   7   8   9

 REQ:
EVAL

 STRENGTH:  0.2500,
 COUPLING:  0.0000,
 MEASURE:   0.250.
```

```
 REQ:
DENO
:
/1,3,4,11,13,15,16,56,64,5,17,18,19,20,21,22,23,41,42,43,44,
:
53,62,65,7,14,24,57,58,60,61,8,9,27,46,47,48,49,50,51,52,
:
25,26,28,29,30,59,31,32,33,34,35,36,54,55,63,37,38,39,40,45/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
    1     3     4     5     7     8     9    11    13    14
   15    16    17    18    19    20    21    22    23    24
   25    26    27    28    29    30    31    32    33    34
   35    36    37    38    39    40    41    42    43    44
   45    46    47    48    49    50    51    52    53    54
   55    56    57    58    59    60    61    62    63    64
   65

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO.  NEW NO.
 ---------------
    2        1
    6        2
   10        3
   12        4

 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
HCM3
 BEST PARTITION MEASURE:     0.333
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1     ( 4)   1   3   2   4

 REQ:
EVAL

 STRENGTH:   0.3333,
 COUPLING:   0.0000,
 MEASURE:    0.333.
```

```
 REQ:
DENO
:
/1,3,4,11,13,15,16,56,64,2,6,10,12,7,14,24,57,58,60,
:
61,8,9,27,46,47,48,49,50,51,52,25,26,28,29,30,59,31,32,33,34,
:
35,36,54,55,63,37,38,39,40,45/
```

THE FOLLOWING NODES HAVE BEEN REMOVED:

1	2	3	4	6	7	8	9	10	11
12	13	14	15	16	24	25	26	27	28
29	30	31	32	33	34	35	36	37	38
39	40	45	46	47	48	49	50	51	52
54	55	56	57	58	59	60	61	63	64

NODES HAVE BEEN RENAMED AS FOLLOWS:

OLD NO.	NEW NO.
5	1
17	2
18	3
19	4
20	5
21	6
22	7
23	8
41	9
42	10
43	11
44	12
53	13
62	14
65	15

```
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.
```

```
 REQ:
HCM3
 BEST PARTITION MEASUFE:     0.480
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 ....................
    1     ( 5)   1   3   5  13  15
    2     ( 5)   2   4   9  11  12
    3     ( 5)   6   7   8  10  14

 REQ:
EVAL

 STRENGTH:   0.8000,
 COUPLING:   0.3200,
 MEASURE:    0.480.
```

```
 REQ:
DENO
:
/1,3,4,11,13,15,16,56,64,2,6,10,12,5,17,18,19,20,21,22,23,41,42,
:
43,44,53,62,65,8,9,27,46,47,48,49,50,51,52,
:
25,26,28,29,30,59,31,32,33,34,35,36,54,55,63,37,38,39,40,45/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
    1     2     3     4     5     6     8     9    10    11
   12    13    15    16    17    18    19    20    21    22
   23    25    26    27    28    29    30    31    32    33
   34    35    36    37    38    39    40    41    42    43
   44    45    46    47    48    49    50    51    52    53
   54    55    56    59    62    63    64    65

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO.  NEW NO.
 ---------------
    7        1
   14        2
   24        3
   57        4
   58        5
   60        6
   61        7

 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
HCM3
 BEST PARTITION MEASURE:     0.333
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1     ( 7)    1    2    3    4    5    6    7

 REQ:
EVAL

 STRENGTH:   0.3333,
 COUPLING:   0.0000,
 MEASURE:    0.333.
```

```
 REQ:
DENO
:
/1,3,4,11,13,15,16,56,64,2,6,10,12,5,17,18,19,20,21,22,23,41,42,
:
43,44,53,62,65,7,14,24,57,58,60,61,25,26,28,29,30,59,31,32,33,34,
:
35,36,54,55,63,37,38,39,40,45/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
    1     2     3     4     5     6     7    10    11    12
   13    14    15    16    17    18    19    20    21    22
   23    24    25    26    28    29    30    31    32    33
   34    35    36    37    38    39    40    41    42    43
   44    45    53    54    55    56    57    58    59    60
   61    62    63    64    65

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO. NEW NO.
 ---------------
    8        1
    9        2
   27        3
   46        4
   47        5
   48        6
   49        7
   50        8
   51        9
   52       10
 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
HCM3
 BEST PARTITION MEASURE:    0.524
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1     ( 3)   1   2   3
    2     ( 7)   4   5   6   7   8   9  10

