CHINESE OUTLINE FONTS SUPPORT IN X WINDOW SYSTEM

BY

RAYMOND CHEUK-KUEN CHEN

A THESIS

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF MASTER OF PHILOSOPHY

DIVISION OF COMPUTER SCIENCE

THE CHINESE UNIVERSITY OF HONG KONG

JUNE 1994

Acknowledgements

Two years are negligible when compared with the history of Chinese Typography, however, it is long enough for a young man to reshape his own view to the world, make friends with some other little young men and, conduct a small scientific research.

Two years has passed by and my research is reaching the end. At this special moment of my life, I would like to express my gratefulness to all the people who have granted me their courteous supports and encouragement. Among them, I would like to give special thanks to my supervisor Dr. Moon Yiu Sang. His advice and guidance has been proved useful and insightful in the course of my research.

Thanks must also be given to my colleague Mr. Wu Kwong Ho for letting me to use his experimental Chinese font rasterizer in my implementation of the Chinese Font Server.

I would also like to thank my friends, Mr. Lam Siu Man and Mr. Wan Kin Man for providing me with so many Chinese articles for testing. Thanks also be given to Miss Patti Tsui for giving me so much advice and suggestion on the writing of this thesis.

Finally, I would like to express my gratitude to all staff members in the Computer Science Department for their excellent technical supports and advice.

C. K. Chen Raymond

Trademarks

Abstract

The X Window System, since its emergence, has gained wide-spread usage in the UNIX world. Although not restricted to the area of UNIX system, X Window System has been the de facto windowing system in the UNIX society. The non-proprietary nature of X Window System allows its source codes to be distributed freely making X Window System an ideal platform for research and development.

The in-born ability of X Window System in handling double-byte characters makes it a good environment to develop Chinese language applications. However, the lack of Chinese font data makes X less competitive for building What You See Is What You Get (WYSIWYG) software.

Moreover, as X is mainly designed for screen display, not much has been done to produce output to printers. Applications have to rely on their own drivers to produce printer output and maintain uniformity between printer output and screen display.

In this thesis, several aspects of outline fonts support in X will be addressed. We will try to introduce the notion of Chinese outline fonts into X by modifying the font server of X Window System. Several different font loading methods will be presented. Experiments are conducted to reveal the deficiency of X Window System in handling double-byte fonts and so, a better loading scheme is suggested.

To overcome the lack of printing facilities in X, an experimental printer server is implemented. This printer server provides device-independent program interface for applications. Double-byte characters are handled by the printer server as easily as single byte characters. Caching will be employed in the printer server to reduce font acquisition time.

As output speed is always a problem in printing Chinese, a new printer driver is designed to speed up the printing process. Comparison with current printer driver will be made with results shown.

Contents

List of Figures

List of Tables

1. Introduction

Chinese language distinguishes itself from other languages, like English and Japanese, in that it is basically a language of ideographics origin. In terms of the number of characters, Chinese has a large character set with over 50,000 characters in which 3,000 are commonly used ones. The large character set of Chinese makes computer processing of the language difficult[16].

Given the number of Chinese characters, it is impossible to encode every character by a single byte representation, which has a total of 256 different combinations only. For this reason, Chinese characters need at least two bytes to encode the full character set. Many different coding schemes have been developed. Two most commonly used schemes are Big5 and GB. Big5 is commonly used in Taiwan and Hong Kong while GB is normally used in mainland China. Since font data with Big5 coding could be more easily accessed, Big5 is used as internal coding scheme in the implementation of this project. For more information of Big5, see Chapter 3, Section 3.3.1.

1.1. Windowing System

Windowing system has won a general acceptance of users for its user friendliness, ease to use and ease to learn. It provides a more comfortable environment for users. For example, to start a new application, a user may simply click on an icon representing that application, rather than typing a command in the terminal. Moreover, applications in a windowing system usually have a common

look and feel. A button on the upper left-hand corner of a window always means the same thing no matter which application it belongs to. This consistency in appearance makes the application software easier to use. Learning new software would also be easier once the user is familiar himself with the window appearance.

In a multi-tasking system, several applications may run simultaneously. Windowing systems allow these applications to share the screen display and input devices for output and input. Each application has its own windows, which are used exclusively by itself. Application can only display information on the windows for which it is the owner. With this sharing of input/output devices, several applications may run simultaneously and interactively. For instance, you may write a report on a word processor window while editing a spreadsheet on another.

Most windowing systems allow sharing of data among different windows[35]. Data, say a text string, can be copied from one window and pasted onto another. Some windowing systems even allow linking of data from one window to another so that any changes of the original one will automatically update the copied one[14].

From a programmer's point of view, windowing system also makes software development easier. Windowing system usually provides higher level toolkits or libraries for application software development[1][25][35][43][44]. These toolkits and libraries include routines for producing screen output, getting keyboard input, getting mouse input and querying system information etc. Instead of writing their own input and output routines, programmers can (and usually must) use these toolkits

and libraries to generate output. Programmers need not interact directly with the hardware on which the applications run; all hardware issues are taken over by the windowing system. By isolating the machine-dependent aspects from the application software, application development becomes easier. Moreover as the applications codes are machine-independent, they can run on any hardware platform supporting the windowing system.

1.2. Fonts

Font is a precious resource in all windowing systems. It forms an important part of the screen interface presented to the users. It is always desirable for a system to produce smooth and elegant character images. In typographical terminology, a font is a typeface in a particular size, weight and slant. All fonts of a typeface share the same design idea of characters, usually defined by the stroke width and serifs. For example, Helvetica is a typeface name while Courier is another. Individual symbols in a font are called glyphs[33].

The size of font is usually measured in point. One point equals to about 1/72 inch. Originally, point size measured the vertical height of the slug that supports characters in printing shops. It should be noted that point size only loosely specifies the size of the characters, that is, characters with the same point size might vary in actual size, as shown in Figure 1-1.

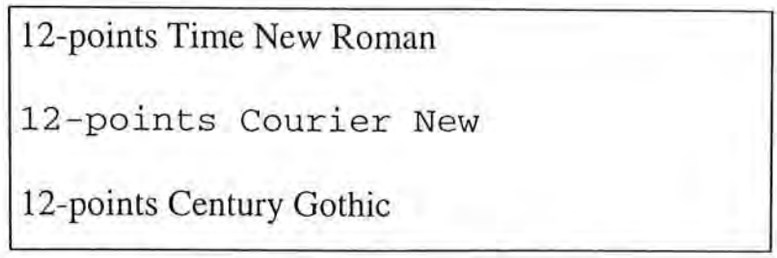


Figure 1-1 12 points characters of three fonts

According to the difference in representations, fonts can be classified into bitmap fonts and outline fonts.

1.2.1. Bitmap Fonts

Bitmap fonts represent each character by a matrix of numbers, with value of either 0 or 1. Value 1 means that a dot will be drawn while value 0 means no dot is drawn. In the actual storage, a bit is needed for each dot. So when the size of the character grows, the memory required to hold the bitmap increases. Moreover, since bitmap fonts depict a character bit-by-bit, the size of the character is fixed. If bitmaps of different sizes are needed, different sets of bitmaps should be provided. Table 1-1 shows the storage requirement of bitmap of different sizes and different number of characters.

Number of Characters	16x16	24x24	32x32	64x64	128x128
1000	31. 25K	70K	125K	500K	2000K
3000	93. 75K	210K	375K	1. 5M	5. 8M
8000	250K	560K	1000K	3. 9M	15. 6M
10000	312. 5K	700K	1. 2M	4. 8M	19M

Table 1-1 Memory requirement(in byte) for different font sizes(in pixel).

Another problem arises when the same bitmap images are used for devices with different resolutions. As bitmap fonts depict a character in a dot-by-dot

manner, the actual size of the character will be different if the size of dot changes. The actual size of a 24x24 bitmap displayed on a 300 dot-per-inch (dpi) printer will be much smaller than that displayed on a 75 dpi screen. This discrepancy in size makes the building of WYSIWYG software difficult. If WYSIWYG is desired, two sets of bitmap data must be provided, one for printer and one for screen.

Despite the apparent disadvantages of using bitmap fonts, they are indeed the only format that can be handled by most printers and screen displays. Only a few printers can handle format other than bitmap. Moreover, as bitmap can be mapped directly to the display surface of printers and screen, the speed of display is extremely fast. In the circumstances where speed is a considerable factor e.g. interactive screen display, bitmap font is usually a better choice.

1.2.2. Outline Fonts

Instead of storing every point on the bitmap, outline fonts only record the boundaries of the characters. For each character, several points of the boundary path are selected, called on-curve **control points** of the character, and their coordinates are stored. Some other points that are not on the boundary, called the **off-curve control points** are also recorded. The boundary of the character can be obtained by connecting these points with straight lines, Bezier curves or cubic spline curves[29].

As the description of the character is independent of the size of the character being used, only one copy of font data is needed. Font bitmap of any size can be

obtained by properly scaling the control points during rasterization. Outline font is also called **scalable font** for its ease to be scaled up and down.

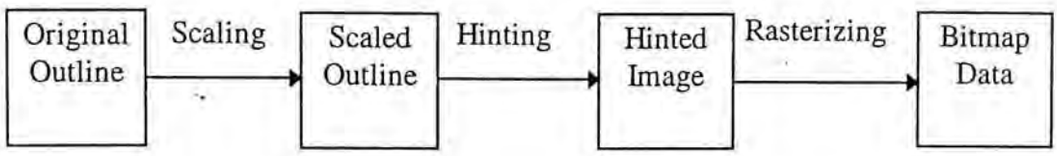


Figure 1-2 Process of font rasterization.

Most output devices are bitmap-oriented. They cannot handle outline fonts directly. Outline fonts should be converted to bitmap before they can be displayed on the output device. The process of converting outline font to bitmap is called **rasterization**. The software responsible for this task is called the **rasterizer**. Usually, rasterization involves several steps, as depicted in Figure 1-2[13][30].

In Figure 1-2, the step hinting is used to compensate for the distortion of low resolution device. With low resolution devices, the scaled outline may not fit properly with the pixel so that some strokes may be thicker while some will be thinner than they should be. Hinting is used to adjust the scaled outline before actual rasterization is performed.

Outline fonts reduce the amount of secondary storage required to store the font data. However, rasterization is slow as compared with the speed of mapping bitmap data to display. Generally, the more complex is the shape of character, the more rasterization time is needed. If a large number of characters is to be rasterized, the response time of the system can be very long. This problem would be more serious in double-byte fonts, for which the sizes of character set are usually very

large. Experiments have been conducted to measure the rasterization time for a whole set of Chinese characters, see Table 3-3 in Section 3.3.4 for details.

The uses of outline fonts in a windowing system leads to other problem. As the rasterizer is usually a part of the system, any misbehavior may be dangerous to the whole system. For instance, in Figure 1-3 a session of Microsoft Chinese

Figure 1-3 A session of MS Window before system corrupt

Windows 3.1 in normal operation is shown. Everything seems to be fine. Then a Chinese character whose font data is known to be bad is accessed. On rasterizing this character, the system returns a *General Protection Fault*[1] and corrupts. The result is shown in Figure 1-4.

[1] General Protection Fault is an error message defined by MS Windows.

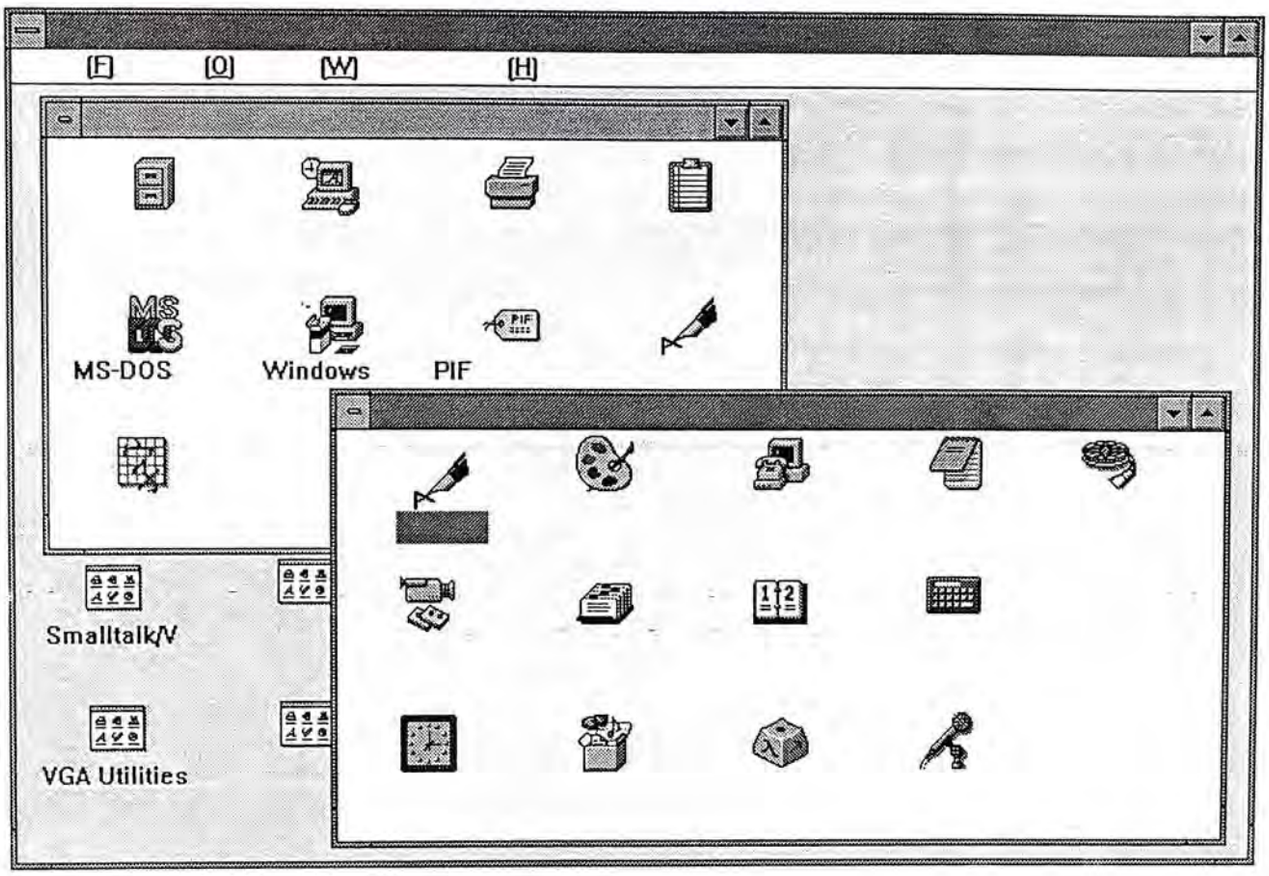


Figure 1-4 A session of MS Windows after system corrupt

1.3. Different font support models

The issue of font support was addressed well before the emergence of any windowing system. Font support refers to the activities of locating, listing, generating and managing of font data. Different modes of font support have been used during the years. In following sections, these different models will be discussed.

1.3.1. Supported by applications

In old days, application programs were usually self-contained. They relied only on the operating system to provide services such reading a disk file. They did not need support from other applications. The application program did all the work

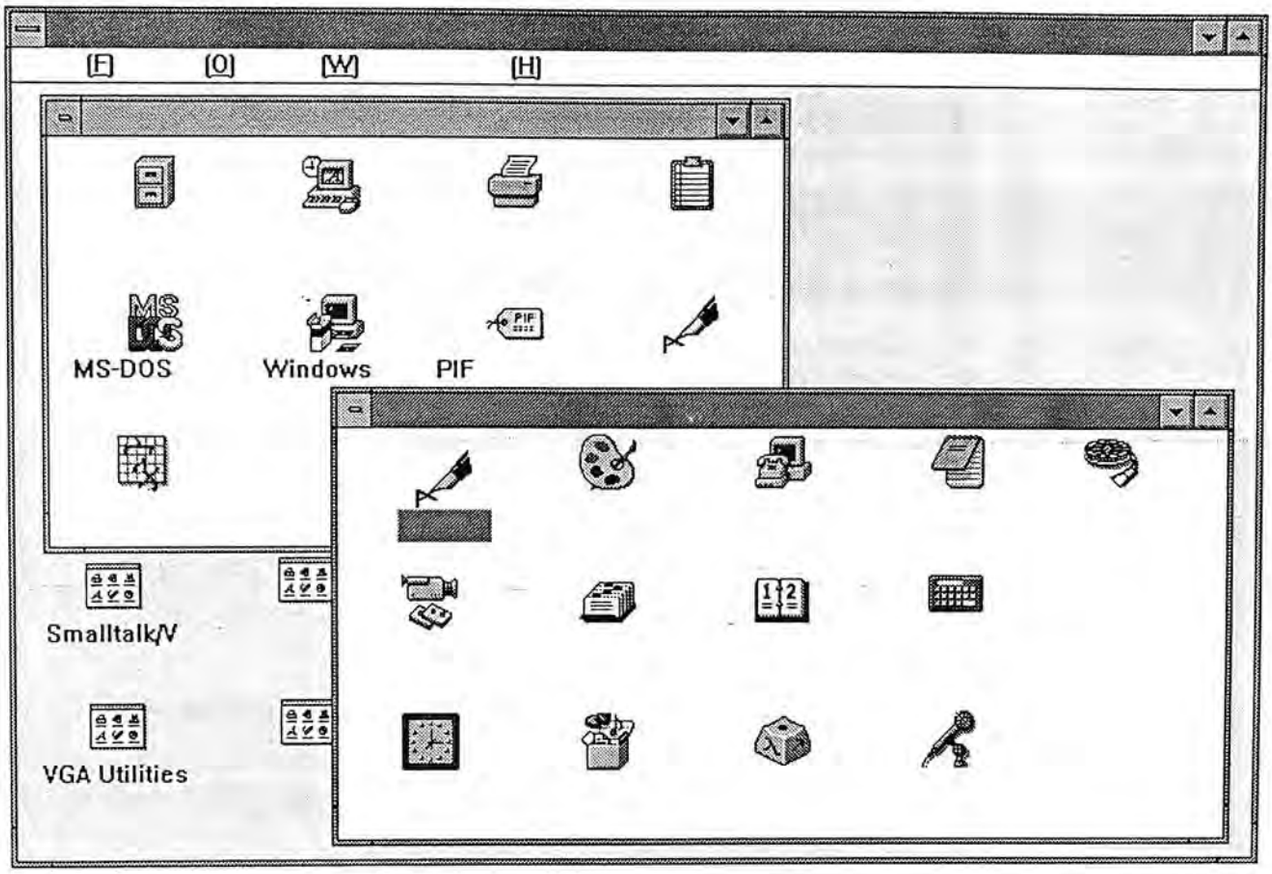


Figure 1-4 A session of MS Windows after system corrupt

1.3. Different font support models

The issue of font support was addressed well before the emergence of any windowing system. Font support refers to the activities of locating, listing, generating and managing of font data. Different modes of font support have been used during the years. In following sections, these different models will be discussed.

1.3.1. Supported by applications

In old days, application programs were usually self-contained. They relied only on the operating system to provide services such reading a disk file. They did not need support from other applications. The application program did all the work

itself. At this time, applications were very machine-dependent. To maintain portability, applications usually came with a set of machine-dependent drivers for every possible variety of monitor, printer and input device. For example, application *A* might be shipped with two display drivers, one for monochrome and one for color display. It might also have two printer drivers, one for 24-pins dot-matrix printer and one for laser jet. Application *B* was the same. So users ended up with a set of incompatible drivers, all for the same hardware configuration.

Besides hardware configuration, the application should also maintain all resources, such as font data, it used. Screen display would be easier if the application is text-based. So, many applications at that time used text-base user interface to simplify and speed up screen display. However, the appearance of text-based applications is usually poorer than that of graphics-based. Some applications prefer to use graphics display and provide the font data they need themselves. Since these font data are maintained and used by individual application, sharing font data among applications is rare and difficult. The situation is shown in Figure 1-5.

If printer output is needed, e.g. in a word processor, the application should use the font data provided by the printer. Since each printer has its own set of built-in fonts, portability of document across different hardware is always a problem, i.e., a document printed on a laser printer would not be the same when it is printed on a dot-matrix printer. For those printers that support soft-font downloading, font data could also be sent to the printers. However, not all applications provide automatic soft font downloading, users might need to do it manually. Furthermore, as the application uses different sources of screen font and printer font, What You See Is

What You Get (WYSIWYG) is not an objective of most word processors of that time[3][12].

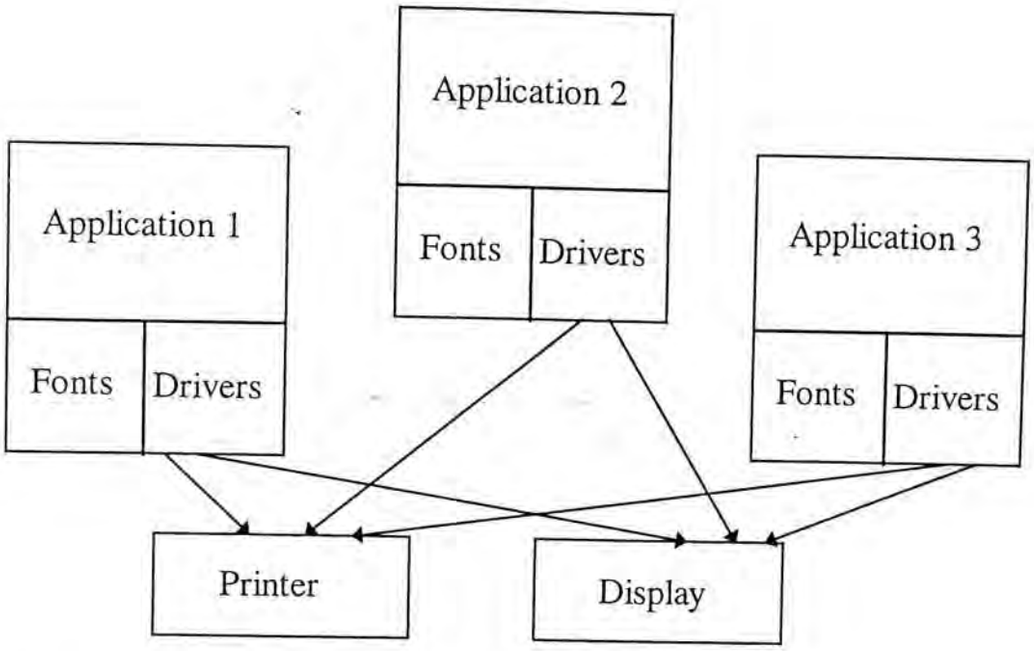


Figure 1-5 Applications support their own fonts and drivers

This model of font support is used in some PC word processors such as WordPerfect and WordStar.