 REQ:
EVAL

 STRENGTH:  0.6667,
 COUPLING:  0.1429,
 MEASURE:   0.524.
```

```
 REQ:
DENO
:
/1,3,4,11,13,15,16,56,64,2,6,10,12,5,17,18,19,20,21,22,23,41,42,
:
43,44,53,62,65,7,14,24,57,58,60,61,8,9,27,46,47,48,49,50,51,52,
:
31,32,33,34,35,36,54,55,63,37,38,39,40,45/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
    1     2     3     4     5     6     7     8     9    10
   11    12    13    14    15    16    17    18    19    20
   21    22    23    24    27    31    32    33    34    35
   36    37    38    39    40    41    42    43    44    45
   46    47    48    49    50    51    52    53    54    55
   56    57    58    60    61    62    63    64    65

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO.  NEW NO.
 ---------------
    25        1
    26        2
    28        3
    29        4
    30        5
    59        6

 REQ:
DIMN
 (PRECLUSTERING COMPLETE)
 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
HCM3
 BEST PARTITION MEASURE:     0.333
 DO YOU WANT TO PRINT THE TREE?
NO

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 --------------------
    1     ( 6)   1   2   3   5   4   6

 REQ:
EVAL

 STRENGTH:   0.3333,
 COUPLING:   0.0000,
 MEASURE:    0.333.
```

```
 REQ:
DEND
*
/1,3,4,11,13,15,16,56,64,2,6,10,12,5,17,18,19,20,21,22,23,41,42,
*
43,44,53,62,65,7,14,24,57,58,60,61,8,9,27,46,47,48,49,50,51,52,25,
26,28,29,30,59,37
   THE FOLLOWING NODES HAVE BEEN REMOVED:
     1     2     3     4     5     6     7     8     9    10
    11    12    13    14    15    16    17    18    19    20
    21    22    23    24    25    26    27    28    29    30
    37    38    39    40    41    42    43    44    45    46
    47    48    49    50    51    52    53    56    57    58
    59    60    61    62    64    65

   NODES HAVE BEEN RENAMED AS FOLLOWS:
   OLD NO.  NEW NO.
   -------  -------
     31        1
     32        2
     33        3
     34        4
     35        5
     36        6
     54        7
     55        8
     63        9

    REQ:
   DIMN
    (PRECLUSTERING COMPLETE)
   H
    PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
    CLUSTERS NOT TAKEN AS SINGLE NODES.

    REQ:
   CM3
    BEST PARTITION MEASURE:     0.528
    DO YOU WANT TO PRINT THE TREE?
   NO

    REQ:
   PRCL

    CLUSTER (NO) OBJECTS
    --------------------
       1     ( 9)   1   2   3   9   4   5   6   7   8

    REQ:
   EVAL

    STRENGTH:   0.5278,
    COUPLING:   0.0000,
    MEASURE:    0.528.
```

```
 REQ:
DENO
 :
/1,2,3,4,11,13,15,16,56,64,6,10,12,5,17,18,19,20,21,22,23,41,42,
:
43,44,53,62,65,7,14,24,57,58,60,61,8,9,27,46,47,48,49,50,51,52,25,
:
26,28,29,30,59,31,32,33,34,35,36,54,55,63/

 THE FOLLOWING NODES HAVE BEEN REMOVED:
    1     2     3     4     5     6     7     8     9    10
   11    12    13    14    15    16    17    18    19    20
   21    22    23    24    25    26    27    28    29    30
   31    32    33    34    35    36    41    42    43    44
   46    47    48    49    50    51    52    53    54    55
   56    57    58    59    60    61    62    63    64    65

 NODES HAVE BEEN RENAMED AS FOLLOWS:
 OLD NO.  NEW NO.
 ---------------
   37        1
   38        2
   39        3
   40        4
   45        5

 REQ:
DIMN
 (PRECLUSTERING COMPLETE)

 PRECLUSTERING PERFORMED AND DISTANCE MATRIX COMPUTED WITH P = 1,
 CLUSTERS NOT TAKEN AS SINGLE NODES.

 REQ:
HCM3
 CURRENT PRCLUSTERING HAS ONLY ONE CLUSTER.
 UNABLE TO DO IT.

 REQ:
PRTL

 CLUSTER (NO) OBJECTS
 ------- ------------
    1     ( 5)   1   2   3   4   5

 REQ:
EVAL

 STRENGTH:  0.6000,
 COUPLING:  0.0000,
 MEASURE:   0.600.
```

```
 REQ:
REST
 ENTER FILE NAME:
SOSA12

 ADJACENCY MATRIX READ FROM FILE SOSA12

 REQ:
NWPA
:
/1,3,4,11,13,15,16,56,64/
:
2,6,10,12/
:
5,18,20,53,65/
:
17,19,41,43,44/
:
21,22,23,42,62/
:
7,14,24,57,58,60,61/
:
8,9,27/
:
46,47,48,49,50,51,52/
:
25,26,28,29,30,59/
:
31,32,33,34,35,36,54,55,63/
:
37,38,39,40,45/
:
$

 REQ:
PRCL

 CLUSTER (NO) OBJECTS
 ---------------------
    1     ( 9)   1   3   4  11  13  15  16  56  64
    2     ( 4)   2   6  10  12
    3     ( 5)   5  18  20  53  65
    4     ( 5)  17  19  41  43  44
    5     ( 5)  21  22  23  42  62
    6     ( 7)   7  14  24  57  58  60  61
    7     ( 3)   8   9  27
    8     ( 7)  46  47  48  49  50  51  52
    9     ( 6)  25  26  28  29  30  59
   10     ( 9)  31  32  33  34  35  36  54  55  63
   11     ( 5)  37  38  39  40  45
```

```
 REQ:
PRLK

 LINKS BETWEEN CLUSTERS    1 &    2 :
                 3 -    6

 LINKS BETWEEN CLUSTERS    1 &    3 :
                 4 -    5
                15 -   65

 LINKS BETWEEN CLUSTERS    1 &    4 :
NONE.

 LINKS BETWEEN CLUSTERS    1 &    5 :
                 3 -   22
                 3 -   23
                13 -   22
                13 -   23
                64 -   22

 LINKS BETWEEN CLUSTERS    1 &    6 :
                 1 -    7
                13 -   14

 LINKS BETWEEN CLUSTERS    1 &    7 :
                 1 -    8
                11 -   27

 LINKS BETWEEN CLUSTERS    1 &    8 :
NONE.

 LINKS BETWEEN CLUSTERS    1 &    9 :
                 1 -   29
                 3 -   25
                 4 -   25

 LINKS BETWEEN CLUSTERS    1 &   10 :
                 1 -   34
                 1 -   54
                 3 -   31
                 3 -   54
                 4 -   31
                 4 -   54
                11 -   31
                15 -   34
                64 -   55

 LINKS BETWEEN CLUSTERS    1 &   11 :
                11 -   38

 LINKS BETWEEN CLUSTERS    2 &    3 :
NONE.
```

```
LINKS BETWEEN CLUSTERS    2 &    4 :
                  2 -   43
                  6 -   43
                 12 -   43

LINKS BETWEEN CLUSTERS    2 &    5 :
                  6 -   21
                  6 -   62

LINKS BETWEEN CLUSTERS    2 &    6 :
                  2 -   61
                 12 -   61

LINKS BETWEEN CLUSTERS    2 &    7 :
NONE.

LINKS BETWEEN CLUSTERS    2 &    8 :
                  6 -   46
                 10 -   46

LINKS BETWEEN CLUSTERS    2 &    9 :
NONE.

LINKS BETWEEN CLUSTERS    2 &   10 :
NONE.