1.3.2. Supported by windowing system

The widespread use of windowing systems, pushes the issue of font support from the application level to system level. Since windowing systems take over the control of screen display from their applications, the job of providing and maintaining font data shifts naturally to the windowing system. Applications can now make use of the display services provided by the system rather than do it themselves. The new arrangement is shown in Figure 1-6.

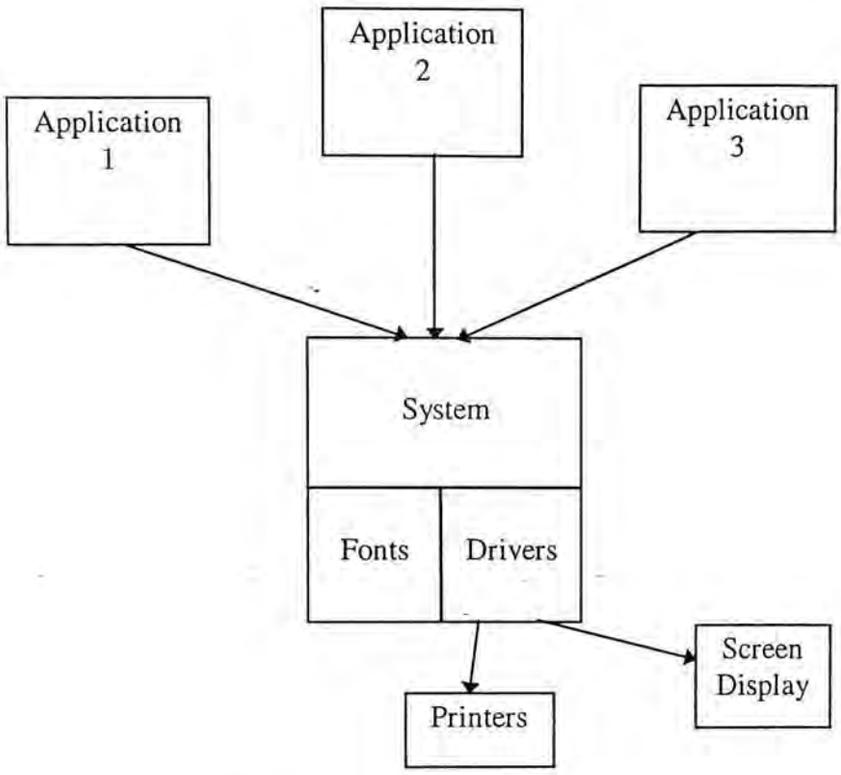


Figure 1-6 Fonts supported by system

This configuration has several advantages. First, applications are freed from managing their font data and which is now taken care of by the system. Second, several applications can share the same set of fonts, without keeping their own set. Third, the system may choose the close match fonts for both screen display and printer output, rendering WYSIWYG possible.

However, this system also has some drawbacks. First, if the system is corrupted by an error during outline font rasterization, all running applications will be affected, as has been discussed in Section 1.2.2. Second, as all fonts are managed by the system, any changes to font format cannot be made without changing the system. For example, in some older versions of X Window System, only one font format was used. To use font data of other format, conversion must be made.

Moreover, new font formats could not be added to X Window System without modification of the system.

1.3.3. Supported by a dedicated server

The last theme in font support moves the font providing unit further away from the applications. The font providing unit is no longer included in the windowing system. It now runs as a separate process. The client-server model is adopted. Font data can be obtained from the font providing unit by sending it a request, as shown in Figure 1-7.

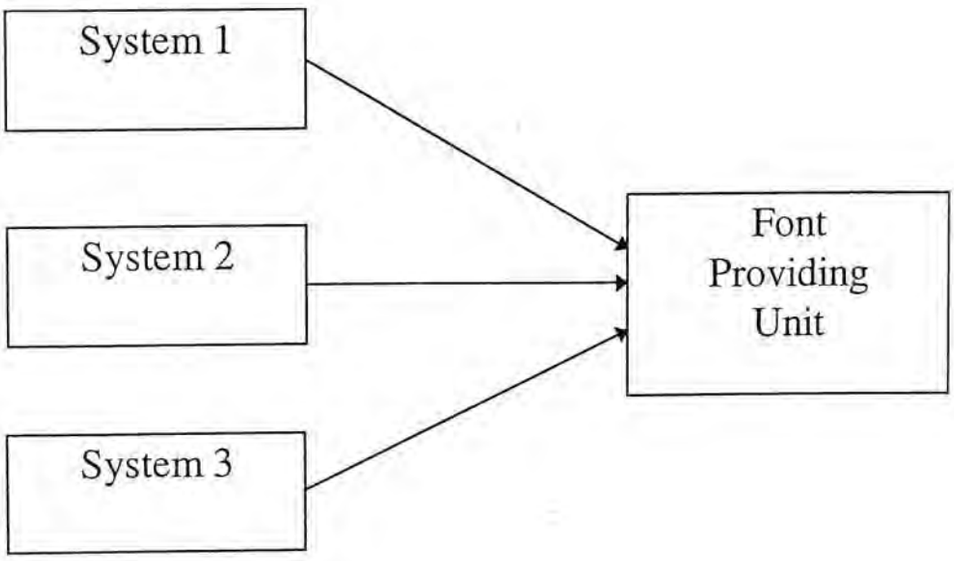


Figure 1-7 Font providing unit as a separate entity

As the font providing unit is separated, different systems may all obtain font data from it as long as these systems obey the protocol defined. Font data is no longer a resource of one single system, it may be shared by several unrelated systems. Separating the font providing unit from a single system also provides the flexibility to change font formats and adds new font formats. Since the systems which need fonts do not interact directly with font data, these changes are transparent to it. Only

the font providing unit is to be changed while keeping all other parts unchanged. The X Window System font server adapts this approach, as will be discussed in Chapter 3.

This arrangement provides higher flexibility; however, the issue of fault tolerance is even more important. As shown in Figure 1-7, several systems rely on the font providing unit to provide font data. If the font providing unit fails (may be due to a single font data error), any subsequent font requests will never be satisfied and the whole system (including System 1, 2 and 3) will be affected. One solution is to employ more than one font providing unit, as shown in Figure 1-8. If any one unit fails, there will still be another font provider in the system[6].

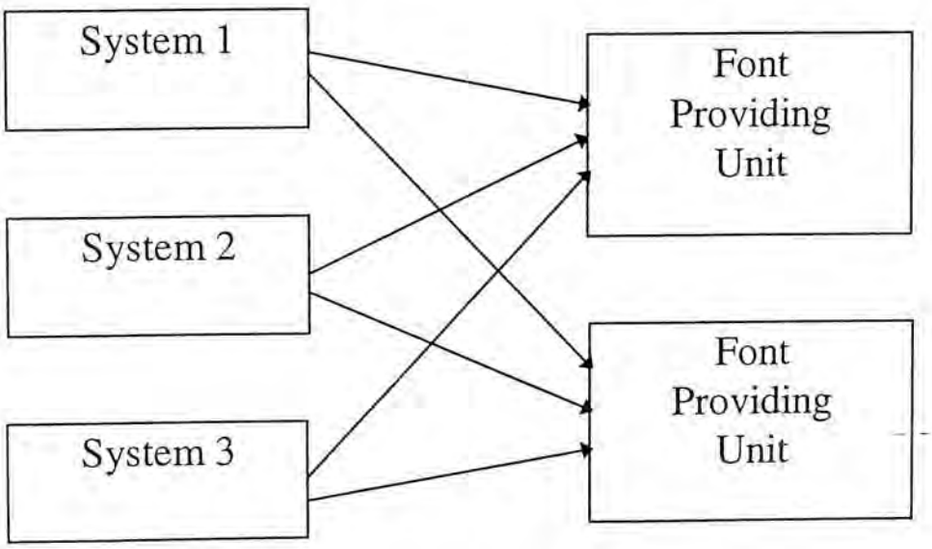


Figure 1-8 System with two font providing units

1.4. Issues of Chinese Font Support

As have been discussed in Section 1.2, the bitmap storage requirement for a large character set language is very high. For example, if two bitmap fonts are used, one being 64x64 and the other being 128x128, then according to Table 1-1, over 24

mega-bytes are required. The high storage requirement makes it unjustified to keep too many Chinese bitmap fonts. In some early Chinese systems[45], only a few Chinese bitmap fonts, say 15x16 and 24x24 were provided in order to save hard disk space.

Using outline Chinese fonts brings another problem. Since Chinese characters are usually more complex in shape than English characters, the time required to rasterize a Chinese character is usually longer. Moreover, if the whole set of Chinese characters is rasterized, the time required will be extremely long. However, Chinese language has a high locality of character usage. Most characters in a Chinese document indeed appear again and again. Font support systems thus should be designed to take advantage of this locality of characters usage in order to reduce the time delay.

Besides storing and generating Chinese fonts, printing of Chinese text is another problematic area of Chinese processing. The printing speed of a Chinese document is usually very slow as compared with printing English text. The reason for the difference is that Chinese characters are usually printed as graphics, not as a soft font. This printing method requires a character's bitmap to be sent to the printer every time when that character is printed. This slows down the printing speed significantly. Better printing methods should be employed to address this problem.

2. Overview of X Window System

2.1. Introduction

The X Window System is a machine-independent, network-transparent windowing system. It was originally developed by the Massachusetts Institute of Technology (MIT) and is now maintained and organized by the X Consortium Inc. X provides a rich set of functions to support both graphical and text output. One of the most important features of X is its device-independence. X allows application programs to display its output on any display hardware without any modification. With its network transparency, an application program running on one machine may also display its output on another one[35][36].

The first version of X that was widely used was version 11. It was released in 1988. Since then, X has been ported to a wide range of platforms with different hardware and operating systems. Despite the fact that X is already a sophisticated and functional complete software, X is constantly under review. The most updated version of X is version 11 release 6 (X11R6) and which was released in May, 1994.

This project is mainly focused on release 5 of version 11, the latest release of X Window System at the time when this project began. Unless otherwise stated, X always refers to this release.

2.2. Architecture

The X Window System is based on a client-server model. It consists of three parts: the X server, X client and the X Window System Protocol. The X server is a program running on a machine with the display hardware. It provides services to generate graphical and text output on its display. Any user of these services is called an X client of the X server. X clients communicate with the server using the X Window System Protocol. The structure of the X Window System is shown in

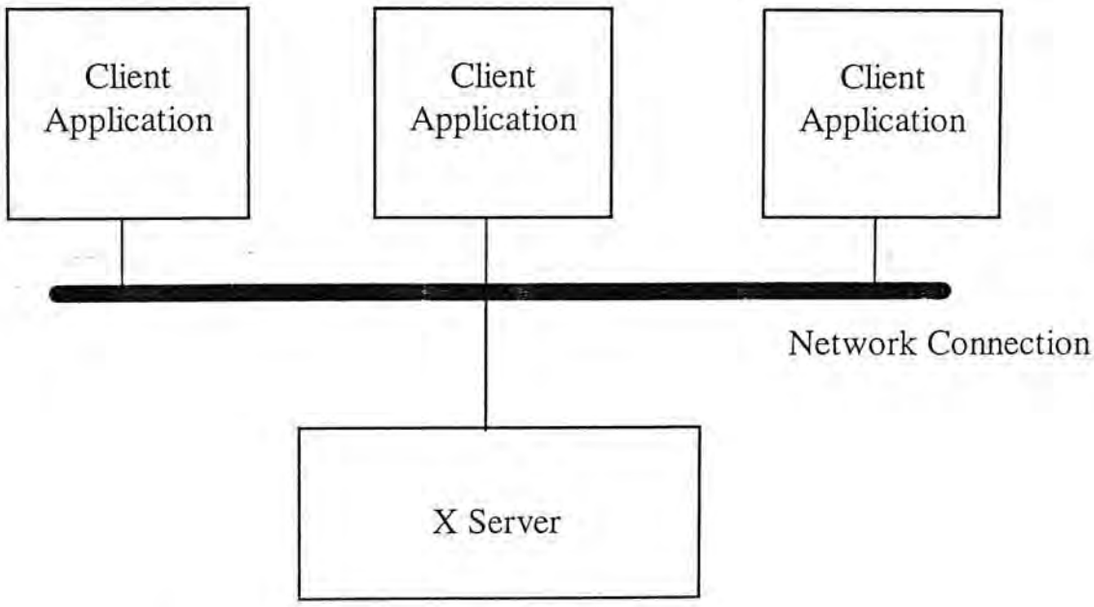


Figure 2-1 Structure of X Window System

Figure 2-1.

2.3. Font Management in the X Window System

Among all other features of X, font management is the one of most interest to us. In X, font is organized by the X server which is responsible for locating, maintaining and retrieving font data for all its applications[17]. The font management

of X has undergone several changes in the past few releases of version 11. The following section will discuss these changes.

2.3.1. Before X Version 11 Release 5

Before Release 5 of Version 11, an X server could only use font data located in the file system accessible to it. The X server maintained the available fonts in several directories called **font directory**. In these releases of X, fonts were usually grouped according to the resolution for which they were designed. In addition to font files, each directory should contained a file called *fonts.dir*. The *fonts.dir* file was used to mapped the font names that were used by client application to the file names. The set of all these font directories constituted the **font path** of the X server. Font path is usually established at the time the X server starts up. Table 2-1 shows the default font path of X.

Directory	Fonts Contained
/usr/lib/X11/fonts/misc	fixed pitch fonts, cursor font
/usr/lib/X11/fonts/75dipi	fonts designed for 75 dot per inch display
/usr/lib/X11/fonts/100dpi	font designed for 100 dot per inch display

Table 2-1 Default Font Path of X

In old releases, all font files were stored in Server Natural Format(SNF)[35] as that was the only format recognizable by X servers. SNF contained the server data structure for the font together with the bitmap information tailor-made for a particular server's architecture. SNF was a bitmap font format. Each SNF file contained data of a font at a particular point size. If several fonts of different point

sizes were needed, several SNF files had to be supplied. Since the SNF format stuck closely to a specified server's implementation, SNF was not portable.

To facilitate fonts interchange among different server implementations, a format called Bitmap Distribution Format(BDF)[35] was defined. All BDF fonts had to be translated to SNF before they could be used by X servers.

Bitmap scaling was not supported by this release. Moreover, outline fonts were also not supported. In this release of X, sharing of font data among several servers of different vendors was difficult as each X server required font data of different formats. The only way to share font data was through BDF files.

2.3.2. In X Version 11 Release 5

X Version 11 Release 5(X11R5) preserves all the font mechanisms of the previous release. In addition, several significant changes are also introduced into the font system, which includes:

- A new standard of font file format, called Portable Compiled Format
- A new font manipulation module, called Font Management Library
- A centralized font serving module called Font Server

2.3.3. Portable Compiled Format

Portable Compiled Format(PCF)[17] was originally designed by the Digital Equipment Corporation. Besides bitmap data describing each character glyph of the font, additional information, like bit-order, byte-order, scan line unit and scan line

padding, of the font file is also given[17]. Applications(including X server) can use these values to convert the bitmap data into the desired format. PCF can be used by all server implementations.

2.3.4. Font Server

Besides the change in font file format, a new font access architecture is introduced in release 5. This new architecture includes a Font Management Library (FML) and a font server. The overall font architecture in Release 5 is shown in Figure 2-2.

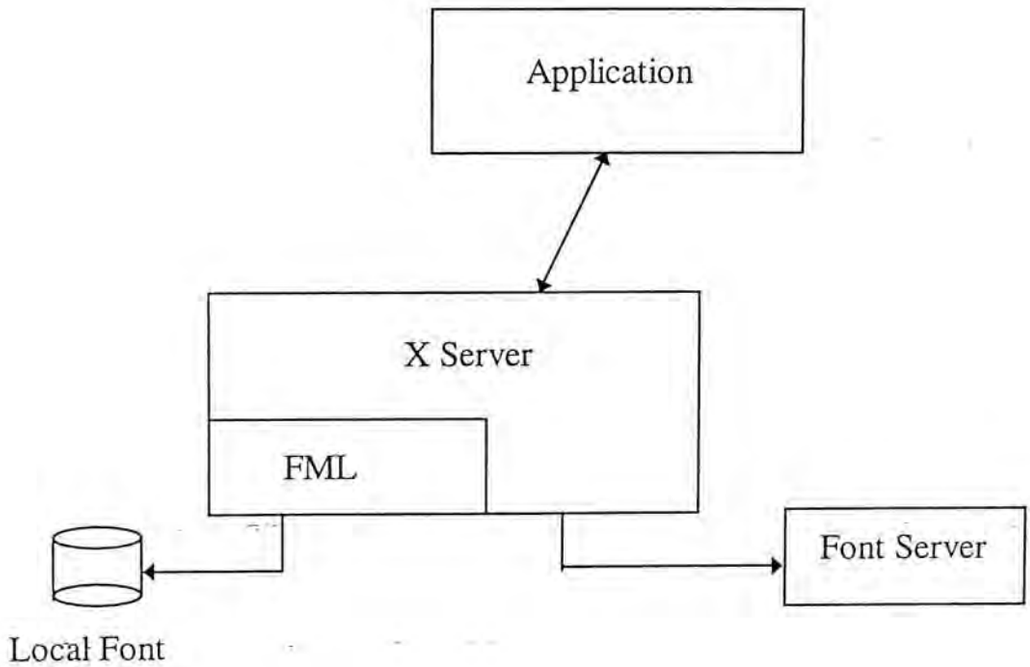


Figure 2-2 Font Architecture in X11R5

A font server is a separate process which is responsible for providing font services to an X server. It may run on a machine other than the one where X server resides. Every time an X server needs information of some fonts in the font server, it may send a request to the font server through the established channel. The font

server then processes the request and generates a reply for that X server. The formats of all requests and replies are defined by a new network protocol, called the X Font Service Protocol[11]. Since the X server and the font server communicate with each other using a new font protocol, not by the X protocol, other applications may also take advantages of the services provided by the font server[21].

In X11R5, the concept of font path is extended to accommodate the naming of the font server. The X server may contain an entry pointing to a remote font server in its font path. For a font server that is using TCP/IP protocols, its name is of the form:

tcp/*hostname:port-number*[31]

where *hostname* is the node name of the machine and *port-number* is the TCP port number that the font server listens to[37]. Unlike naming of font directories as specified in Section 2.3.1, font server names never start with "/". This is to prevent any confusion with entries representing font directories[41].

The existence of a font server is transparent to applications. As shown in Figure 2-2, applications do not request font data directly from the font server. It is the task of the X server to determine whether or not to request font data from the font server.

Throughout the thesis, the term font client will always refer to any processes which request font services from the font server. One example of a font client is the X server.

The font server's architecture allows font data to be managed by a central authority. Also, as a font server can be accessed by more than one font client, font data in the font server can be shared among these font clients. Besides, as the job of reading font files resides in the font server, if a new font format is to be used, only a new font file reader or font rasterizer needs to be included in the font server.

2.3.5. Font Management Library

With the introduction of a font server, the X server now requires functions to access font data from local font directories as well as functions to access the remote font server. To integrate the old functions for local font access and the new module for connecting remote font server, a Font Management Library (FML) is designed. This library contains all the font manipulating functions of the X server, which includes routines for both reading font data from local file system and getting font data from a remote font server.

This font management library is also used by a font server. By using the network connection component of the Font Management Library, a font server can also connect other font server, which in turn connects a third font server and forms a chain of font servers, as shown in Figure 2-3[17]. Note that in Figure 2-3, the existence of font servers 2 and 3 are also transparent to the X server.

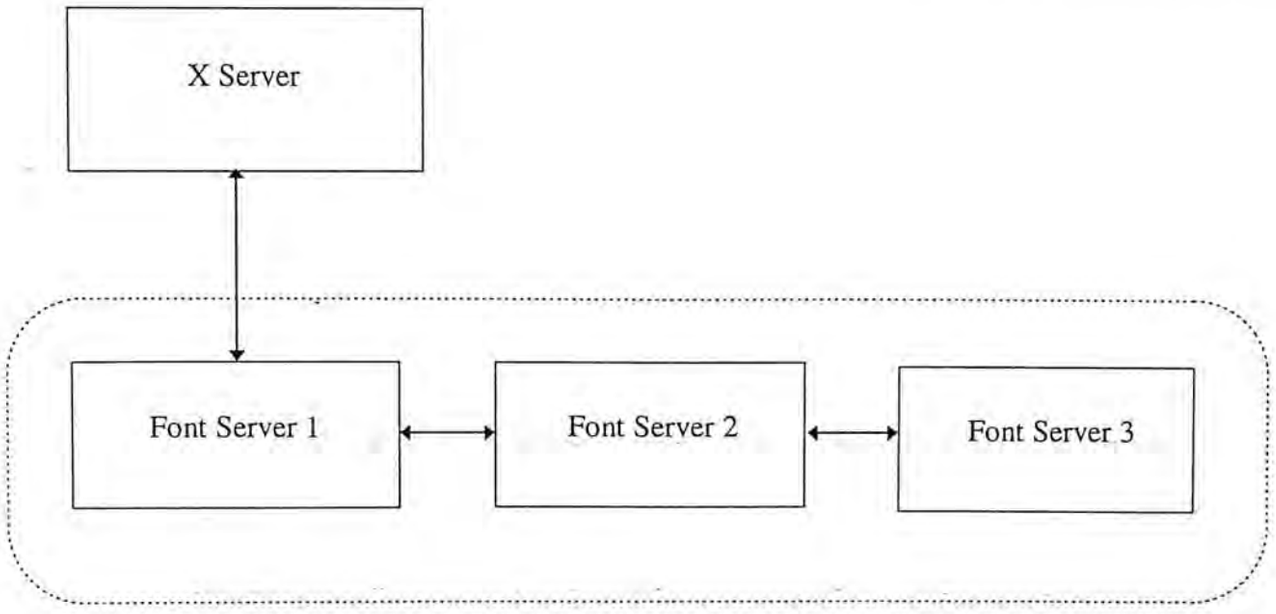


Figure 2-3 X Server connects to a chain of Font Servers

The architecture and operations of the font server and the font management will be discussed in more detail in Chapter 3.