LINKS BETWEEN CLUSTERS    2 &   11 :
                  6 -   40
                 10 -   45

LINKS BETWEEN CLUSTERS    3 &    4 :
                  5 -   19
                 18 -   17
                 65 -   41

LINKS BETWEEN CLUSTERS    3 &    5 :
                  5 -   21
                 20 -   42

LINKS BETWEEN CLUSTERS    3 &    6 :
                 18 -   57

LINKS BETWEEN CLUSTERS    3 &    7 :
                  5 -    8
                 18 -    8

LINKS BETWEEN CLUSTERS    3 &    8 :
                 53 -   46
                 53 -   47
                 53 -   48

LINKS BETWEEN CLUSTERS    3 &    9 :
NONE.
```

LINKS BETWEEN CLUSTERS 3 & 10 :
NONE.

LINKS BETWEEN CLUSTERS 3 & 11 :
18 - 38
20 - 38
20 - 45

LINKS BETWEEN CLUSTERS 4 & 5 :
41 - 42
43 - 42
44 - 42

LINKS BETWEEN CLUSTERS 4 & 6 :
NONE.

LINKS BETWEEN CLUSTERS 4 & 7 :
NONE.

LINKS BETWEEN CLUSTERS 4 & 8 :
NONE.

LINKS BETWEEN CLUSTERS 4 & 9 :
NONE.

LINKS BETWEEN CLUSTERS 4 & 10 :
41 - 31
41 - 36
41 - 55

LINKS BETWEEN CLUSTERS 4 & 11 :
NONE.

LINKS BETWEEN CLUSTERS 5 & 6 :
NONE.

LINKS BETWEEN CLUSTERS 5 & 7 :
NONE.

LINKS BETWEEN CLUSTERS 5 & 8 :
62 - 51

LINKS BETWEEN CLUSTERS 5 & 9 :
22 - 30
62 - 30

LINKS BETWEEN CLUSTERS 5 & 10 :
22 - 31

LINKS BETWEEN CLUSTERS 5 & 11 :
22 - 37
22 - 40

LINKS BETWEEN CLUSTERS 6 & 7 :
7 - 8
60 - 9

LINKS BETWEEN CLUSTERS 6 & 8 :
24 - 47
24 - 48

LINKS BETWEEN CLUSTERS 6 & 9 :
7 - 29
58 - 29
58 - 59

LINKS BETWEEN CLUSTERS 6 & 10 :
NONE.

LINKS BETWEEN CLUSTERS 6 & 11 :
NONE.

LINKS BETWEEN CLUSTERS 7 & 8 :
8 - 46
9 - 50
27 - 50

LINKS BETWEEN CLUSTERS 7 & 9 :
8 - 26
8 - 30
27 - 29
27 - 30
27 - 59

LINKS BETWEEN CLUSTERS 7 & 10 :
27 - 31

LINKS BETWEEN CLUSTERS 7 & 11 :
8 - 37
27 - 37

LINKS BETWEEN CLUSTERS 8 & 9 :
NONE.

```
 LINKS BETWEEN CLUSTERS    8 &   10 :
NONE.