2.4. Internal Code

Another distinct feature of X is its loose restriction on internal coding scheme. In fact, X does not define any internal codes. It allows font data to be associated freely with any internal codes. This distinct feature of X allows font data with any internal coding to be used in X. Moreover, X can handle both single byte and double bytes fonts.

The free policy of X on internal coding makes it an ideal platform for research and experiment[18][49][50]. Work has also been done on the localization of X Window System to several different languages[31].

3. Chinese Font Server

3.1. Motivation

X Window System has been designed with the capability to handle languages with large character sets. Language characters with double-byte representation can easily be managed by X. Graphics routines are also provided by the system to draw double-byte fonts characters. However, the availability of Chinese fonts is poor. Currently only several bitmap Chinese fonts are available in X[2]. These Chinese fonts are designed mainly for screen display purpose. As they are all bitmap fonts, their sizes cannot be scaled up and down without distorting the shape of character. Moreover, being designed to suit screen display resolution which is much lower than resolution of most laser printers, these Chinese bitmap fonts are inadequate for printing. There is indeed a need for more font sources for both screen display and printer output.

From the discussion on the font management system of X Window System given in Chapter 2, it is undoubted that the introduction of the Font Server makes font handling more flexible. With this new font handling mechanism, the inclusion of Chinese outline fonts becomes simpler. Moreover, by modifying a font server, all X servers which need Chinese font data can request from that font server. X server does not need to be changed.

[2] X version 11 Release 6 includes three Chinese bitmap fonts.

Bases on the sample font server included in the distribution of X Version 11 Release 5, a Chinese font server is implemented. Since outline fonts are more flexible than bitmap fonts, the Chinese font server is mainly designed for supporting Chinese outline fonts. Indeed, as will be seen in later chapters, using outline fonts widens the ways the font server is to be used, not to be confined to the X server only.

3.2. Font Server Architecture

The sample font server provided by the X Consortium consists of three modules, including:

- The Device Independent Font Server layer (DIFS)
- The Operating System layer (OS)
- The Font Management Library (FML)

The architecture of Font Server is shown in Figure 3-1.

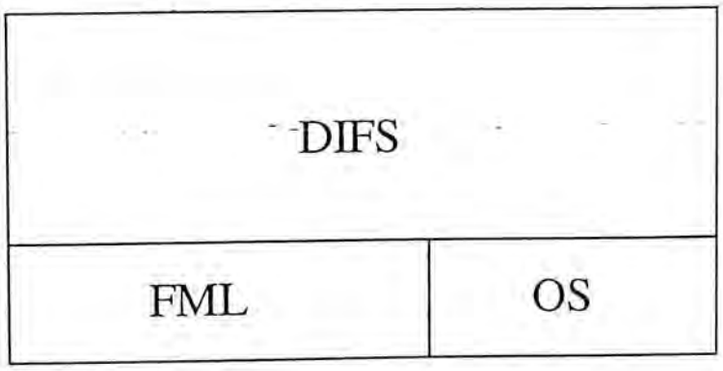


Figure 3-1 Structure of the Font Server

3.2.1. Device Independent Font Server layer(DIFS)

DIFS contains the main event loop of the font server. This event loop is responsible for dispatching requests to the request handling routines. For each protocol request, there is a specific function to handle it. The dispatcher manages the set of all request handling functions into an array which is indexed by the request type. When a request arrives, the request type is used to address the appropriate request handling routine and that routine will be invoked.

DIFS is also responsible for managing font clients. DIFS organizes all font clients into a global array of client record. For each font client of the font server, DIFS assigns an entry of the array for it. The client record maintains information about an individual font client, which includes the client's byte order, its index in the server's client array, sequence number of the last processed request of the client and pointers of routines to process client requests. The maximum number of font clients of the font server is implementation dependent.

3.2.2. Operating System layer(OS)

OS contains routines for network communications, server configuration and error handling. The OS layer is also responsible for font catalogue manipulation. The codes contained in OS may be operating-system dependent[17].

3.2.3. Font Management Library(FML)

The Font Management Library is a special module of the font server. All font manipulation functions are contained in the FML. Font accessing functions can be divided into two main categories, namely the functions to access a remote font server and functions to manipulate font files.

The Font Management Library collects the set of routines for manipulations of a particular font origin into a **Font Renderer**. The term font origin refers to the location, either local or remote, from which font data can be obtained. Font Renderer contains routines to open font, close font and list font, etc.

In the sample server, there are two Font Renderers, called **Font File Renderer** and **Font Server Renderer**[17]. Font File Renderer is responsible for managing font data that are stored in the local file system while the Font Server Renderer is used to request fonts from remote the font server. The structure of FML is shown in Figure 3-2.

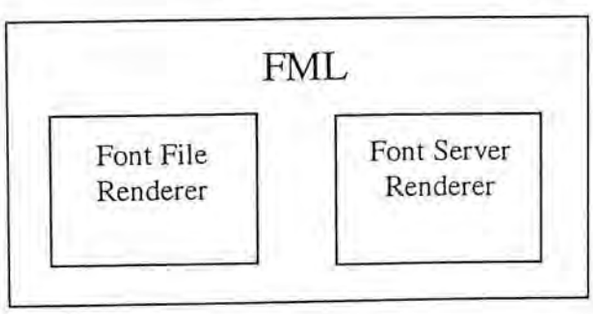


Figure 3-2 Hierarchical Structure of Font Management Library

Font Renderers provide routines for font access. However, the FML used another data structure to perform the actual font manipulation. This data structure is called the **Font Path Element(FPE)**.

3.2.4. Font Path Element

Font Path Element is the core font accessing unit of the FML[32]. Basically, a font path element is an association of a font source with a Font Renderer. A font source can be a directory of font file or a remote font server. The FPE data structure contains function pointers for name-checking, initialization, open font, close font and list font, etc. These functions are provided by the two Font Renderers and must be registered to the FML before they can be used.

At the time the server starts, the font path is passed to the FML. The FML validates each element in the font path by calling the name checking routines registered in it in turn until the element is recognized or all name checking routines have been invoked. If the element is recognized by any one of the name checking routine, a new FPE is created. Otherwise an error message is returned. A list of active FPEs is thus set up. How the name checking routines validate the element in the font path is specific to the Font Renderer. For Font File Renderer, the first character of the name is compared with the character "/". For the Font Server Renderer, the first three character is compared with the string "tcp".

After establishing the FPEs list, each FPE is initialized by invoking the its own initialization routine. For the Font File Renderer, the file named *fonts.dir* in

each font directory is read while network connection to remote font server is established for Font Server Renderer[41].

3.2.5. Font File Renderer

The Font File Renderer is responsible for accessing font data from a local file system. The operation involved may be as simple as reading a bitmap font file or as complex as rasterizating an outline font, depending on the format of font files. Currently, the Font File Renderer of the sample server recognizes the following font file formats:

- . bnf Bitmap Distribution Format
- . snf Server Natural Format
- . pcf Portable Compiled Format, new in Release 5
- . spd Speedo outline format, also new in Release 5

A Font File Renderer also manages a set of **font readers** for retrieving and converting font data. These font readers must be registered to the font file renderer before they can be used. For each of the font file formats listed above, there should be a font reader registered. If a new font format is to be added to the font server, a new font reader should be registered to the Font File Renderer. An exploded view of the FML is shown in Figure 3-3.

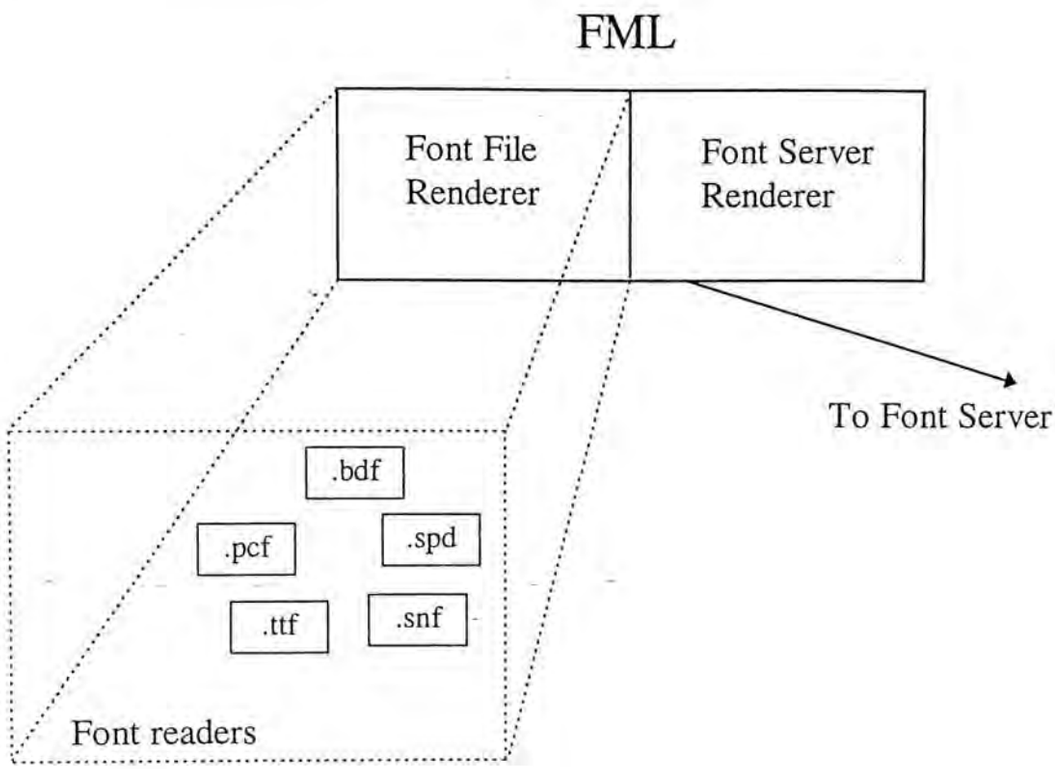


Figure 3-3 Font Dependent Renderers in FML.

3.2.6. Font Server Renderer

A Font Server Renderer is used for getting font data from other remote font servers. The Font Server Renderer allowed X server as well as font server to access font data located in a network font server.

3.3. Implementation of Chinese Font Server

3.3.1. Font data and code set

In current implementation, TrueType font format[29] is used for describing our Chinese font data. TrueType font format was originally defined by Apple Computer Inc. and has been widely used in Microsoft Windows. Since the introduction of Microsoft Chinese Windows[46][47], more and more Chinese

TrueType fonts have been designed and made available in the market. Choosing TrueType font format gives a wide set of font data for the testing of this font server.

The Chinese code set used by TrueType font in Microsoft Chinese Windows is called Big5 code. Standard Big5 code is a two byte code ranging from A140(hex) to F9D59(hex). The range of the first byte is from A1(hex) to F9(hex), with C7(hex) and C8(hex) unused. The legal second byte is 40-7E(hex) and A1-FE(hex). Totally, there are about 13000 characters in Big5 code set. The code ranges and classification of Big5 characters are summarized in Table 3-1.

Code Range	Usage
A140 -- A3BF	Non-Chinese Characters
A440 -- C67E	Frequently used Chinese Characters
C940 -- F9D5	Non-frequently used Chinese Characters

Table 3-1 Standard Big5 Code Range

However, since some commonly used Chinese characters such as 銹, 裏, 恒, 嫻, etc. are not defined in standard Big5 code set, TrueType does not use standard Big5 code. Instead, it uses an Extended Big5 code set[45] as its internal code. Extended Big5 code is almost the same as standard Big5 code except in the last code range. It defines the range F9D6(hex) to F9FE(hex) for some commonly used Chinese characters and symbols. The code ranges defined by Extended Big5 are shown in Table 3-2.

Code Range	Usage
A140 -- A3BF	Non-Chinese Characters
A440 -- C67E	Frequently used Chinese Characters
C940 -- F9FE	Non-frequently used Chinese Characters

Table 3-2 Extended Big5 Code Ranges

3.3.2. Registering a new font reader

The first step involved in the implementation of Chinese font server is to inform the font server that a new font file format is to be used. As revealed in the architecture of the font server, the function of reading font data from a local file system resides in the Font File Renderer. Unfortunately, the entry point for registering a new font reader to the Font File Renderer is not described in any available documentation. The source code of the sample font server implementation thus should be browsed.

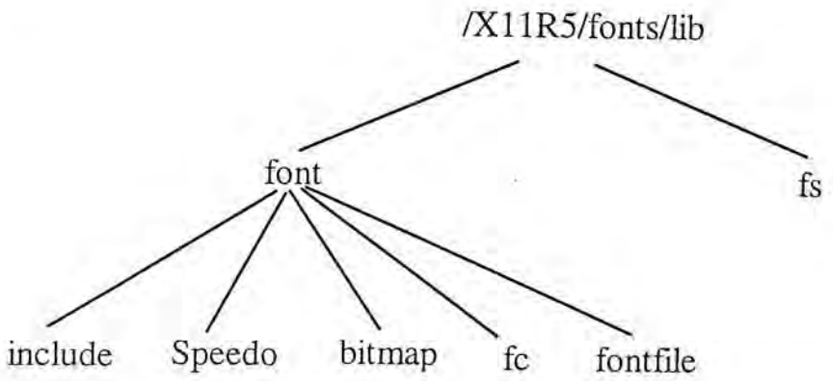


Figure 3-4 Directory Tree of FML.

The codes of FML in the sample font server are organized into a directory tree as shown in Figure 3-4. The top level directory is the */X11R5/fonts/lib*[3] . The second level includes two sub-directories. Directory *fs* maintains a self-contained library for accessing the remote font server. It provides high level interface for the requests defined in the X Services Protocol. This library can be used by any applications. The *font* directory contains all font access procedures used by the FML, including

[3] The name of the top level directory may not be the same across different implementations.

procedures for both local and remote font accesses. Since only the Font File Renderer's routines are of interest, discussion will be focused on the *font* directory.

The *font* directory is further divided into several sub-directories. The *bitmap* directory contains codes of bitmap readers for BDF fonts, SNF fonts and PCF fonts. In the *Speedo* directory, a font rasterizer for the Speedo fonts provided by Bitstream can be found. Speedo is the first outline font format contributed to the X consortium. The codes contained in that directory are a valuable reference for those want to embed new outline fonts reader to the FML.

The directory named *fontfile* is the place where the name checking, open font and close font routines of the Font File Renderer reside. It also contains procedures for registering new font reader. By modifying a function in the Font File Renderer called **FontFileRegisterFontFileFunctions**[4], a new font reader for Chinese font is registered.

The Chinese font reader should register the following information to the Font File Renderer:

- **".ttf"**, file suffix of the file format recognized by this font reader.
- **_OpenScalable**, a pointer to the function that will be called when a scalable font is to be opened

[4] The functions definition is in the file *renderers.c* in the *fontfile* directory.

- **_InfoScalable**, a pointer to the function that will return the scalable font information
- **_OpenBitmap**, a pointer to the function that will be called when a bitmap font is to be opened
- **_InfoBitmap**, a pointer to the function that will return the bitmap font information

The Font File Renderer differentiates whether the incoming font is bitmap font or scalable font. It also determines which function to call. In current implementation, the Chinese font reader supports outline fonts only. So only the functions to open outline fonts and query outline fonts information are implemented. They are called **TrueTypeOpenScalable** and **TrueTypeGetInfoScalable** respectively. The function pointers **_OpenBitmap** and **_InfoBitmap** are set to zero. The file suffix of TrueType font file is "ttf".

On receiving an open font request, DIFS calls all the active font path elements(FPE) in turn until a match is found or no more FPE remains. If the FPE is associated with the Font File Renderer, the font reader registered under the Font File Renderer whose file suffix equal to that of the font file being opened is invoked. and its function **_OpenScalable** or **_OpenBitmap** is called, depending on the nature of the font. If the called function returns properly, a successful message will be passed back to the DIFS together with information of the newly opened font. The sequence of control flow is depicted in .

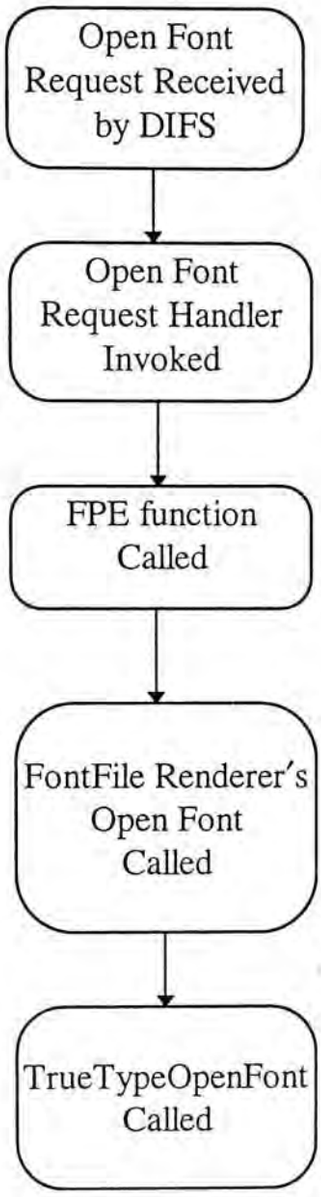


Figure 3-5 Control flow of open font request

The function **TrueTypeOpenScalable** is called whenever a font file with file extension ".ttf" is opened. If **TrueTypeOpenScalable** function returns successfully, a completed font record is also returned. This font record contains all information associated with the font such as the font metric information, properties associated with the font, code ranges, bit and byte order of the font, etc. Besides font information, font record also contains pointers to functions for obtaining font data and font glyphs information.

The completed font record will be returned to DIFS which attaches this font record to the record of the font client on which behalf the font is opened. Subsequent request for this font, such as getting bitmap data and querying font information, will be processed by the routines indicated in the font record.

The font reader decides when to build the character glyphs. In some implementation, all font data are built when a font is opened. However, due to the large number of characters in Chinese font, we defer the building of character glyphs until they are actually needed.

3.3.3. Font specific functions

Each font record must contain pointers to functions for getting bitmaps and extents. These two pointers are called **get_bitmaps** and **get_extents**. The functions pointed to are used directly by DIFS.

Whenever bitmap data are requested, the **get_bitmaps** function in the font record is called. The range of required character glyphs, bit order and byte order of the font client are also passed to this function. As the font server allows clients to get complete character set of a font or just a few characters, so a flag is also passed to **get_bitmaps** to indicate whether the complete character set or only a subset is needed. According to the value of this flag, we have two policies for font requesting. We call these **Load-All scheme** and **Demand-Loading Scheme**[4].

3.3.4. Load-All Scheme

As we have mentioned already, all font data are generated from outlines only when they are requested explicitly. So if a client attempts to get all the character bitmaps of the font in one request, the font reader must generate all these bitmaps when it receives that request. However, outline font rasterization is quite time-consuming. Generation of bitmaps for the whole Chinese character set (about 13000 characters) will result in a significant time delay.

We used our experimental font rasterizer to conduct an experiment to measure the average rasterization time for the whole Chinese character set. The program ran on a SPARC10 workstation with 64M RAM and a SPARCclassic with 16M RAM. Both machines were using SunOS 4.1.3. The times required are recorded. An average rasterization time is taken as the mean value of the results obtained from the two machines. Experimental results are summarized in Table 3-3. Note that this experiment measured only the font rasterization time. The time required to transfer font data from the font server to the client was not included.

Bitmap Size(in pixel)	Average Rasterization Time
20x20	7'32"
50x50	11'36"
100x100	18'24"
150x150	18'58"

Table 3-3 Experimental Result of Font Rasterization

As shown in Table 3-3, the rasterization time is quite long. And the time required increases as the size of bitmap increases. The long time delay is intolerable

in all cases. Unfortunately, the major client of the font server, the X server, tends to load all the characters when it opens a new font. So remedial action for this problem is necessary.

In current implementation, a whole set of 20x20 pixels Chinese character is pre-rasterized. Every time a font client wants to load all the characters, the font server reads in these pre-rasterized bitmaps and sends back to the client. This method reduces time delay significantly. However, as only one size is provided, clients requesting different font sizes may all get the bitmaps of the same size. A better solution is to provide pre-rasterized bitmaps of several different sizes and find the closest match for clients. But this method demands more hard disk space for storing bitmaps of different sizes.

3.3.5. Demand-Loading Scheme

Instead of a complete character set, a more flexible way for font loading is demand-loading. Font clients request only the character bitmaps actually used. This reduces the rasterization time greatly as only a relatively small number of bitmaps are needed. Clients which need a large among of character bitmaps may divide their request into several requests.

3.3.6. Embedding of font rasterizer

In current implementation, the embedding of a font rasterizer is simple and flexible. The actual implementation of the font rasterizer is not important as long as the rasterizer takes the font file name, the Big5 code and the bitmap size as function

arguments and returns the rasterized character bitmap. Any font rasterizer can be embedded into the font server if it obeys the interface shown in Figure 3-6.

```
char* FontRasterizer (char * FontFilename,
                      short Big5Code, int size);
```

Figure 3-6 The font rasterizer interface

Our experimental font rasterizer is originally for Intel-based personal computers. We ported and adapted it to our font server. This font rasterizer can handle any non-stroke-based Chinese TrueType font format[29].

The font rasterizer is invoked by the **get_bitmaps** function. It is the task of the **get_bitmaps** function to convert the font data produced by the font rasterizer into the format required by the font client. The conversion may include inversion of bit-order, padding extra bytes on each scanline and swapping 2 or 4 bytes of data. The **get_bitmaps** function should convert the bitmap data into the format as specified in the request sent by the font client. In order to simplify the conversion process, the sample font server has already included several routines for swapping bytes and re-padding scanlines, etc.[41].

3.4. Test Results

3.4.1. X Application Tests

In order to test the pre-rasterization scheme, X applications were used since an X server tends to load all font data when it opens a new font. Several applications

were used in the tests. The font server was firstly started on a SPARCclassic workstation equipping with 16M RAM and running SunOS 4.1.3. Two X servers, one on another SPARCclassic workstation and the other on a 486DX50 running Linux were started and connected to the font server. Two X based-applications namely CXterm and xfd were used. These two applications were tested on both X servers. Test results are shown in Figure 3-7, Figure 3-8 and Figure 3-9.

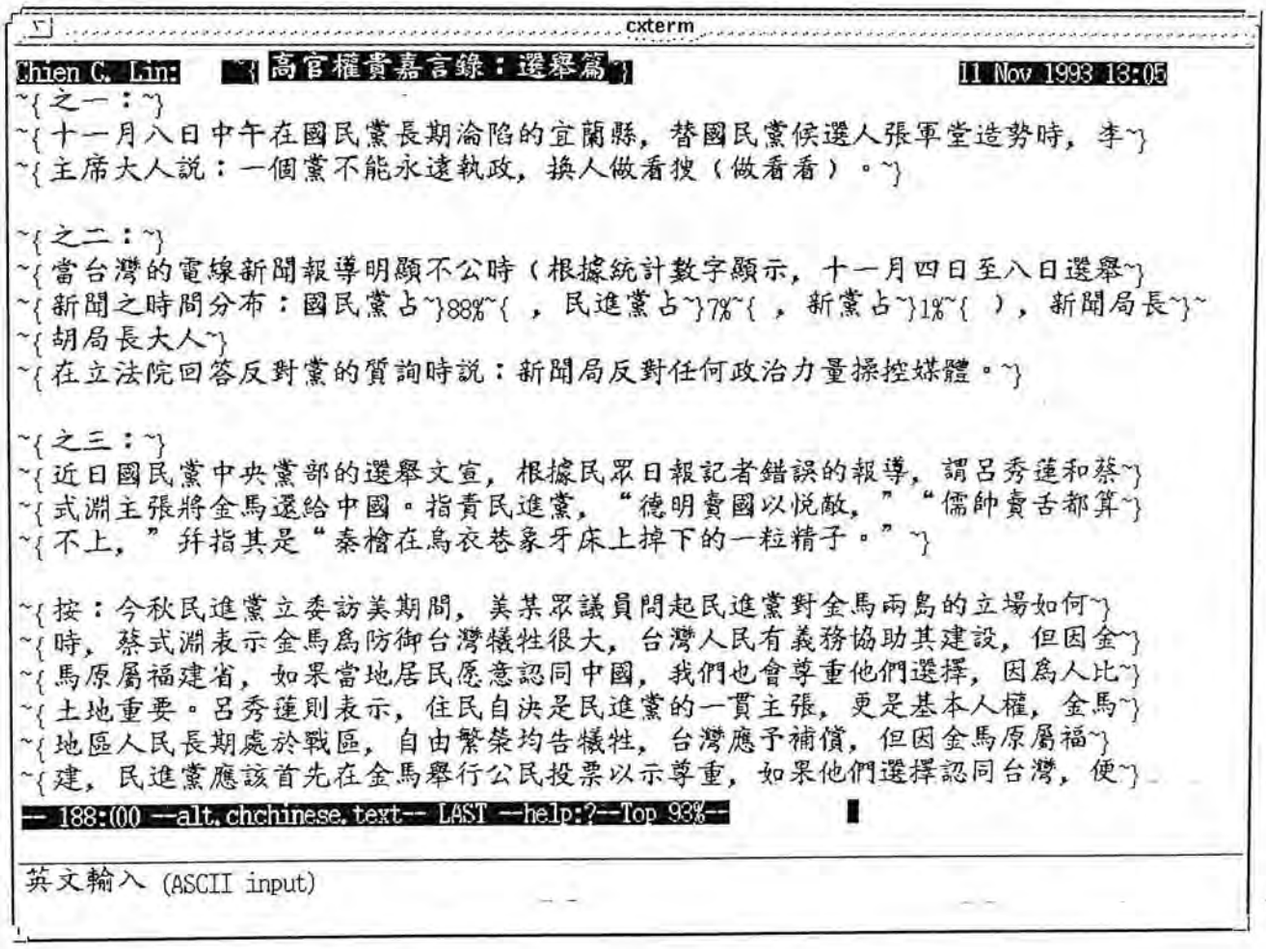


Figure 3-7 A session of CXterm reading Chinese News

The two X servers were used in the tests in order to test the ability of the Chinese font reader to handle font clients with different byte ordering. The SPARCclassic has a Most Significant Byte First ordering while the 486DX50 has a Least Significant Byte First ordering scheme. Test results show that the font server worked properly on both machines despite the difference in machine architectures.

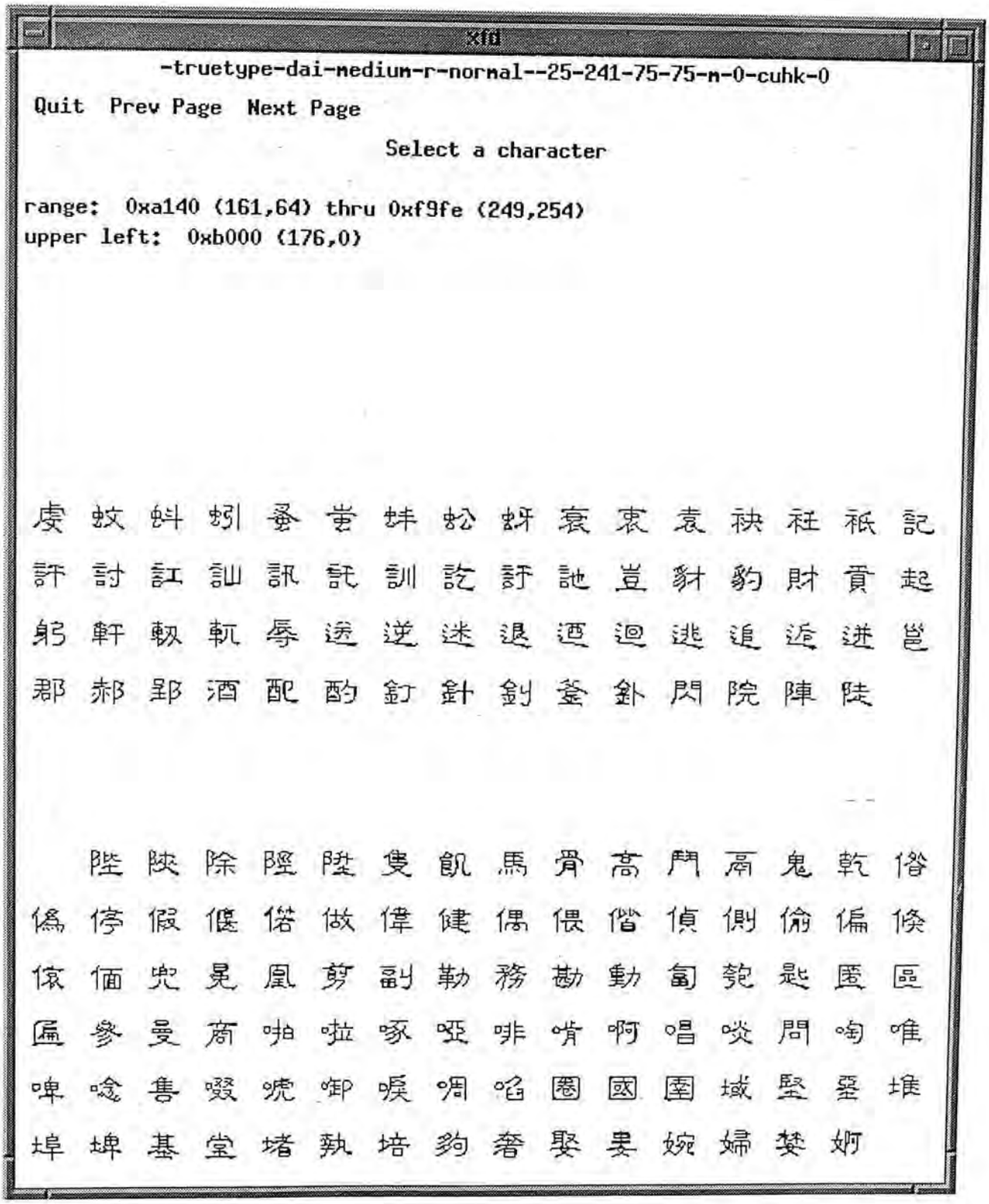


Figure 3-8 Xfd showing part of a font

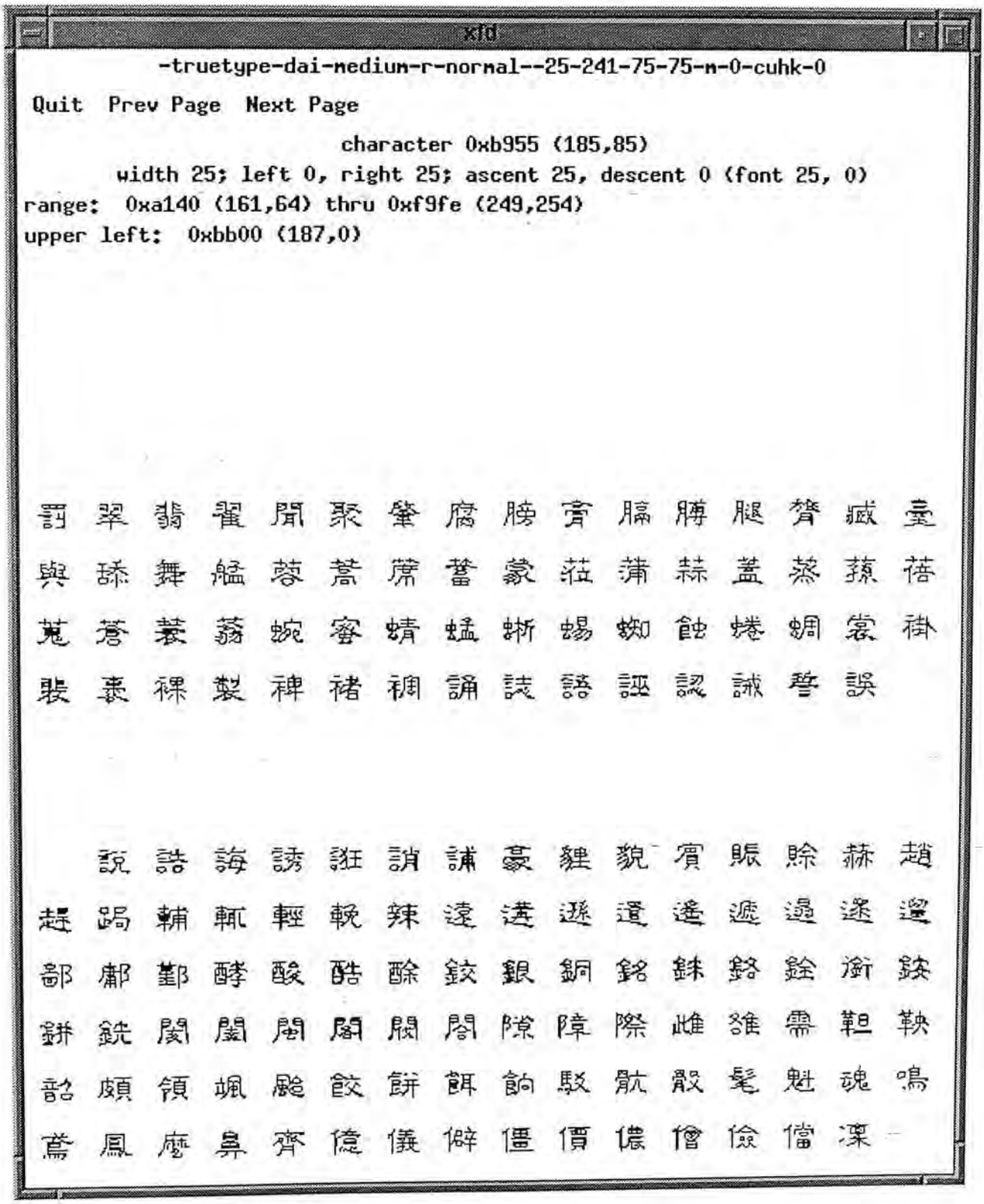


Figure 3-9 Xfd showing another part of the font

3.4.2. Demand-Loading Test

Another two programs were used to test the demand-loading capability of the font server. The first program simply read strings from a Chinese document and then requested the bitmap data of the strings from the font server. The time for getting all bitmaps of the characters in a Chinese document was recorded. Several documents with different numbers of characters were used.

Number of characters	Average Font Acquisition Time For 20x20 bitmap	Average Font Acquisition Time For 100x100 bitmap
181	4. 5"	7"
535	11"	23"
1046	13"	34"
2048	19"	1'7"
3641	22"	1'34"

Table 3-4 Experimental Result of Demand Loading

As shown in Table 3-4, the response times are much shorter than loading all characters. As an application loads only the bitmaps it actually needs, the rasterization time is thus much shorter.

The second program was indeed an X based version of the first one. In addition to reading strings and getting bitmaps, it also displayed the characters on a window. This program bypassed the X server font getting mechanism and acquired font data directly from the font server.

Three Chinese fonts, namely Dai(隸書), Sung(楷書) and Yuan(中圓) were used in the test. Some results of the tests are shown in Figure 3-10, Figure 3-11 and Figure 3-12.

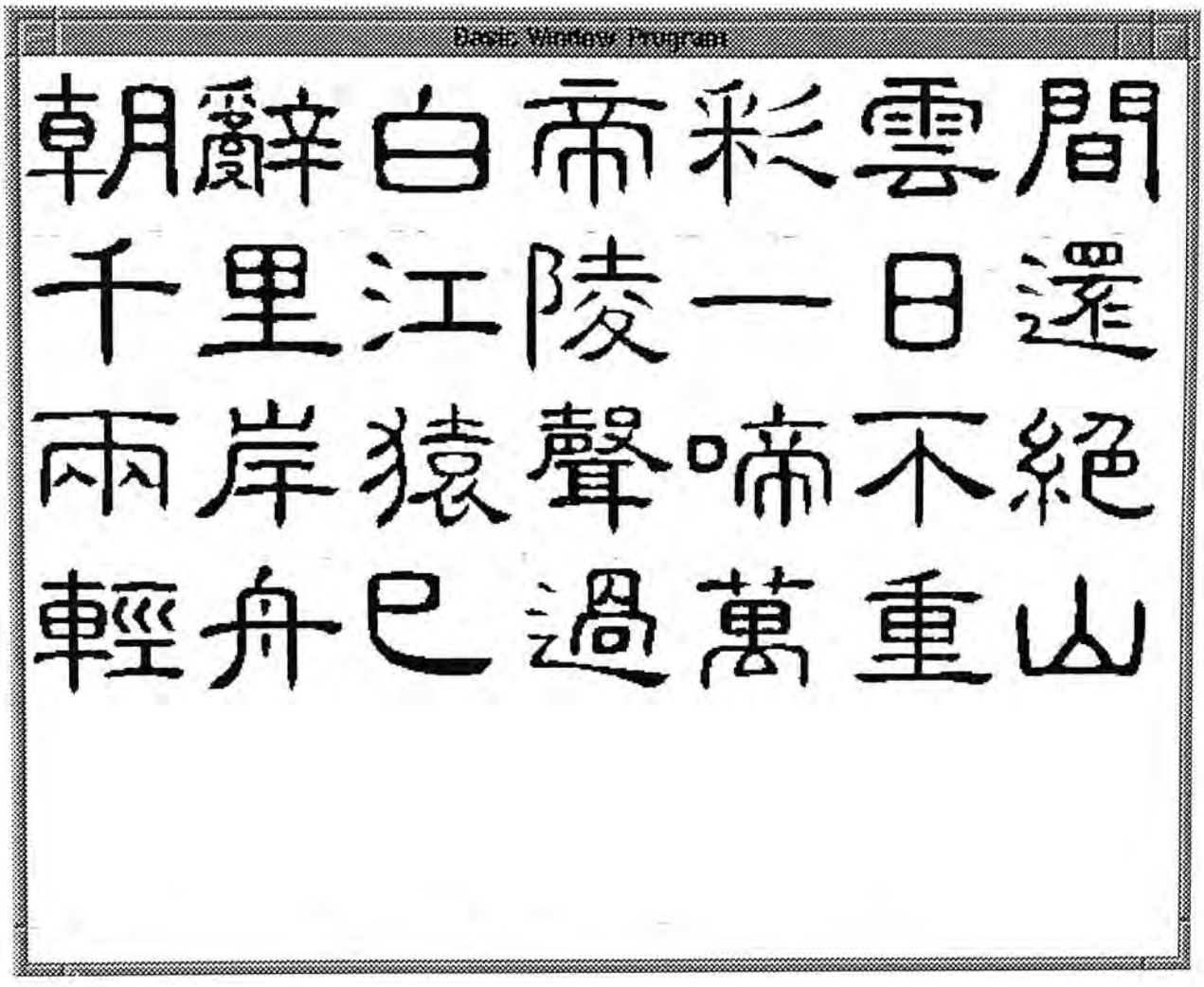


Figure 3-10 Chinese Font Display program using Dai Font (隸書)

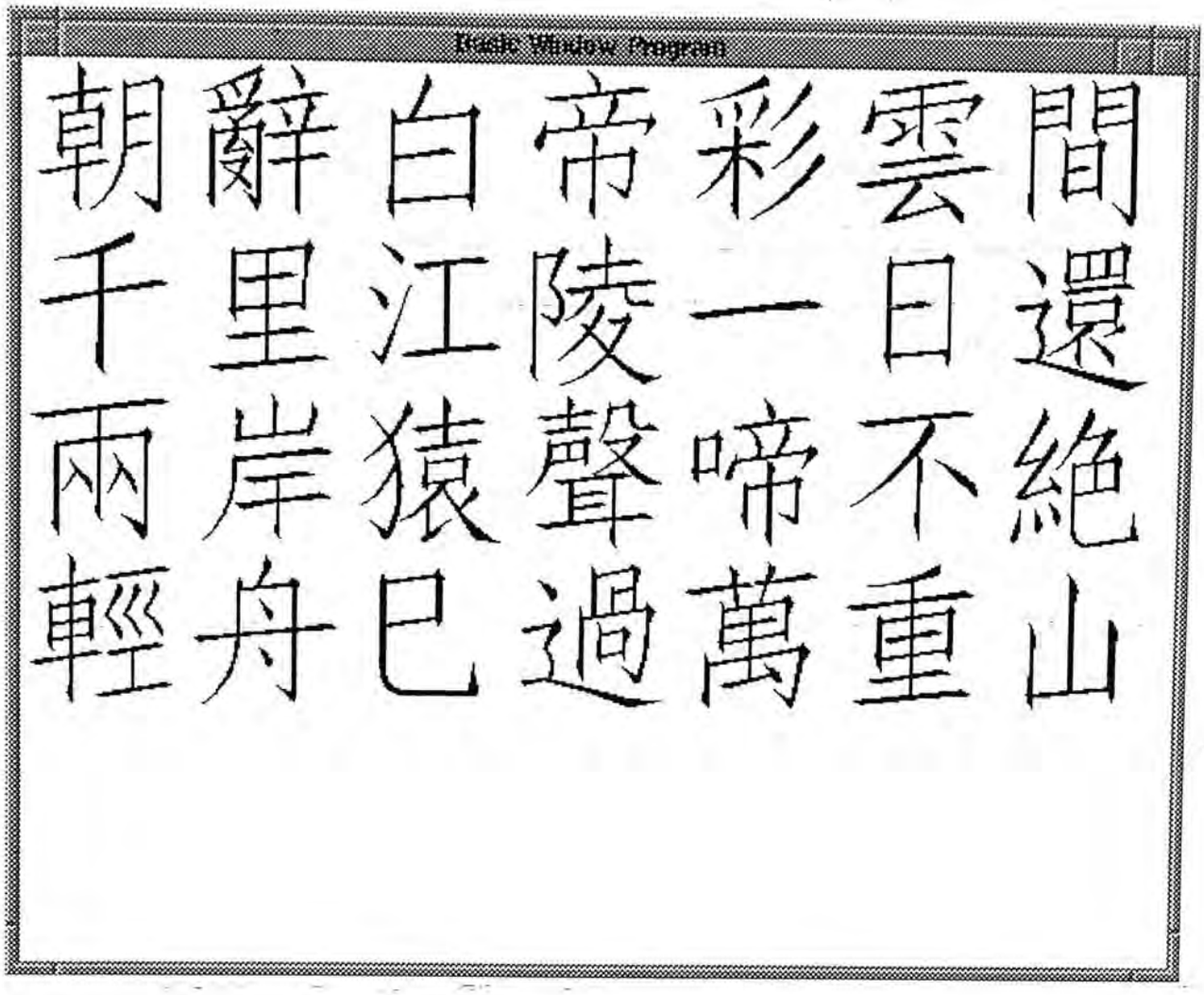


Figure 3-11 Chinese Font Display using Sung Font (仿宋體)