 LINKS BETWEEN CLUSTERS    8 &   11 :
                50 -   39

LINKS BETWEEN CLUSTERS     9 &   10 :
                25 -   54
                26 -   54
                28 -   54

LINKS BETWEEN CLUSTERS     9 &   11 :
                25 -   37
                26 -   37
                29 -   37

LINKS BETWEEN CLUSTERS    10 &   11 :
                31 -   40
                33 -   40
                34 -   40
                35 -   40
```

APPENDIX J

Main Subproblems Resulting From The Second Iteration of the Decomposition Analysis

Note: (11) The number in the parenthesis indicates the number of interdependencies identified for the requirement.

Main Subproblem 1: Supervisor Process:

7 (10): The operating system must provide for a multi-programming environment.

9 (8): All resource requests must pass through the supervisor.

10 (8): System resources must be allocated to a job prior to it being runnable.

17 (5): System process routines are re-entrant and shared.

19 (9): Supervisor process must schedule jobs and prepare them for execution.

21 (5): Jobs are initiated strictly on a first-come, first-served basis.

22 (3): Supervisor process must be modularized so that improvements are easy.

62 (2): Supervisor process must load the user-supplied object deck into memory.

70 (9): There is one supervisor process per job stream.

Main Subproblem 2: Extended Machine Instruction Mechanism:

8 (5): Operating system must run on a machine that has two states.

12 (8): User communication with operating system is via special call.

16 (5): Certain system routines are user callable.

18 (4): Extended machine instructions are executed in the supervisor state.

Main Subproblem 3: Process Control Functions:

SUBPROBLEM MS 3-A - Process Scheduling:

11 (6): A process must be ready to run prior to being allocated a processor.

24 (7): Ready processes are scheduled in round-robin fashion by process scheduler.

26 (6): A process shall be blocked when awaiting synchronization.

59 (6): If no messages are available to a process explicitly then it goes blocked.

71 (3): I/O interrupt handler must provide for a synchronous scheduling of a process requiring fast processing.

SUBPROBLEM MS 3-B - System Initiated Interrupts:

23 (3): Process scheduler must time-slice CPU usage.

25 (9): A process shall be blocked when its time quantum is exceeded.

47 (7): Interrupt handler must be provided for I/O interrupts.

49 (6): Interrupt handler must be provided for supervisor call interrupts.

50 (5): Interrupt handler must be provided for external interrupts.

SUBPROBLEM MS 3-C - User Process Initiated Interrupts:

27 (4): A process shall be blocked when it specifically relinquishes control.

28 (10): Supervisor routine must reclaim all system resources when a job is completed.

29 (5): Supervisor must reclaim resources when an error condition is raised.

48 (8): Interrupt handler must be provided for program interrupts.

68 (7): User process must signal completion to the operating system.

Main Subproblem 4: Process Creation Functions:

13 (7): Operating system must protect user jobs from each other.

20 (3): Initially one process is created for each user's job.

30 (5): Reference to a process is by symbolic name.

63 (5): All processes may dynamically create additional processes.

64 (7): Dynamically created processes run on the same memory area as parent job.

66 (6): User processes can destroy other user processes only within the same group.

67 (5): User processes run in the problem state.

Main Subproblem 5: Interprocess Communication:

MS 5-A - Operating System Information Tables:

14 (10): Operating system must utilize information tables to monitor and control.

15 (4): System tables can be dynamically allocated and released.

33 (9): Operating system may dynamically allocate memory

to itself for workspace.

MS 5-B - Message Facility

52 (10): Message facility must be provided to all processes.

53 (5): Processes receiving messages must be able to determine the originator.

54 (6): Receiving process may read the name and text from originator.

55 (2): Messages are of an arbitrary yet specified length.

56 (7): Any number of messages may be queued.

57 (4): All messages are released when a process terminates.

58 (4): Messages are not receipted for.

Main Subproblem 6: Memory Allocation Functions:

31 (8): The operating system must allocate memory for a job.

32 (8): Memory is allocated to a job in contiguous 2K blocks.

34 (4): Memory is allocated using a best-fit algorithm.

35 (7): Memory must be protected to prevent simultaneous allocation.

36 (7): Free storage areas are collapsed into blocks when a job is freed.

65 (5): User processes cannot dynamically allocate memory.

Main Subproblem 7: Device Management Functions:

37 (13): Operating system must supply a device management system.

38 (5): Device handler routines must support multiple job streams.

39 (8): A device is dedicated to a job.

40 (10): The device handler routine supports one card reader per input stream.

41 (6): The device handler routine must support one line printer.

42 (8): The user can provide his own routines for non-atandard devices.

60 (12): User programs use JCL to specify resource requirements.

61 (6): Operating system must accept input data from user's job stream.

69 (7): User's job can reference at most 1 input, 1 output, and 1 non-standard device.

Main Subproblem 8: Process Synchronization Functions:

43 (10): A process synchronization mechanism must be provided as a lock database.

44 (7): A process synchronization mechanism must be provided for synchronous process.

45 (5): A process synchronization mechanism must be provided for sender and receiver of messages.

46 (10): A process synchronization mechanism must be provided to lock a device.

51 (6): P-V operations are available only to system processes.

APPENDIX K

Linkage - Interface Assessment

LINKAGE - INTERFACE ASSESSMENT

Subproblem	Cluster Number	Module
Process Scheduling	3	Process Management (lower) Module
System Initiated Interrupt Handler	4	
User Initiated Interrupt Handler	5	
Process Sychronization Mechanism	11	
Memory Allocation	9	Memory Management Module
Operating System Information Tables	7	
Process Creation	6	Process Management (upper) Module
Message Facility	8	
Device Management Functions	10	Device Management Module
Supervisor Process	1	Supervisor Process Module
Extended Machine Instruction Mechanism	2	Supervisor Call Handler

NUMBER OF LINKAGES BETWEEN SUBPROBLEMS

	3	4	5	11	9	7	6	8	10	1	2
3	0	3	2	3	-	2	6	3	-	2	-
4	3	0	3	-	-	-	-	-	3	-	3
5	2	3	0	2	2	-	-	1	1	5	2
11	3	-	2	0	3	2	-	1	4	1	2
9	-	-	2	3	0	5	3	-	-	-	-
7	2	-	-	2	5	0	2	3	-	2	-
6	6	-	-	-	3	2	0	2	-	2	2
8	3	-	1	1	-	3	2	0	-	-	2
10	-	3	1	4	-	-	-	-	0	9	-
1	2	-	5	1	-	2	2	-	9	0	1
2	-	3	2	2	-	-	2	2	-	1	0