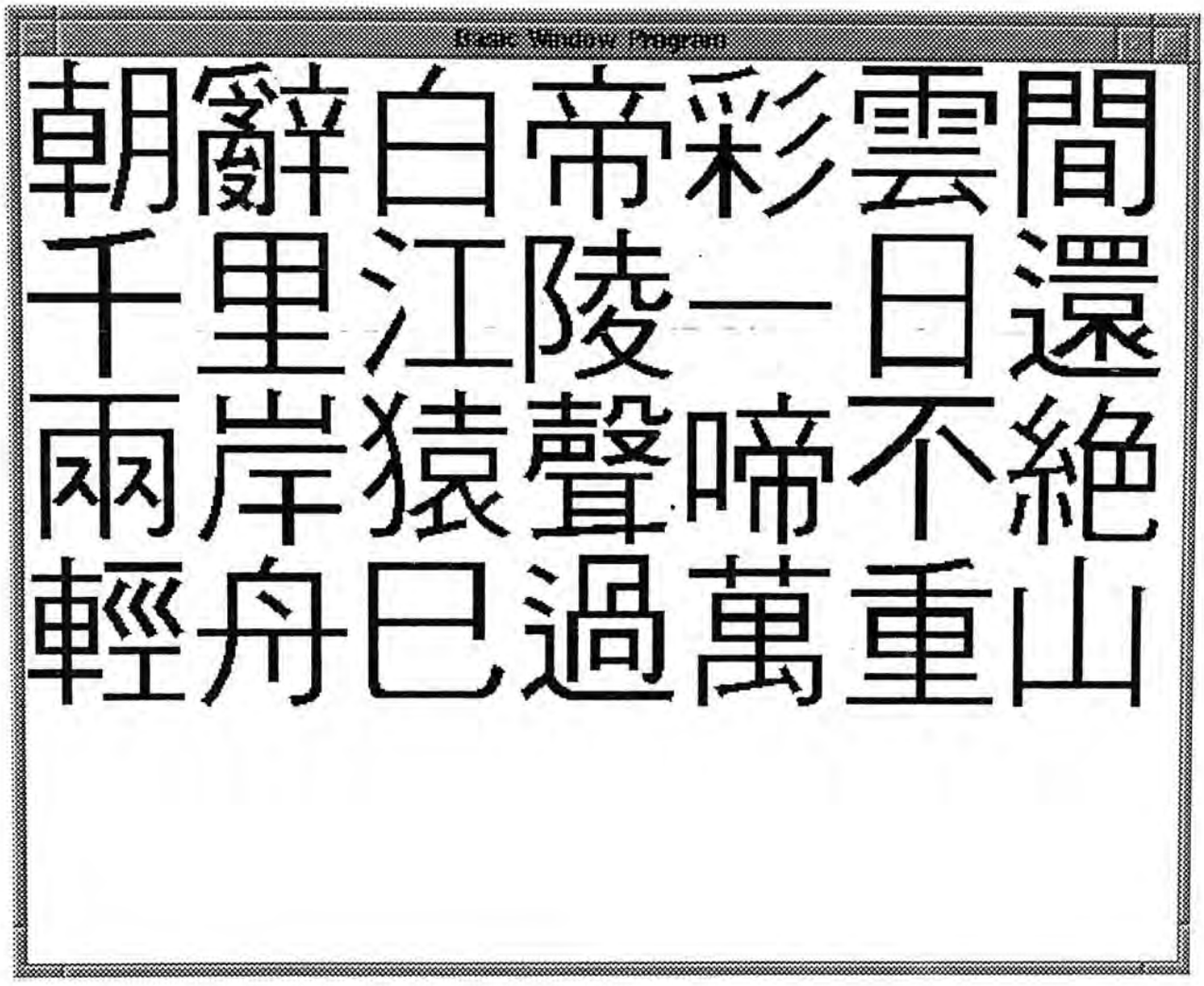


Figure 3-12 Chinese Font Display program using Chung Yuan(中圓體)

Testing results of these programs show us that fast X applications that request Chinese outline fonts can indeed be written by using the demand-loading feature of the Chinese font server.

3.5. Some Remarks

The experimental results reveal the deficiency of the font system of X11R5 in handling languages with large numbers of characters. The load-all-characters mechanism of the X server incurs great time delay in font acquisition when the number of characters is large, as in the case of Chinese font. The demand-loading tests show that efficient X-based Chinese applications can indeed be developed with better font management strategy, such as demand-loading. However, in X11R5, demand-loading can only be achieved if the application by-passes the font mechanism of X server.

Fortunately, the problem of load-all-characters strategy with languages with larger character set has been made known to the X Consortium. In the new upgrade of X Window System, the release 6 of Version 11, this problem is fixed. The X server can now be configured to used a demand-loading scheme instead of load-all-characters strategy when loading fonts with large numbers of characters. The improvement renders the development of efficient Chinese applications easier.

The implementation of Chinese outline font server widen the sources of font data that can be used while the improvement in font management in X makes font acquisition more effective and efficient. It is expected that more and more Chinese applications with attractive user interfaces and responsive to the user will be made available in the X world soon.

4. Overview of Printing System

4.1. Motivation

Some currently available windowing systems, such as Microsoft Windows and Macintosh system, provide device-independent application program interfaces to printing devices[1][25]. These interfaces allow applications on these systems to prepare printer output as easily as screen output. Furthermore, the application program interfaces shield much device dependent detail from applications. Applications can run smoothly in spite of any changes in printing devices.

Currently, X Window System provides no facility for applications to prepare printer output. Applications which are required to generate printer output should do all the work by themselves, which may include managing font data used, taking care of what kind of printer is being used and generating output in the format of the printer. This arrangement has several disadvantages. First, it places additional burden on the applications. Second, if more than one type of printer is being used, applications should be able to generate output of different formats. This makes adding a new printer to applications more difficult. Lastly, font data may not be shared among applications as they may have different formats for storing fonts.

The deficiency of supporting outline fonts in earlier versions of X may also account for the lack of printing facility in X Window System. As only bitmap fonts are used, the X server may need to maintain several copies of bitmap fonts in order to

entertain devices of different resolutions. This incurs a large storage problem in X when the number of fonts used is large.

The introduction of font server in Release 5 fixes this problem as new formats, bitmap or outline, can be easily incorporated into X. This advance in font management technology in X undoubtedly opens the door for implementation of the printing component of X.

In light of this situation, an experimental printing system is introduced. This printing system has the following design goals:

- Provides device independent printing interface for applications
- Allows font data to be shared among applications
- Allows new printers to be added to the system without any changes to applications
- Allows double-byte fonts (such as Chinese Fonts) to be handled with the same mechanism as single byte font

4.2. Design Considerations

To achieve the above design goals, several different approaches are considered. The following sections will discuss the relative merits of these designs.

4.2.1. Modification of the X server

The most straightforward way to incorporate the desired printing services is to modify the X server directly. In this design, the printing module is embedded as part of the X server. X client only needs to send printing requests to the X server which will do everything else on behalf of the X client. The X server may obtain the desired font resources from its local database or from a remote font server. The embedded module maintains a set of printer drivers which are responsible for all device-dependent operations pertaining to the printing requests. The embedded system is shown in Figure 4-1

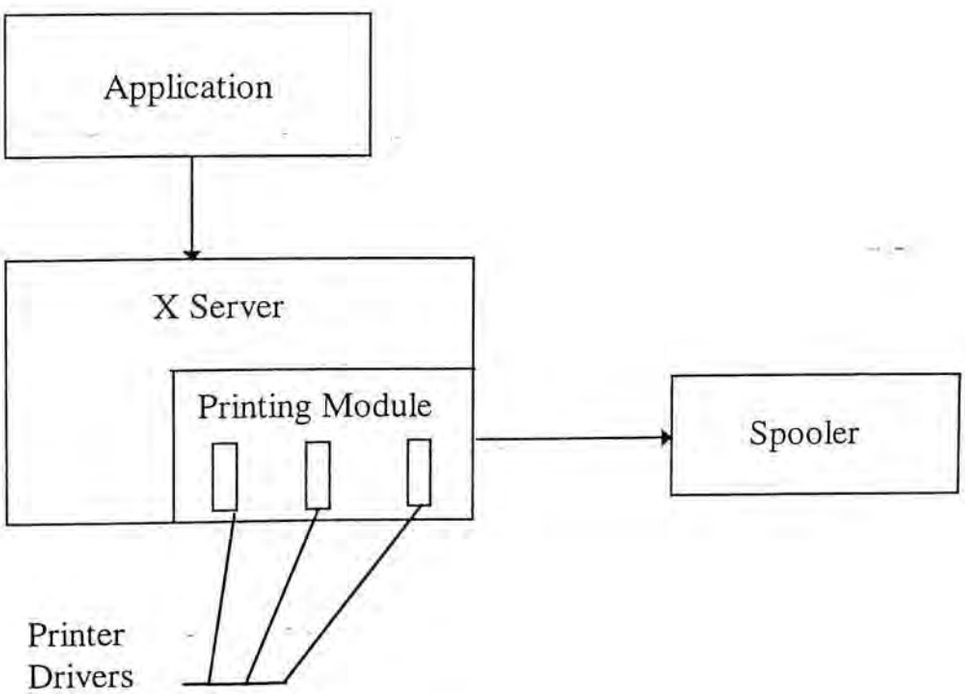


Figure 4-1 X server with embedded printing module

4.2. Design Considerations

To achieve the above design goals, several different approaches are considered. The following sections will discuss the relative merits of these designs.

4.2.1. Modification of the X server

The most straightforward way to incorporate the desired printing services is to modify the X server directly. In this design, the printing module is embedded as part of the X server. X client only needs to send printing requests to the X server which will do everything else on behalf of the X client. The X server may obtain the desired font resources from its local database or from a remote font server. The embedded module maintains a set of printer drivers which are responsible for all device-dependent operations pertaining to the printing requests. The embedded system is shown in Figure 4-1

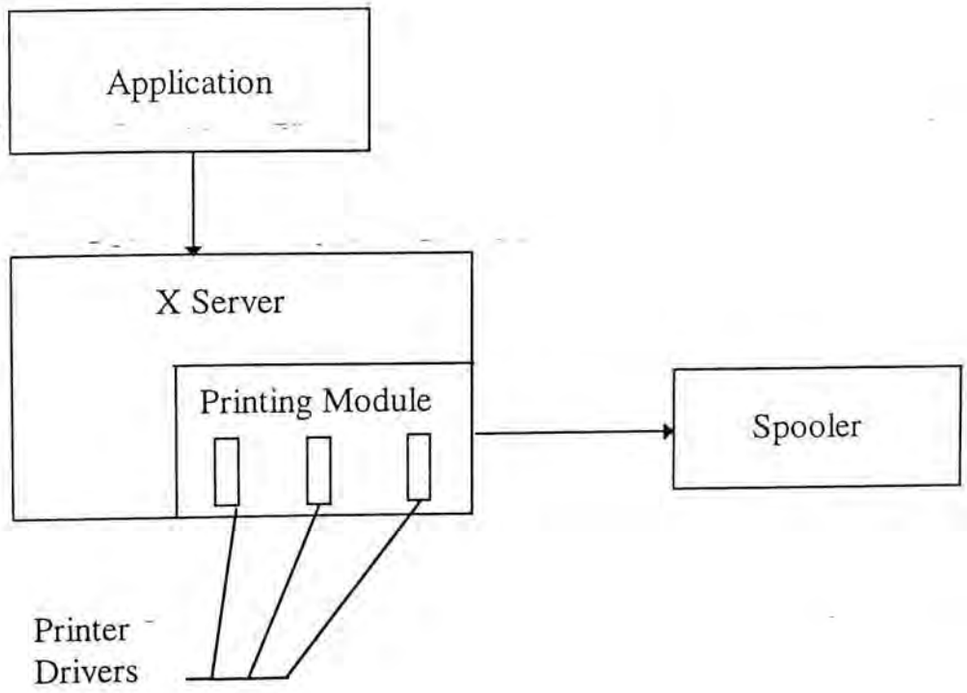


Figure 4-1 X server with embedded printing module

Despite the fact that it is conceptually simple and clean, this approach is difficult to implement. First, modifying the X server directly requires all X servers to be changed. Second, in order to address the new printing services, the X protocol must be changed also. Both factors induce a compatibility problem with the world-wide X society. Applications that run on the modified X server may not be able to run properly on another unmodified one. Lastly, as the printer drivers are attached to the X server, adding or removing a printer driver requires recompilation of the X server.

4.2.2. Embed the printing system into the font server

Another approach is to include the printing module into the font server, as shown in Figure 4-2.

In this design, client applications should establish connections with two different servers, one for screen display and the other for printer output. This should not be considered as a disadvantage as even in the previous design, applications should also generate two kinds of output with one for the screen and the other for printer output.

This approach provides higher flexibility than the previous one. When a new printer driver is added or an old one is removed, only the font server is changed. Moreover, as the printer drivers are now part of the font server, they can obtain the font data they need through several procedure calls only, not by network connection. This makes font acquisition more effective.

However, this design requires extension of the X Font Service Protocols to accommodate these new printing requests. This again induces compatibility problems with other font servers. Furthermore, letting the font server deal with two conceptually different kinds of requests, may not be a good choice.

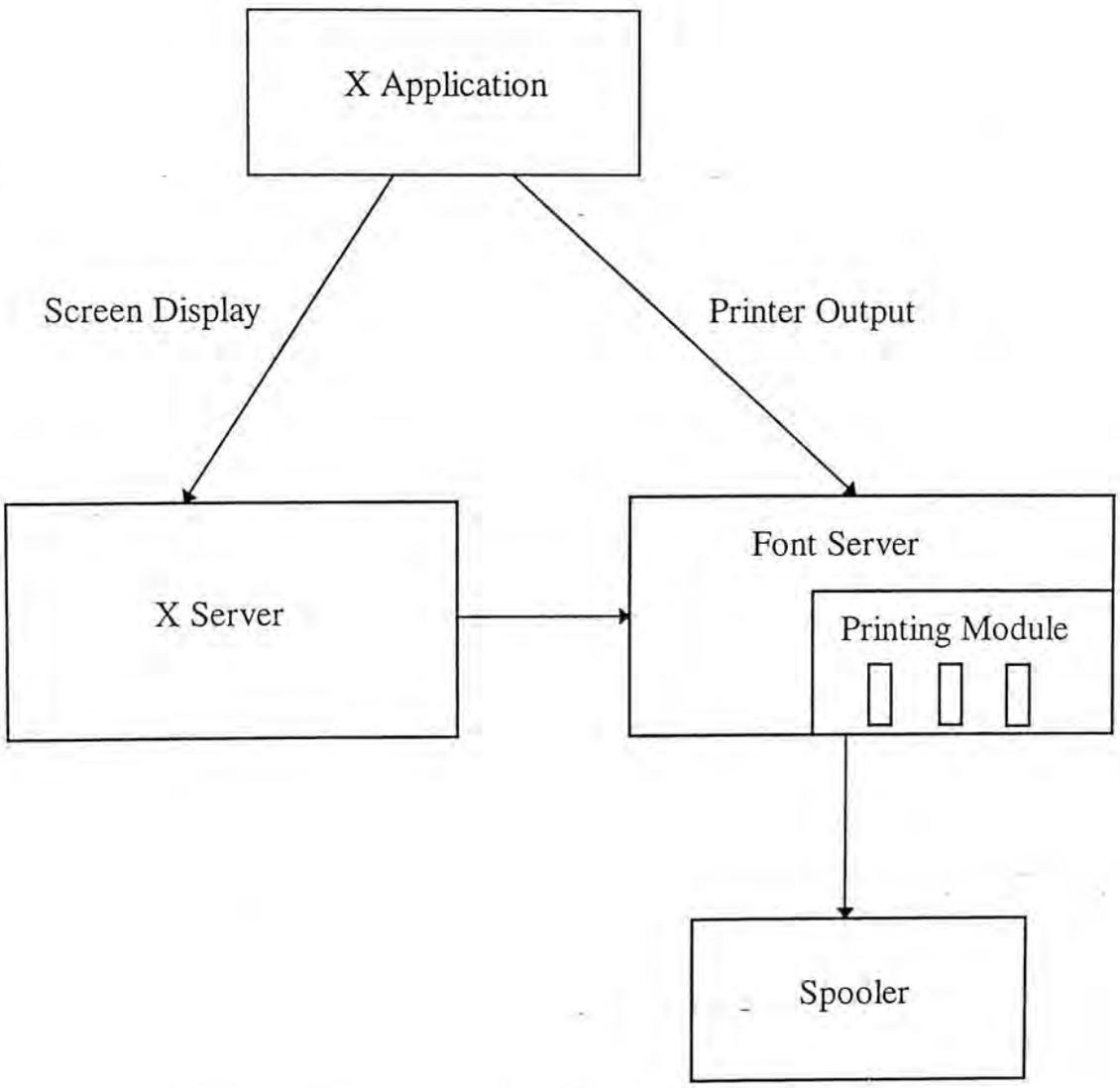


Figure 4-2 Font Server with embedded printing module

4.2.3. Distributed Architecture

The last approach moves the printing module out as a single process, as shown in Figure 4-3. In this design, the printing module appears as a stand-alone entity called Printer Server which does not require network support from any other

processes, such as X server or font server to provide communication channel for client applications. It does, however, rely on the font server (not shown in the figure) for font data. Font acquisition is less efficient than with the modified font server.

The introduction of Printer Server has several advantages over the previous two models:

- It keeps X server as well as font server unchanged.
- New printer drivers can be added with even a higher flexibility since only the Printer Server, not the X server or the font server, is recompiled.
- It provides higher extensibility as only the Printer Server is changed.
- Non X-based applications can also take advantages of the Printer Server

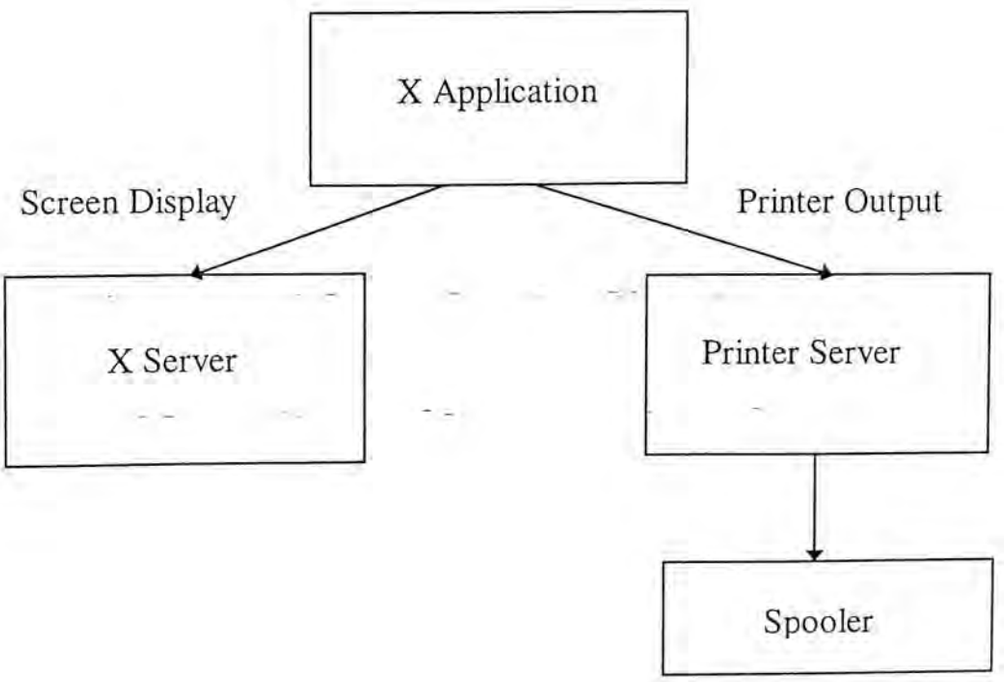


Figure 4-3 Printing services provided by a dedicated printer server

In light of the above comparison, this approach is adopted in current implementation of the printing system.

4.3. System Architecture

The design with a dedicated printer server is adopted in current implementation for its flexibility and extensibility. The printing system with a dedicated printer server is shown again in Figure 4-4, with an exploded view of client applications. The printing system consists of the following components:

- A Printer Server
- A Font Server
- An X Window System Server
- A Printing Services Protocol
- A Printer Server Library
- Client Applications

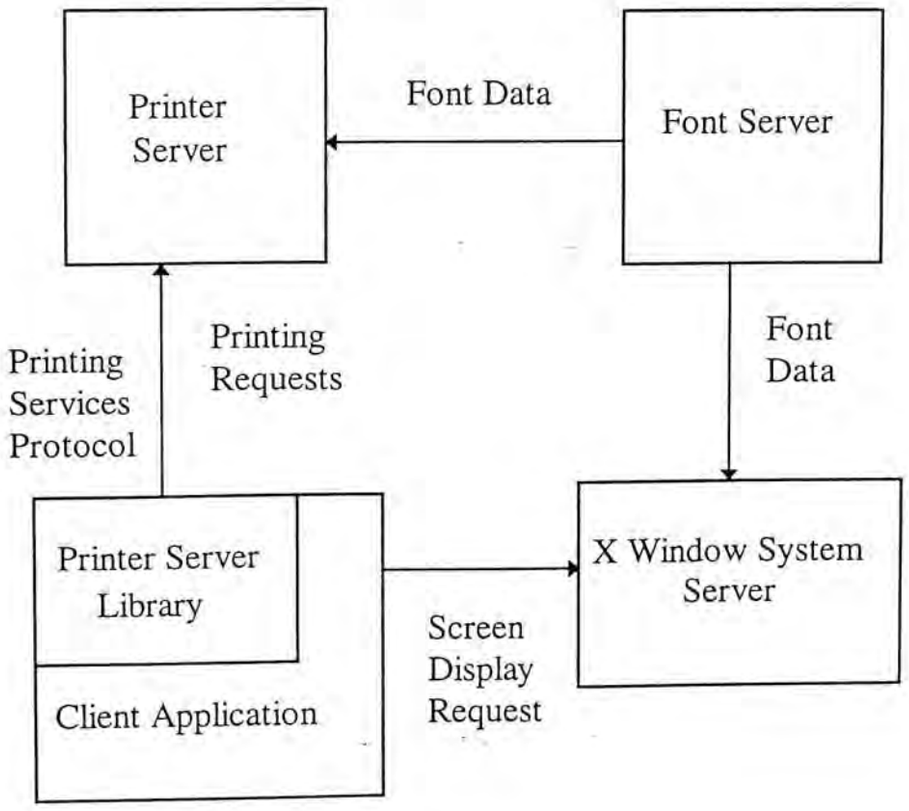


Figure 4-4 Architecture of Printing System

These modules need not run on the same machine. They can communicate with each other by using the network facilities. Note that communication with the X server is defined by the X Window System Protocols[34] and that with the Font Server is defined by the Font Services Protocols[11].

4.4. Printer Server

The Printer Server is the central printing component of the whole system. It is responsible for managing print jobs of client applications. Whenever a client application performs a printing operation, it sends a request specifying what operations it wants to the Printer Server. The Printer Server will in turn perform the requested operations for that client. The Printer Server shields much operation details such as retrieving font data, invoking the printer driver, checking printer memory and updating printer font record etc. , from the client applications.

The Printer Server includes a list of printer drivers which are responsible for actual output generation. Each client application must select a printer before any output operation can be performed. All subsequent output operations will be directed to the printer driver associated with the client application.

Basically, the operation of the Printer Server includes checking for any client request, dispatching the request to the responsible printer driver and, after the request is processed, sending a reply to the client.

The Printer Server also maintains a list of client records for storing information of each client application. Each client record contains information such as the current printer used and the communication channel owned by the client. The Printer Server uses all this information to determine which printer driver to call, which font to use and to which print job the operation is applied.

The Printer Server is not supposed to have the ability to generate font bitmap data. Instead, it requests font bitmap data from the font server, and this will be discussed in Section 4.5. However, in order to reduce network traffic and time delay in font acquisition, the Printer Server will include a local font cache which contains bitmap data of several recently used fonts. The Printer Server should control the amount of memory used by the font cache and the number of fonts stored in the cache.

As the current Printer Server runs on top of the UNIX operating system, it is supposed that the Printer Server should be compatible with the current spooling system of UNIX[37], i.e., the Printer Server should not interact directly with the printers. Instead, it generates printing output files and sends the files to the print queue for printing.

The Printer Server must be started first before any client application connects it. In the experimental implementation, client applications should know in advance at which machine the Printer Server is located and which port number[37] the Printer Server is listening to.

4.5. Font Server

The font server is the central font provider of the system, and almost all font data is generated by it. As mentioned in Section 4.4, the printer server itself does not generate any font data, it relies entirely on the font server to provide printer font data.

Although the X Window System server may contain some font data in its local font database, the use of a font server is recommended. This is because if the Printer

Server and the X Window System server are both using the same font server, the character images displayed on screen may match those printed on paper. This makes the implementation of WYSIWYG applications possible.

Uses of a font server also makes the Printer Server system less dependent on the X Window System. As mentioned in Chapter 3, any application can connect the font server as long as it obeys the font server protocol. The separation of font provider from X Window System server allows non-X-based applications to take advantages of the Printing System.

4.6. Printing Services Protocols

The Printing Services Protocols define the communication protocols between client applications and the Printer Server. In current experimental implementation, the following requests are defined:

- ConnectServer

- setup connection with the Printer Server

- DisConnectServer
  - disconnect from the Printer Server
- QueryPrinterState
  - query the current setting such as PaperSize, PaperDirection etc.
- PrinterEscape
  - sending printer escape such as STARTDOC, ENDDOC and RESET where STARTDOC signals the printer to start a new document, ENDDOC signals the end of the article and RESET informs the printer to reset
- SetPrinterState
  - set the printer setting
- ListFont
  - list available fonts
- LoadFont
  - request the Printer Server to load a font
- CloseFont
  - request the Printer Server to close an opened font
- SetFont
  - set the current font used
- GetTextExtent
  - query the metric information of a specified string
- DrawString
  - draw a specified string

- PrinterEjectPage
  - request the printer to eject current page

4.7. X Window System Server

The X Window System Server is used for screen output display. As have been mentioned in Section 4.4 and 4.5, the Printing System is designed to be as independent of the X Window System server as possible. So applications which would like to use their own display routines can indeed bypass the X Window System Server.

4.8. Printer Server Library

The Printer Server Library contains routines for sending requests to the Printer Server and reading replies from Printer Server. This library shields the network communication details from client applications. From the client applications' point of view, sending requests to the remote Printer Server just like making a normal function-call. When the function-call returns, it may contain the information needed by the client applications or an acknowledgment from the Printer Server. This library should be linked to client applications at compile time.

4.9. Client Applications

To take advantages of the Printer Server System, applications should include the Printer Server Library when it is being developed. On the other hand,

- setup connection with the Printer Server

- DisConnectServer
  - disconnect from the Printer Server
- QueryPrinterState
  - query the current setting such as PaperSize, PaperDirection etc.
- PrinterEscape
  - sending printer escape such as STARTDOC, ENDDOC and RESET

where STARTDOC signals the printer to start a new document, ENDDOC signals the end of the article and RESET informs the printer to reset

- SetPrinterState
  - set the printer setting
- ListFont
  - list available fonts
- LoadFont
  - request the Printer Server to load a font
- CloseFont
  - request the Printer Server to close an opened font
- SetFont
  - set the current font used
- GetTextExtent
  - query the metric information of a specified string
- DrawString
  - draw a specified string

applications are unaffected by any changes in printer driver attached to the printer server as all printer drivers provide the same set of printing interface for applications.

In the current design, the screen display routines are provided by the X Window System server. However, client applications may also use other systems to produce screen output.

In spite of the screen display routines used by an application, it is highly recommended that both the screen display routines and the Printer Server use the same font server to obtain font data so as to achieve WYSIWYG effect.

5. Design and Implementation of a Printer Server

It is always of great difficulty to design a system with considerable complexity. In the design and implementation of the Printer Server, an object-oriented approach is adopted. Object-oriented method is employed to simplify the design process. The main advantage of this design methodology is its emphasis on individual objects' behavior. The design focuses on the objects which altogether constitute the whole system, their behaviors and their relationships. The whole system is broken down into several sub-systems which in turn are divided into several objects. From the implementation point of view, as each object responds only to the message it receives, regardless of the nature of the client from which the message is sent and why the message is sent, each object can be coded and tested individually and independently[22]. Moreover, object inheritance allows objects with some degree of similarity to be grouped into a class, saving the efforts to write similar codes again and again[38].

Like most object-oriented system, the whole process of software construction is divided into an objects identification phase and an objects organization phase[24].

5.1. Objects identification

In the first stage of design, the classes involved are identified. The following section is devoted to describe the classes and their responsibilities. Note that the names of the identified classes are given in parenthesis next to the concept they represent.

5.1.1. Dispatcher (dispatcher)

Dispatcher is the main control unit of the Printer Server. Its responsibilities include:

1. Initializing the Printer Server
2. Checking for client connection request
3. Maintaining a set of interested channels on which requests from client are expected
4. Checking for client operation requests and dispatching the requests to the request handlers
5. Scheduling the sequence of processing of client requests
6. Resetting the Printer Server

5.1.2. Communication Channel (ComChannel)

The ComChannel class encapsulates the network communication details for its users. Reading and writing of channel appear only as normal file accesses. The responsibilities of the ComChannel include:

1. Establishing connections between clients and Printer Server
2. Providing reading and writing routines

The ComChannel provides only byte-streamed reading and writing routines. It does not define any message boundaries. The formats of messages are defined and enforced by users of the ComChannel.

5.1.3. Font Cache Manager (FnCache)

The FnCache class maintains the font resources of the Printer Server, and all other modules of the Printer Server should only request font data and font information from the FnCache. Its responsibilities include:

1. Establishing and discarding connections with remote font servers
2. Returning font data and font metric information on request of other module
3. Maintaining the state of its internal font cache
4. Controlling the amount of memory used by cache
5. Facilitating font sharing among clients

5.1.4. PrnFont (PrnFont)

The main design purpose of the PrnFont class is to maintain a per-font cache pool to cache up font bitmap of one font. Its basic operations includes:

1. Maintaining information of the cached font
2. Returning font bitmap and font information to users
3. Requesting font bitmap and font information from remote font servers
4. Updating the state of its per-font cache pool

The set of all PrnFonts constitute the font cache of the Printer Server. These PrnFonts are maintained by the FnCache. Note that the PrnFont class determines when to update its per-font cache pool but not how to update. The details of the replacement policy reside in the CacheStruct class described below.

The division of the large cache pool into several small PrnFonts simplifies the management of the font cache. The FnCache only determines the total amount of memory used by each PrnFont, and the details of each PrnFont is transparent to it. Moreover, the use of memory is also more flexible than allocating a large pool as additional memory is required only when a new PrnFont is opened. The replacement of cached bitmap would not consume additional memory as everything are done within the amount of memory allocated to each PrnFont.

5.1.5. Per-Font Cache (CacheStruct)

The Per-Font Cache is responsible for enforcing the replacement scheme of the Font Cache. Its responsibilities include:

1. Returning font bitmap data that are requested by the PrnFont
2. Maintaining the font bitmap into a font cache
3. Updating the contents of the cache by using the pre-defined replacement algorithm

By isolating the replacement policy from the PrnFont, several different caching algorithms can easily be implemented. As long as the CacheStruct class interfaces are unchanged, any modification to the CacheStruct class is transparent to the PrnFont class. Different caching polices can easily be incorporated into the font cache by subclassing from the CacheStruct class.

5.1.6. Font Server (FnServer)

This is a C++ wrapper over the C source library provided by X Consortium, whose only task is to provide interface functions for accessing remote font server.

5.1.7. Client Manager (LRUList)

The client manager is responsible for keeping track of the clients which are currently using the Printer Server. Its operations include:

1. Maintaining the set of connected clients
2. Adding a new client
3. Removing a disconnected client
4. Returning identification code of each client to users

5.1.8. Client Record (ClientRec)

The ClientRec class contains status and information of the client. These include the printer the client used, the client number and the identity of the communication channel etc. In fact, not much is done directly by the ClientRec class. Its sole responsibility is to maintain information peculiar to the client.

5.1.9. Printer Driver (PrnDriver)

The PrnDriver class takes over all device dependent operations concerning a print request. Its main-task is to provide routines for translating device independent print requests into printer commands that can be understood by a specific printer. For each new printer type attached to the Printer Server, a subclass of the PrnDriver must be provided. Operations of PrnDriver includes:

1. Keeping track of the printer state such as available memory, paper size etc.
2. Providing routines for generation of printer output

5.1.10. Down Loaded Font Table (DownLoadedFont)

The DownLoadedFont class records the information of the downloaded fonts of a printer. It is used by the PrnDriver class to maintain its downloaded fonts. The responsibilities of this class includes:

1. Maintaining a set of downloaded fonts for a specified printer
2. Keeping track of the memory used by each downloaded font
3. Maintaining information of each downloaded font to facilitate double-byte fonts downloading

5.1.11. Request Header (reqHeader)

The reqHeader class is one of the important modules of the Printer Server. It defines the formats of messages exchanged between the Printer Server and its clients. Its basic operations include:

1. Writing request to server
2. Reading requests from client
3. Cooperating with other classes to process the request

As has been described in Chapter 4, currently the Printing Services Protocol defines 12 different requests. Each request required different number and types of parameters. The values returned from different requests are also different. From an object-oriented point of view, only the object itself knows what information to write

to the Printer Server and what to expect from the clients. Moreover, each request may employ a processing mechanism which varies from requests to requests. It follows naturally that the reading, writing and processing routines a of request are contained in the request itself. The reqHeader class is thus subclassed in order to support the different requests, as shown in Figure 5-1.

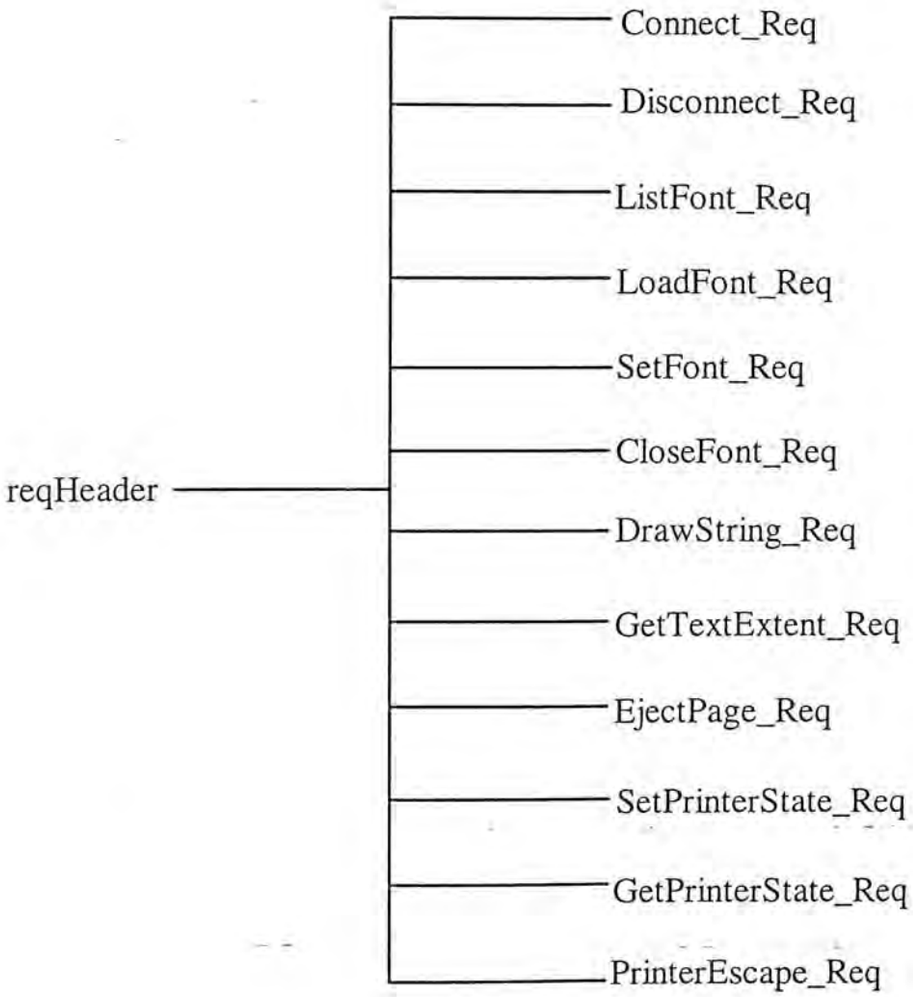


Figure 5-1 Inheritance in the request class

The reqHeader class is used by both the Printer Server and the Printer Server Library. The Printer Server uses it to read requests from the client while the Printer Server Library (i.e. the print client) uses it to write requests to the Printer Server.

5.1.12. Generic Reply(replyGeneric)

Similar to reqHeader, replyGeneric class defines the replies which are sent back to the print clients. Its basic operations include:

1. Writing reply to client
2. Reading reply from server
3. Processing of reply

The replyGeneric class is also subclassed to support the different kinds of replies sent back to client. However, since most of the replies simply return an error code to indicate whether the request is processed successfully or not, only a few replies are different from the base class. For those replies which need special handling, a subclass of the replyGeneric is defined. The class hierarchy of the replyGeneric class is shown in Figure 5-2.

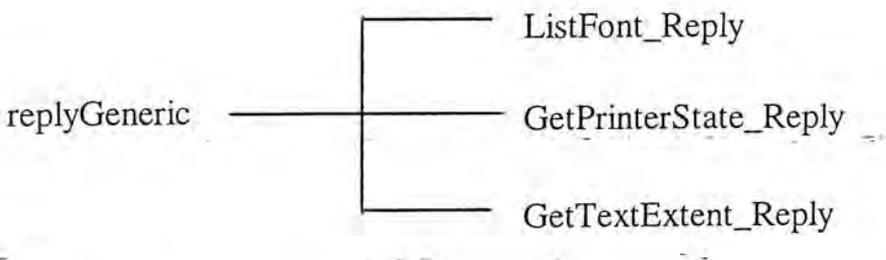


Figure 5-2 Class hierarchy in replyGeneric class

5.2. Objects Organization

After the identification of objects, the next step involved is to organize them properly. As can be seen in the objects identification phase, several concepts are

more important than others. These concepts usually involve more than one classes of objects or related to critical operations. These concepts are listed below:

- Server Control
- Client Management
- Request Handling
- Font Management

The objects identified in previous phase are organized around these concepts and subsystems are constructed for each concept. The relationships among these subsystems are shown in Figure 5-3. Note that Figure 5-3 shows only an overview of the system; the detail messages transfer will be described in the following sections.

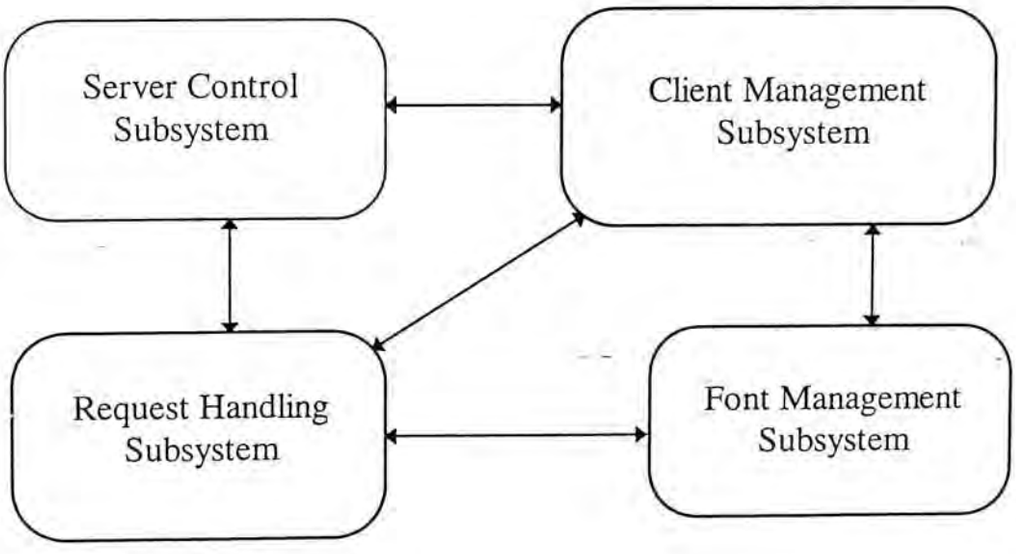


Figure 5-3 Architecture of the Printer Server

5.2.1. Server Control Subsystem

The server control subsystem consists of the dispatcher class and the communication channel class. The dispatcher class responds mainly to three

messages, namely DIS_INIT, LOOP and RESET while the ComChannel responds to, in this case, the ACCEPT and COM_INIT messages. The message diagram of the server control system is shown in Figure 5-4. Note that in Figure 5-4 all message names are in capital letters. This typographic convention will be used throughout the chapter.

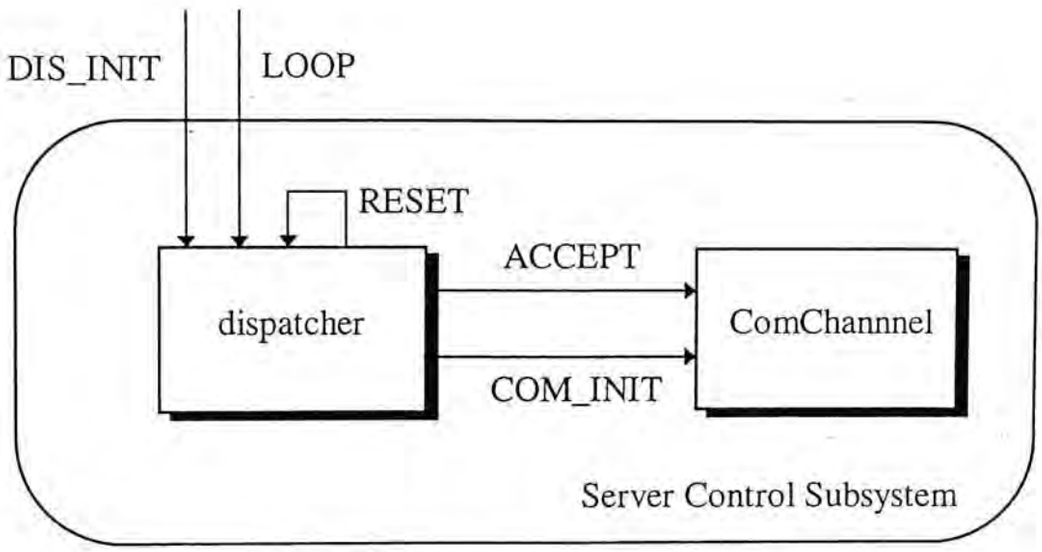


Figure 5-4 Message Diagram of the Server Control Subsystem

In response to the DIS_INIT message, the dispatcher performs printer server initialization. Printer Server initialization includes instantiation of its internal objects, sending the name list of active font servers to the Font Management Subsystem for font server connection establishment(will be discussed later) and setting up its own communication channel for client connection by sending a COM_INIT message to the ComChannel class.

The dispatcher enters its main processing loop on receiving a LOOP message. In this processing loop, the dispatcher listens indefinitely on the set of interested channels. The set of interested channels include both the channels used by print

clients to send requests and the channel dedicated to connection establishment. If any one of the channels is ready for processing, the dispatcher will ask the Client Management Subsystem to identify the requesting client and invoke the Request Handling Subsystem to process the request.

The dispatcher should also enforce policy to make sure that a single client cannot dominate the printer server. When several requests are waiting to be handled, the dispatcher should take control back from one client to ensure fair service among clients. Priority can be assigned to a client when needed. In this design, all clients bear the same priority.

After the dispatcher has been idle for a long period and no print client is currently connected to the Printer Server, the dispatcher will refresh the Printer Server by sending itself a RESET message. The RESET message instructs the dispatcher to clear all interested channels except the one for print client connection. The dispatcher will also send a message to the Font Management Subsystem to inform it to clear its font database and reconnect the font servers.

The dispatcher also cooperates with the Client Management Subsystem to maintain the state of each client. When a new client application sends a connect request to the server, the dispatcher invokes the Client Management to initiate a new client record for that client application. When a client application disconnects, the dispatcher sends a message to the Client Management Subsystem to inform it to discard the client record of this client application and to free all resources allocated to this client application.

On receiving a COM_INIT, the ComChannel object performs the necessary steps required to establish a network channel from which all print clients can connect the Printer Server. This channel number is the Printer Server's listen port. In current design, this port is fixed.

The ACCEPT request informs the ComChannel that the listen port is ready for processing. The ComChannel then processes the listen port, prepares a channel for the print client and returns the number or identification of the channel to the dispatcher. This identification will be passed back to the dispatcher and will be used when a new client record for this client is created.

5.2.2. Client Management Subsystem

The Client Management Subsystem involves the following classes, namely the Client Manager(LRUList), the ClientRec, the ComChannel, the PrnDriver and the DownLoadedFont. The organization of the Client Management Subsystem is shown in Figure 5-5.

The LRUList class is responsible for keeping track of the state of all client applications connected to the Printer Server. It maintains information of each client application into a list of ClientRec objects, with one object for each client application. Every time a new print client connects the Printer Server, the dispatcher sends an ADD_REC request to the LRUList which in turn creates a new ClientRec object for that print client. This newly created ClientRec object is also added to the

LRUList. Similarly, when a print client discards, the dispatcher sends a DEL_REC request to the LRUList to remove the ClientRec object from the list and destroy that object.

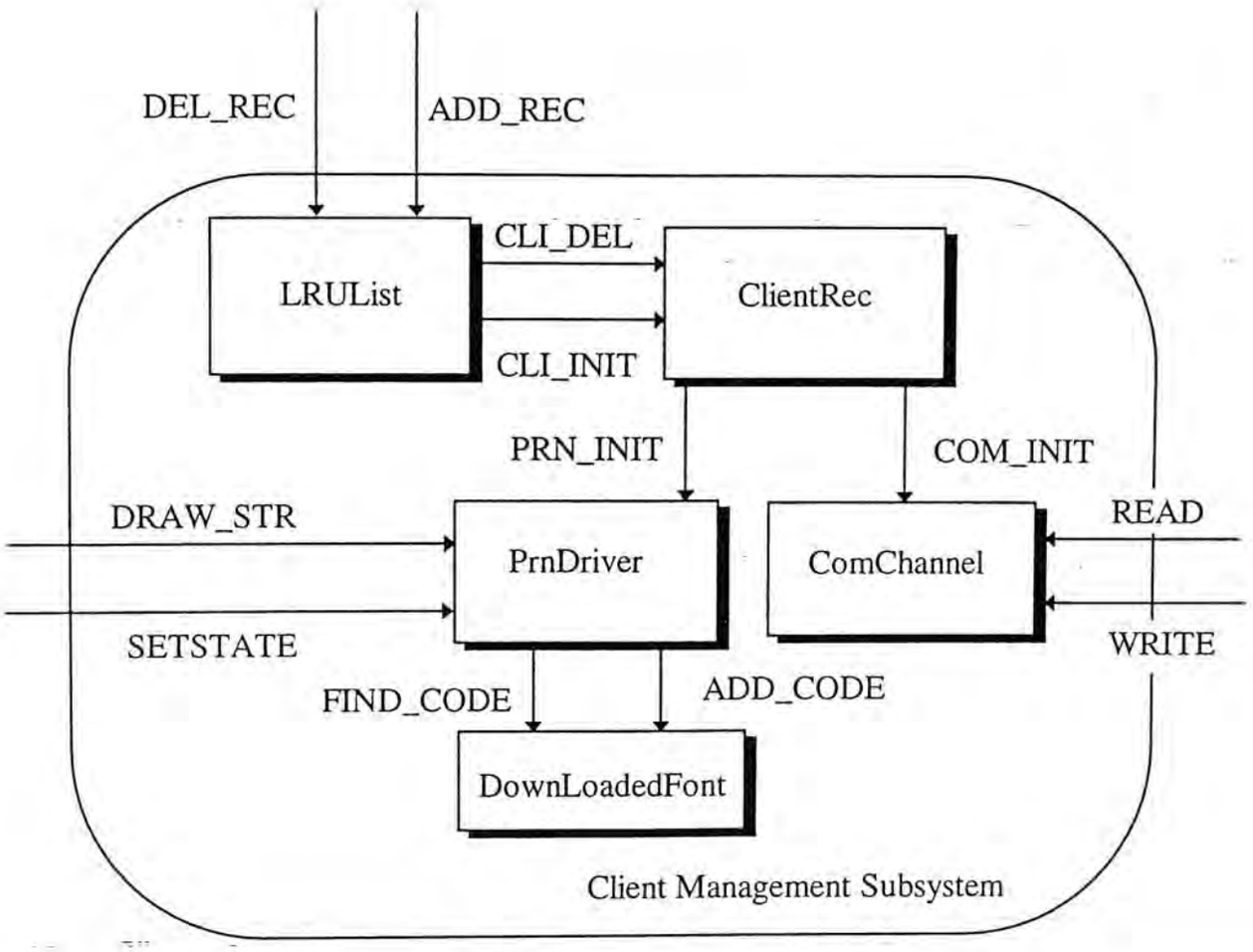


Figure 5-5 Message Diagram of the Client Management Subsystem

The ComChannel class is also included in this subsystem, or more precisely, instances of ComChannel class are included in the Client Management Subsystem. As has been mentioned already, each print client has a dedicated channel for sending requests to the Printer Server. The Printer Server does need to identify which channel is used by which print client. It would be more convenient if an instance of ComChannel is contained in each ClientRec object. This also ensures that each

client can only read the communication channel dedicated to it and will not disturb other clients.

The ComChannel responds to three messages, namely COM_INIT, READ and WRITE. The ComChannel actually does nothing when it receives the COM_INIT message, except to create a new instance of itself. The COM_INIT message is given in the Figure 5-5 just to show that it is the ClientRec which initiates the instantiation of ComChannel.

The ComChannel responds to the READ and WRITE by reading from and writing to the dedicated channel. As already mentioned, the ComChannel provides low-level services only, it does not define any message boundaries for the information it reads and writes. In current design, BSD socket is being used[37]. If a new communication mechanism is employed, only this class needed to be changed while the user of this class is not affected.

One may note that the behavior of the ComChannel class in this subsystem is a little different from what is described in Section 5.2.1. In fact, the ComChannel class defined here does not respond to the ACCEPT message and it does nothing on receiving the COM_INIT message. To cope with this discrepancy in behavior, the ComChannel class described in Section 5.2.1 is redefined to be a subclass of the ComChannel class. This new subclass will be called MainChannel class from here on, to differentiate from its superclass ComChannel.

The MainChannel class inherits all methods from ComChannel class. However, it has its own response to the COM_INIT message. It also has an additional method for the ACCEPT message. The class hierarchy is shown in Figure 5-6.

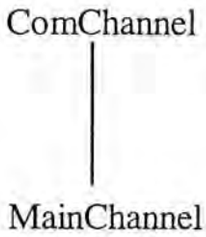


Figure 5-6 Class hierarchy of the ComChannel

In order to shield application programs from tackling device-specified printing issues, the concept of device driver is employed. With the isolation of application programs from device-dependent operations, application programs can run smoothly across systems with different hardware configurations. The design objective of the printer driver is to achieve such isolation.

The printer driver class PrnDriver provides a set of operations for client applications. Whenever a client application issues a printer-related request, the printer driver is invoked. According to the request type, the PrnDriver class performs the corresponding operations for the client. As the PrnDriver class always work on behalf of a specific client, an instance of PrnDriver class is contained in the ClientRec object. Each time a print client wants to do some printer operations, messages are sent to that instance of PrnDriver class.

The PrnDriver class responds to several messages namely the DRAW_STR, GETSTATE, SETSTATE, ESCAPE, EJECTPAGE. Not all the messages are shown in Figure 5-5 in order to save space and to keep the diagram less complex. The responses of the PrnDriver to these are usually very simple, they are listed in Table 5-1.

MESSAGE	RESPONSE
DRAW_STR	Draw a string to the page using the specified font
GETSTATE	Return the current state such as paper orientation, number of copy etc..
SETSTATE	Set the state of the Printer according to the value specified
ESCAPE	Perform printer escape, which may be starting a new document, or closing the current document and resetting the printer
EJECTPAGE	Eject the current page

Table 5-1 Messages responded by the PrnDriver class

The DRAW_STR message deserves more discussion. For double-byte fonts, character bitmap cannot be downloaded directly to the printer, and some mapping must be made to transform a double-byte font into several single-byte fonts. The DownLoadedFont class is designed mainly for this purpose. For single byte fonts, the DownLoadedFont class does nothing special but records the state of the downloaded font such as which character has been downloaded and the font identity number of

this font in the printer etc.. The DownLoadedFont responds to two messages, namely FIND_CODE and ADD_CODE. The FIND_CODE informs the class to check whether the specified code is stored, and return a truth value. The ADD_CODE message tells the class to update its record to include the code specified in the ADD_CODE message.

A printer driver is needed for each printer attached to the printer server. As applications using the printer server should not be machine-dependent, most machine specified operation are picked up by the printer driver. Every printer driver should at least provide the minimal set of operations for client applications. These operations include producing output in the format of the printer, returning printer state to the client applications and setting the printer state in according to client's request etc.. Appendix A lists the complete set of program interface that must be supported by a printer driver in this experimental printer server.

Instead of sending its output directly to the printer, the printer driver generates an output file and sends the file to the UNIX spooler for queuing. As each client should only generate output to its own output file, an instance of printer driver is contained in each client record. That instance of printer driver should generate output to the client's output file only.

When a new printer is to be added to the printer server, a new printer driver must be provided. All printer drivers must conform to the program interface as defined in Appendix A.

The ClientRec class contains all information of the related print client, including the ComChannel and the PrnDriver objects. The ClientRec object itself does not respond to many messages, only the CLI_INIT and the CLI_DEL messages.

Figure 5-5 shows the message passing among different objects in the Client Management Subsystem, a more dynamic, run-time snap shot is shown in Figure 5-7.

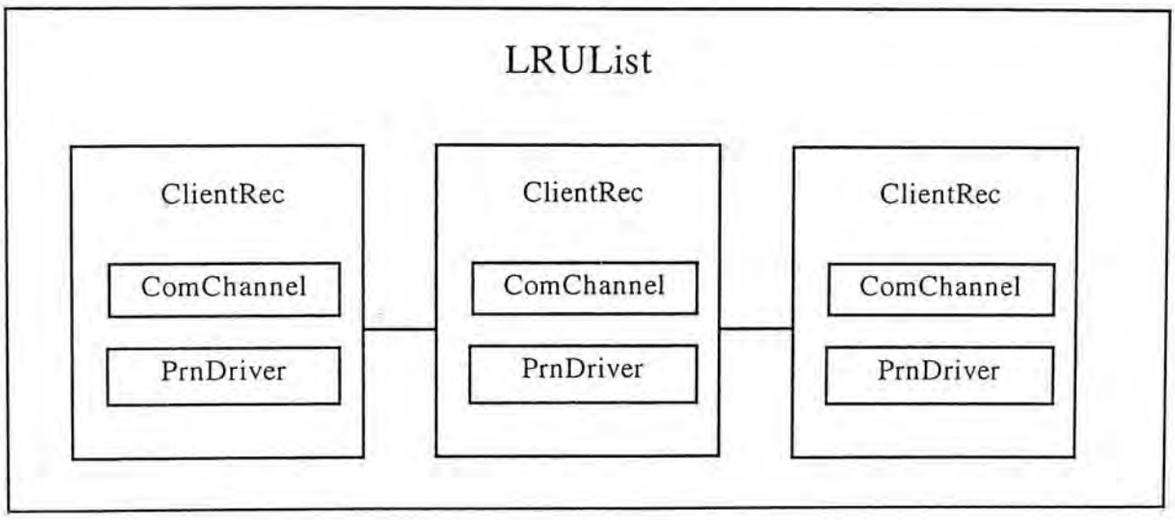


Figure 5-7 Structure of Client Management Subsystem

5.2.3. Request Handling Subsystem

This subsystem includes the reqHeader class, the replyGeneric class and all their subclasses. At the center of the subsystem is the reqHeader class. It responds to four messages, namely READ_REQ, PROCESS_REQ, CREATE_REQ and the WRITE_REQ message. The WRITE_REQ request is only used in the Printer Server Library, the Printer Server itself never generates any WRITE_REQ.

Every time the dispatcher discovers that a channel is ready for reading, it invokes the reqHeader object by sending it a CREATE_REQ message together with the ClientRec object which posses this channel. On receiving the CREATE_REQ message, the reqHeader object reads the first several bytes, called the request header of the request, from the channel using the ComChannel object contained in the ClientRec. After reading the request header, the reqHeader object determines the actual type of the request from the information stored in the request header. A new instance of subclass of reqHeader will be created according to the type of the request. For example, if it is a SetFont request, an instance of SetFont_Req will be instantiated. A PROCESS_REQ message will be passed to this newly created object, together with the ClientRec and the request is processed by the object on behalf of the client indicated in the ClientRec object. The request handler uses the PrnDriver object contained in the ClientRec object to generate output.

When the request handler finishes processing the request, it instantiates an instance of replyGeneric or its subclass, depending on the type of the request, and sends it a SEND_REPLY message together with the ClientRec instance. The replyGeneric or its subclass then writes back a reply for the requesting client, using the channel provided in the ClientRec. The flow of messages is shown in Figure 5-8.

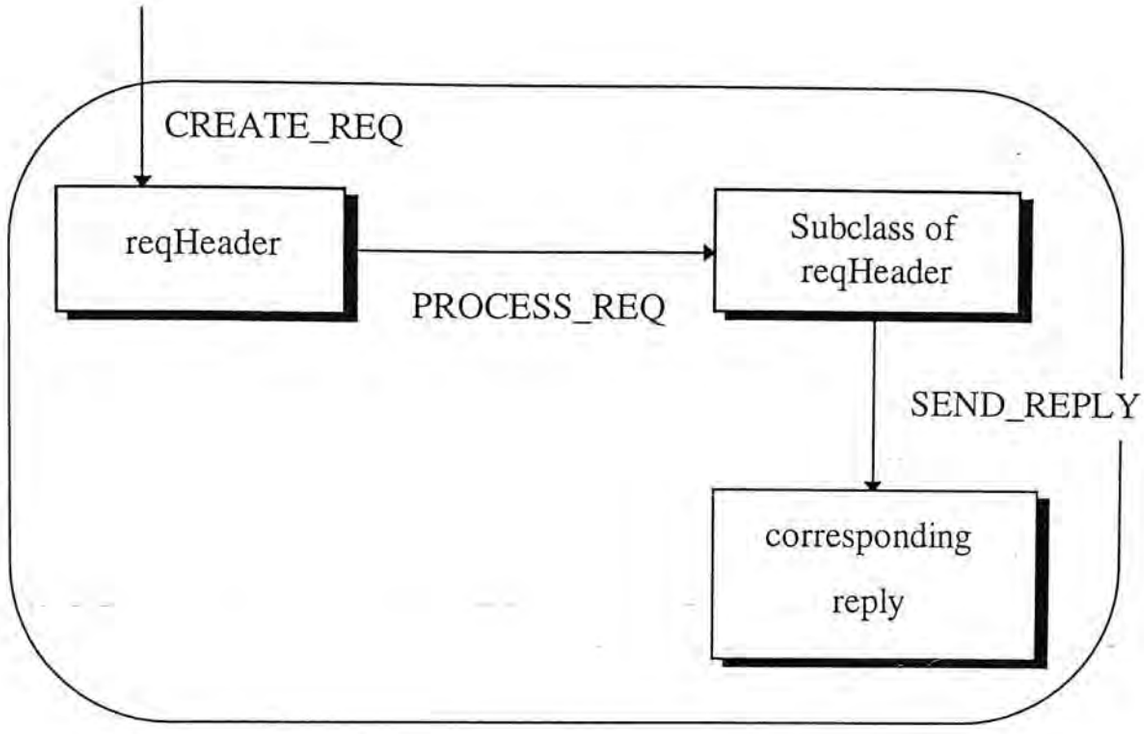


Figure 5-8 Message diagram of the Request Handling Subsystem

In the current design, client and server are synchronous[39], i.e., the client application should wait for the server reply before it can send another request. This synchronization is enacted by the Printer Server Library, and client applications may not even notice it.

5.2.4. Font Managing Subsystem

The Font Managing Subsystem is a relatively independent module in the system. It is not as integrated as the other modules. Basically, the Font Managing Subsystem controls the font access of the whole system, and all other modules should request through the Font Management Subsystem only.

The Font Management Subsystem includes the FnCache, the PrnFont, the FnServer and the CacheStruct classes. The FnCache is the interface for other modules to access font data. It shields all font related details from the other modules.

The FnCache class responds to quite a lot of messages, namely FNC_INIT, FNC_OPEN_FONT, FNC_CLOSE_FONT, FNC_LIST_FONT, FNC_GET_BITMAP, FNC_GET_EXTENT, and FNC_GET_INFO. The responses of the FnCache to these messages are summarized in Table 5-2.

MESSAGE	RESPONSE
FNC_INIT	Establishes connections with remote font servers
FNC_OPEN_FONT	Opens a font and returns the font id
FNC_CLOSE_FONT	Closes a specified font
FNC_LIST_FONT	Lists the available fonts
FNC_GET_BITMAP	Returns bitmaps of the specified characters
FNC_GET_EXTENT	Returns metric of the specified characters
FNC_GET_INFO	Returns information of the specified font

Table 5-2 Messages responded by the FnCache class

As has been mentioned in Section 4.4 , the Printer Server cannot generate any font data. Whenever the Font Managing Subsystem receives a font request, it send a font request to the font servers. The font server will generate the required bitmap for the Printer Server. The Printer Server should be able to connect at least one font

server in order to guarantee the availability of font data. This connection must be established when the Printer Server starts.

At the time the Printer Server starts up, the Server Control Subsystem sends a FNC_INIT message to the FnCache object together with a list of known font servers. On receiving the FNC_INIT message, the FnCache tries to establish connections with these remote font servers. If none of the font servers can be connected, the FnCache will return an error message to the Server Control Subsystem which will in turn terminate the Printer Server with a proper error message to the user.

For each connected font server, the FnCache creates an instance of FnServer for it. The FnCache manages the set of connected font server into a list of FnServer instances. The FnServer class is where the FnCache, i.e., the Printer Server obtains font data. The messages listened to by the FnServer are similar to those of FnCache class, as shown in Table 5-3.

Whenever a FNC_OPEN_FONT message is received by the FnCache, it first checks whether the font is already opened or not. If it is a new font, the FnCache sends the FS_CHECK_NAME message to the list of FnServer in turn until any one of them acknowledges that it has the requested font or the end of list is reached. If the FnCache class receives a positive response from any of the FnServer objects, it finds a new entry in the font cache table, creates a new PrnFont object and attaches this instance of the PrnFont object to the table. The index of this PrnFont instance in the table is returned to the print client, which uses this number for subsequent requests of this requested font.

MESSAGE	RESPONSE
FS_CONNECT	Connects remote font server
FS_DISCONNECT	Disconnects from remote font servers
FS_OPEN_FONT	Requests remote font server to open font
FS_CLOSE_FONT	Requests remote font server to close font
FS_QUERY_BITMAP	Queries remote font server for bitmap
FS_QUERY_EXTENT	Queries remote font server for font metric
FS_LIST_FONT	Requests remote font server to list fonts
FS_CHECK_FONT	Checks whether a font is available from the font server
FS_QUERY_INFO	Queries remote font server for information of a font

Table 5-3 Messages responded by FnServer class

The PrnFont class contains attributes of each font. It also contains an instance of CacheStruct class which performs the actual data searching and bitmap replacement operations. The PrnFont should contain the following font attributes:

- Font full name - full name of the font as described by XLFD[9]
- Typeface name
- Font size in tenth of a point
- Font bitmap size in pixel
- Font Pitch - whether the font is mono-spaced or proportional-spaced

- Posture - whether it is upright or italic
- Stroke Weight of Font - can be normal, medium or bold
- Character Set - contains the character encoding of the font

When a new instance of PrnFont is created, these entries are filled by the FnCache. Since the font name supplied by the client application may not contain all the above information, the FnCache should calculate the unknown values from the available information in the font name. These font attributes are used by the mapping function(which will be described later) in determining whether two fonts are the same or not.

Besides being used in font matching, these font attributes are also important for the Printer Driver. When the Printer Driver downloads a new soft font to the printer, it also describes the font being downloaded to the printer. These attributes are used to describe the downloaded font. This description is important to the printer as some printers allow applications to select a font by simply specifying one or more attributes of that font. The printer thus needs these attributes to enforce the selection.

The PrnFont also contains a reference count. Whenever an existing PrnFont is chosen by the mapping function, the reference count is increased by one. When this font is closed by a client, the reference count is decreased by one. If the reference count drops to zero, that instance of PrnFont is deleted. The messages listened to by the PrnFont class are summarized in Table 5-4.

MESSAGE	RESPONSE
PRN_INIT	Instantiates a CacheStruct instance
PRN_GET_BITMAP	Returns characters bitmap
PRN_GET_EXTENT	Returns metric information

Table 5-4 Messages listened to by PrnFont class

Whenever the FnCache is requested to open a font, the mapping function is invoked. The mapping function is responsible for determining whether the requested font is already opened or not. If the font that matches the mapping criteria is found in the available instances of PrnFont, the index of this PrnFont is returned. Otherwise, FnCache creates a new PrnFont for the requested font.

The mapping function uses the following two criteria to match an incoming font:

1. If the Font Full Name of the incoming font is **exactly** the **same** as that of any one instance of the PrnFonts, the mapping function uses this instance.
2. If the Typeface name, Font Pitch, Posture, Stroke Weight and Character Set of the incoming font match with an instance of the PrnFont, the mapping function compares the bitmap size (in pixel) of two fonts. If they are equal, the mapping function returns that instance of PrnFont, otherwise it continues the matching with the unchecked instances.

The set of all CacheStruct instances constitute the font cache of the Printer Server. To increase flexibility, the CacheStruct is separated as a single object from

the PrnFont so that when the caching strategy is changed, only this part of the codes needs to be changed. The CacheStruct class determines the caching policy used. When a new instance of PrnFont is instantiated, it may, depending on the nature of the font being used, select a CacheStruct that best suits the font[10][20].

The FnCache is responsible for limiting the amount of memory used by each CacheStruct object. When the CacheStruct object is instantiated, the FnCache sends it the maximum amount of memory it may use (MaxMemory) and the maximum number of characters (MaxChars) it may hold in the cache to the CacheStruct. The actual maximum amount of memory that can be used by the CacheStruct is defined in Equation 5-1.

$$MaxCacheMemory = \min(MaxMemory, MaxChars \times GlyphBitmapSize)$$

Equation 5-1 Cache Memory

Basically, the Least Recently Used (LRU) replacement algorithm is employed in the cache. The LRU is so generic that it can be applied to any fonts, including both English and Chinese fonts. However, any other caching strategy can be implemented and adapted from the CacheStruct so that it can take into consideration the special characteristic of a language to achieve better performance[20].

The CacheStruct class responds to two messages, namely SEARCH_CODE and UPDATE_CODE. The SEARCH_CODE informs the CacheStruct class to search the specified character code in the CacheStruct and return the found bitmap.

The UPDATE_CODE tells the CacheStruct class to update the content of the cache, using the predefined algorithm, to include the specified characters bitmap.

Whenever a FNC_GET_BITMAP message is received, the FnCache sends a PRN_GET_BITMAP message to the requested font. The instance of PrnFont then sends a SEARCH_CODE message to the CacheStruct instance contained in it to do the actual bitmap searching. If the requested bitmap data are found, the bitmap will be sent back to the clients immediately. If that bitmap is not available, the PrnFont sends a font request to the font server to obtain font data. By making all font requests to be handled by the FnCache, font data can be shared among all clients. A partial message diagram of the Font Management Subsystem is shown in Figure 5-9.

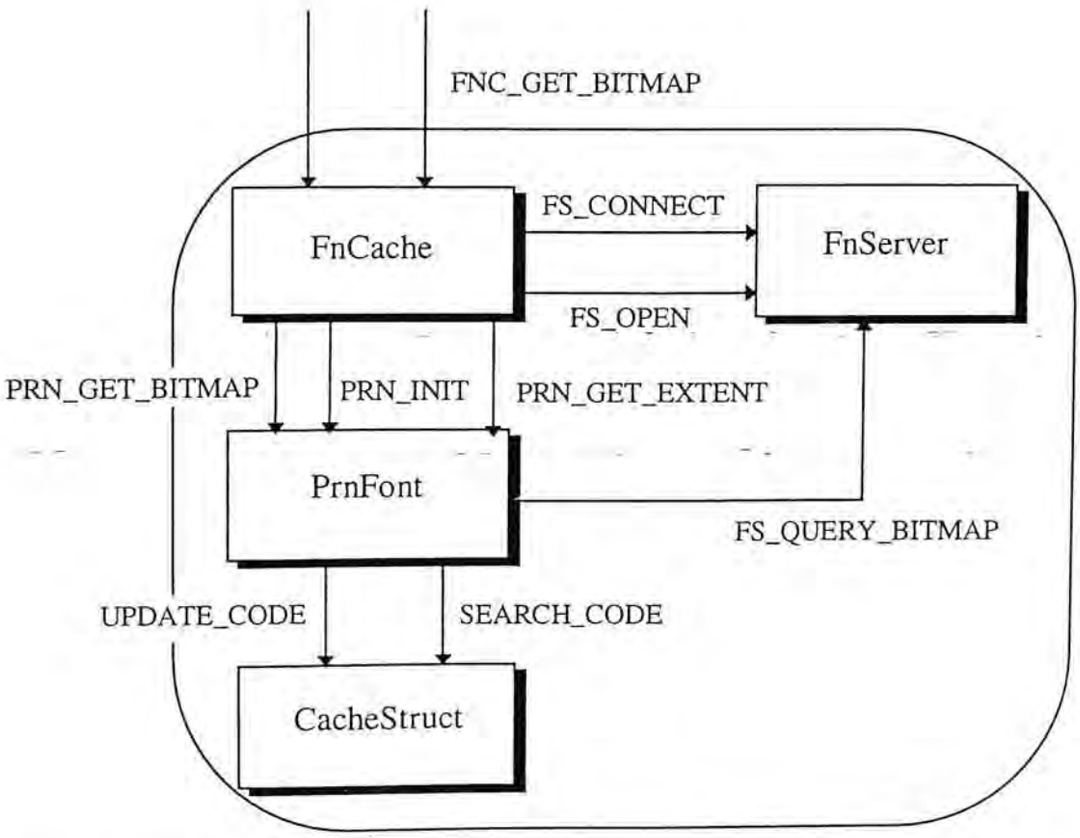


Figure 5-9 Message diagram of the Font Management Subsystem

6. Sample Printer Driver

A printer driver is responsible for managing all printer output for the Printer Server. It provides a set of functions that the client application uses to generate printer output. These printer driver functions translate device-independent printing commands into device-dependent printer primitives of a specific printer. A printer driver must be provided for each printer attached to the Printer Server.

In the implementation of the experimental Printer Server, a printer driver for HP LaserJet III printer is provided. Besides supporting the basic functionality's of PrnDriver class described in Appendix A, this sample printer driver also implements a new soft font handling algorithm which could speed up Chinese characters printing.

6.1. Printer Control Languages

Printer Control Language is the type of programming language used and understood by a specified printer. Different printers may have their own printer control languages. For example, PostScript is the printer control language of Apple LaserWriter printer while HP laser printers use PCL as their control language. As our sample printer driver is designed for HP laser printers, the following discussion is restricted to the PCL language.

From a glance at the PCL language quick reference guide[15], one may notice that PCL is relatively simpler and more primitive as compared with other high level languages such as C or Pascal. In fact, PCL provides no command for decision

making, loop or subroutine branching. In stead, PCL supplies a set of commands for setting and resetting the states of the printer. For example, you may issue a PCL command to set the primary font used by the printer to font number 3. As long as you do not reset the primary font used, the printer will always refer to font number 3 for its primary font.

The set of all these states of the printer constitutes the environment or configuration of the printer. Whenever the printer executes a command, it refers to the environment for the information it needs. The default environment is set when the printer first starts or resets. The default values will not change until you explicitly issue a PCL command to alter them.

Similar to other programming languages, PCL has its own grammar. Basically, all PCL commands must be started with a Escape Character *esc* (ASCII code 27). So PCL commands are also called escape sequences. The *esc* character is used to inform the printer that the coming characters are part of a PCL command, not normal text data. Knowing that the characters constitute a command, the printer will not print it out. Instead, the command will be executed by the printer.

6.1.1. Structure of PCL Command

Generally, the PCL language command consists of five components:

- Escape character *esc*
- A zero or one character **Parameterized character**
- A one character **Group character**

- A **command argument**
- A one-character **Terminating character**

If the command requires additional data, these data will follow immediately after the terminating character.

The **Parameterized character** is used to tell the printer that the following command includes a parameter. It informs the printer to parse through the command before executing the command. Note that not all commands are parameterized. If the command is not parameterized, the parameterized character is omitted. Furthermore, more than one character code is used in representing parameterized character. PCL allocates ASCII code range 33 to 47 (i.e., from "!" to "/") to be used as parameterized characters. Which one of character in the range is used depends on the group to which the command belongs to.

The **Group character** indicates the type of the command. This character distinguishes the different types of PCL commands. For example, the group character "&" indicates page format commands while "*" indicates raster graphics commands. Group characters lie in the range of ASCII characters 39 to 126 (i.e., from "' " to "~").

The **Terminating character** can be uppercase or lowercase ASCII letter. It specifies to which parameter does the command argument apply to. If the character

is lowercase, it indicates that the command will be continue. Otherwise, the command is concluded and will be followed by command data or a new command.

The **command argument** field contains numerical parameters required by the command. It can be any decimal integer. Sign indicator "+" and "-" can also be placed in front of the integer value[2].

6.1.2. PCL Command Example

For instance, in the command

esc *p 400 X

character "*" is the parameterized character, character "p" is the group character, value 400 is the command argument and character "X" is the terminating character. This command instructs the printer to move the cursor to 400 pixels right of current position.

One may notice that the **Parameterized character** and **Group character** do overlap in the range 39 - 47. This implies that some characters (those which fall into the overlapped region) can be used alone to indicate parameterized command group while others need two characters (one for parameterized character and one for group character).

6.2. Printer Font Resources

Among the various features of a printer, the one of most interest to us is its fonts handling ability. The HP LaserJet classifies the fonts it uses according to the font sources. A font source refers to the location in which the font is stored. In HP LaserJet, fonts can be stored in three different locations, namely permanent internal ROM, external font cartridges and internal RAM.

The permanent internal ROM contains the resident fonts (or built-in fonts) of the printer. These fonts are always available whenever the printer is powered on. They do not consume any internal printer memory. Resident fonts cannot be deleted by any PCL command.

External font cartridges contain permanent ROM chips which hold bitmap data of several fonts. Cartridge fonts are available whenever the appropriate font cartridge is plugged in. Cartridge fonts cannot be deleted by any PCL command. Since font cartridges can be plugged in and out of the printer, users can choose the fonts they really need and place the appropriate cartridge into the printer. Similar to permanent ROM fonts, cartridge fonts do not consume memory.

The internal memory of the printer can store fonts downloaded from the computer. These fonts are called soft fonts as they are stored in RAM and not "burned" into permanent ROM. Soft fonts can be downloaded to the printer when they are actually needed and can be deleted when no longer required. However, soft

fonts would consume printer memory that is used to process the document. The more soft fonts are used, the more memory is consumed and the less remains.

Theoretically, PCL allows a maximum of 32,767 soft fonts to be downloaded. However, the number of soft fonts that can be stored in the printer memory is bound by the memory available.

PCL further classifies soft fonts into permanent soft fonts and temporary soft fonts. Permanent soft fonts are stored in the printer memory until the printer is turned off or the font is explicitly deleted by PCL commands. They will not be affected by printer resets. Temporary soft fonts, on the other hand, will be deleted when the printer resets. They will also be overwritten by other soft fonts and graphics when memory is scarce. So, if an overwritten soft fonts is used again, it must be re-downloaded to the printer. Permanent soft fonts are usually used when the fonts are used very intensively and if the font sizes are not very large. Otherwise, temporary soft fonts are used.

6.3. Traditional Font Handling Methods in a Printer Driver

Basically, there are two main methods to draw a string to a page. In the first place, we may send the character bitmaps to the printer directly and instructs the printer to visualize the character glyphs as graphics images. This method is conceptually simpler and requires no additional effort. Furthermore, the bitmap data sent to the printer do not consume much memory since it is stored only temporarily. The disadvantage of this method is that for each time a character is to be drawn, the

bitmap data must be send again. The printer cannot retrieve the bitmap data of that character since it is not stored in the printer memory.

The second method makes use of the soft font handling capabilities described in Section 6.2. To draw a character, the printer driver creates a new soft font (in the format required by the printer) and downloads the font to the printer. Each time a character of that soft font is printed, the printer retrieves the bitmap data from the printer memory and draws the character glyph image on paper. This method requires additional effort to create the soft font and extra printer memory to store the bitmap data and information (such as font identity number and font name) of the downloaded font. However, this method is more effective when the repetition rates of the characters in the soft font are high.

Although the second method seems to be superior to the first one, its application is subjected to limitations. Firstly, some printer control languages (such as PCL) restrict the number of characters in a downloaded soft font to 255. [5] For double-byte fonts (such as Chinese fonts), this maximum is far too small. Secondly, this method assumes the target printer to have significant printer memory (usually several hundred kilobytes) to hold downloaded font data, this requirement may not be met by most dot-matrix and ink-ject printers. Thirdly, since not all types of printer are equipped with soft font handling capabilities, drawing characters as graphics may be a better choice for printers that cannot support soft font downloading.

[5] PCL allows a maximum of 245 characters in a soft font. All characters are printable except ASCII values 0, 7-15 and 27.

In the PCL printer driver of MS Windows 3.1[27], one-byte fonts are handled using the second method while double-byte (such as Chinese) fonts are drawn as graphics images. It is natural to have double-byte fonts to be printed as graphics as PCL cannot support soft font with more than 245 characters. However, as has been mentioned already, printing double-byte fonts as graphics image required bitmaps to be downloaded every time they are needed. This slows down the whole printing process if the repetition rate of the characters is high. Unfortunately, Chinese characters do have an uneven usage frequency[48]. Some characters have very high usage frequencies while others are very low. Printing Chinese characters as graphics does not take advantage of this high repetition rate.

6.4. Soft Font Creation in PCL Printer

In this section, the steps involved in the creation of a soft font in PCL printer is presented. PCL can handle two types of soft fonts, namely bitmap font and scalable font (see Section 1.2). However, since the font server is bitmap-oriented, only bitmap data is passed to the Printer Server and printer drivers. Hence the sample printer driver always downloads soft fonts as bitmap fonts. In the following discussion, the term soft font always refers to bitmap font.

Creating a bitmap soft font in PCL printer involves the following steps:

1. Assign font ID number to the new soft font
2. Create and downloads a **Font Descriptor**
3. Assign character code to the character being downloaded
4. Create a **Character Descriptor**

5. Download the character bitmap together with a character descriptor

Note that steps three to five must be repeated for each character being downloaded.

6.4.1. Font ID number

Every soft font of a printer should have a unique font ID number. This number is used to select the soft font when it is needed. This number is assigned and kept track of by the printer driver. If a font ID number is used again, the old soft font assigned with this font ID number will be deleted automatically by the printer.

6.4.2. Font Descriptor

The font ID number is followed by a collection of parameters depicting the font being downloaded. This collection is called the **Font Descriptor**. Entries in the Font Descriptor are used to tell the printer what kind of font is being downloaded. They are also used when an application selects a font by its attributes rather than its font ID number. In this circumstance, the printer matches the specified attributes against the available fonts and uses the closely matched one.

PCL defines a 64-byte font descriptor for bitmap soft fonts. The printer driver should specify all entries of the font descriptor to the printer, with the reserved entries set to 0. The format and entries of the font descriptor are shown in Table 6-1.

Byte	Most Significant Byte (Most Significant Bit First)	Least Significant Byte (Most Significant Bit First)
0	Descriptor Size	
2	Reserved	Font Type
4	Reserved	
6	Baseline Distance	
8	Cell Width	
10	Cell Height	
12	Orientation	Spacing
14	Symbol Set	
16	Pitch	
18	Height	
20	X Height	
22	Width Type	Style
24	Stroke Weight	Typeface
26	Reserved	Serif Style
28	Reserved	
30	Underline Distance	Underline Height
32	Text Height	
34	Text Width	
36	Reserved	
38	Reserved	
40	Pitch Extended	Height Extended
42	Reserved	
44	Reserved	
46	Reserved	
48-63	Font Name	

Table 6-1 Bitmap Font Descriptor

Note that all two-byte entries of the font descriptor are stored according to Most Significant Byte First convention. If the printer driver runs on a machine with different byte ordering, swapping should be done before the font descriptor is downloaded. Figure 6-1 depicts the meanings of some entries.

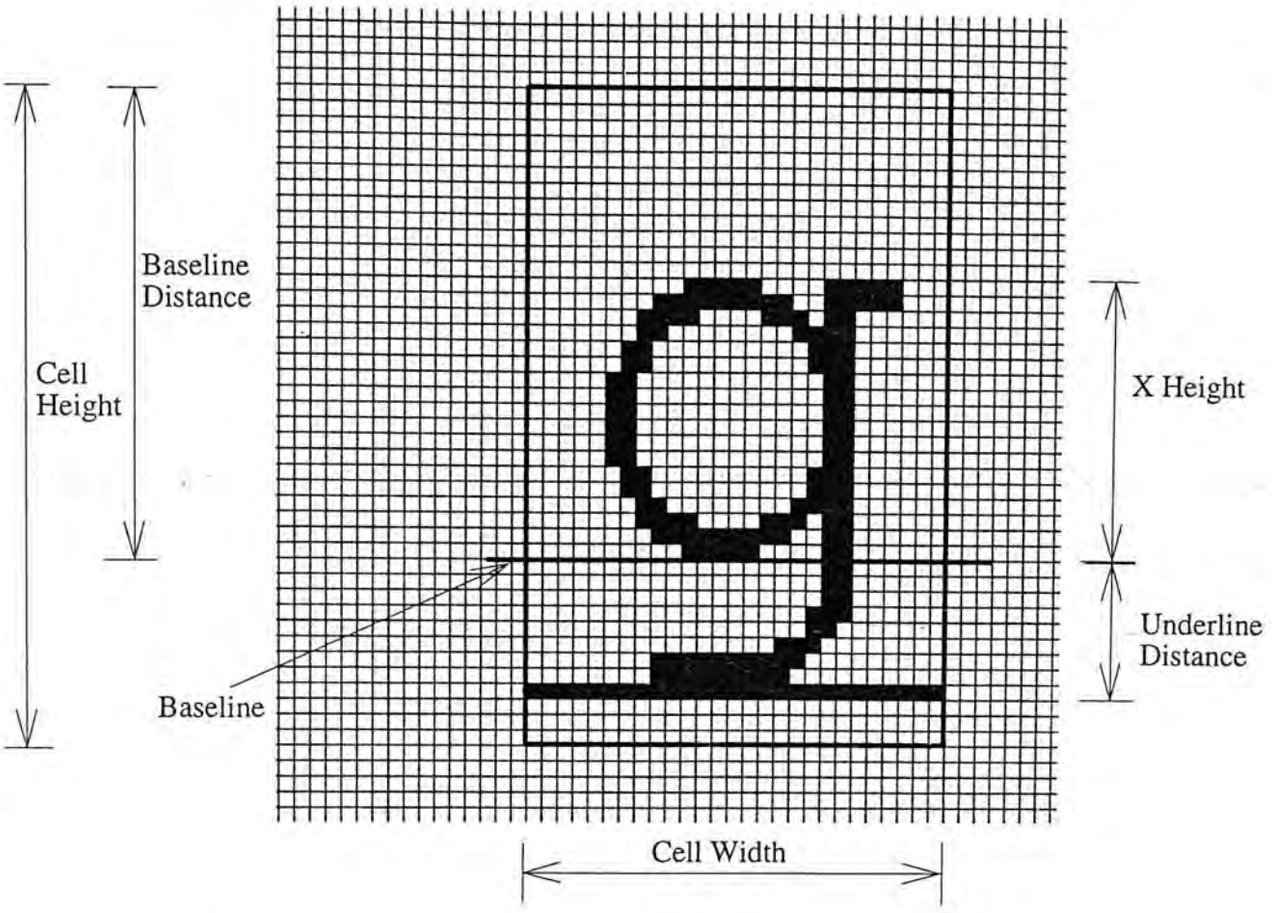


Figure 6-1 Font Descriptor Entries

6.4.3. Character Code

A character code must be specified for each character being downloaded to the printer. This character code must be unique within the selected font. If the same character code is assigned twice within a font, the old character bitmap will be overwritten by the new one.

According to the value in the Font Type field in the Font Descriptor, a maximum of 245 different codes can be assigned to a downloaded character.

Whenever a character is to be printed, the respective character code is sent to the printer. The printer used the font ID number and the character code to identify which character bitmap to print.

6.4.4. Character Descriptor

The next step is to described the character being downloaded to the printer. It is similar to the description of the font except that this time parameters of individual character are sent.

PCL defines a 16 bytes **Normal Character Descriptor** and a 2 bytes **Continuation Character Descriptor** for character description. The normal character descriptor is used with the first block of data of the character while the continuation character descriptor is used with the subsequent blocks. The format and fields of a normal character descriptor is shown in Table 6-2. The meanings of some fields are depicted in Figure 6-2.

Byte	MSB	LSB
0	Format	Continuation
2	Descriptor Size	Class
4	Orientation	Reserved
6	Left Offset	
8	Top Offset	
10	Character Width	
12	Character Height	
14	Delta X	

Table 6-2 Normal Character Descriptor

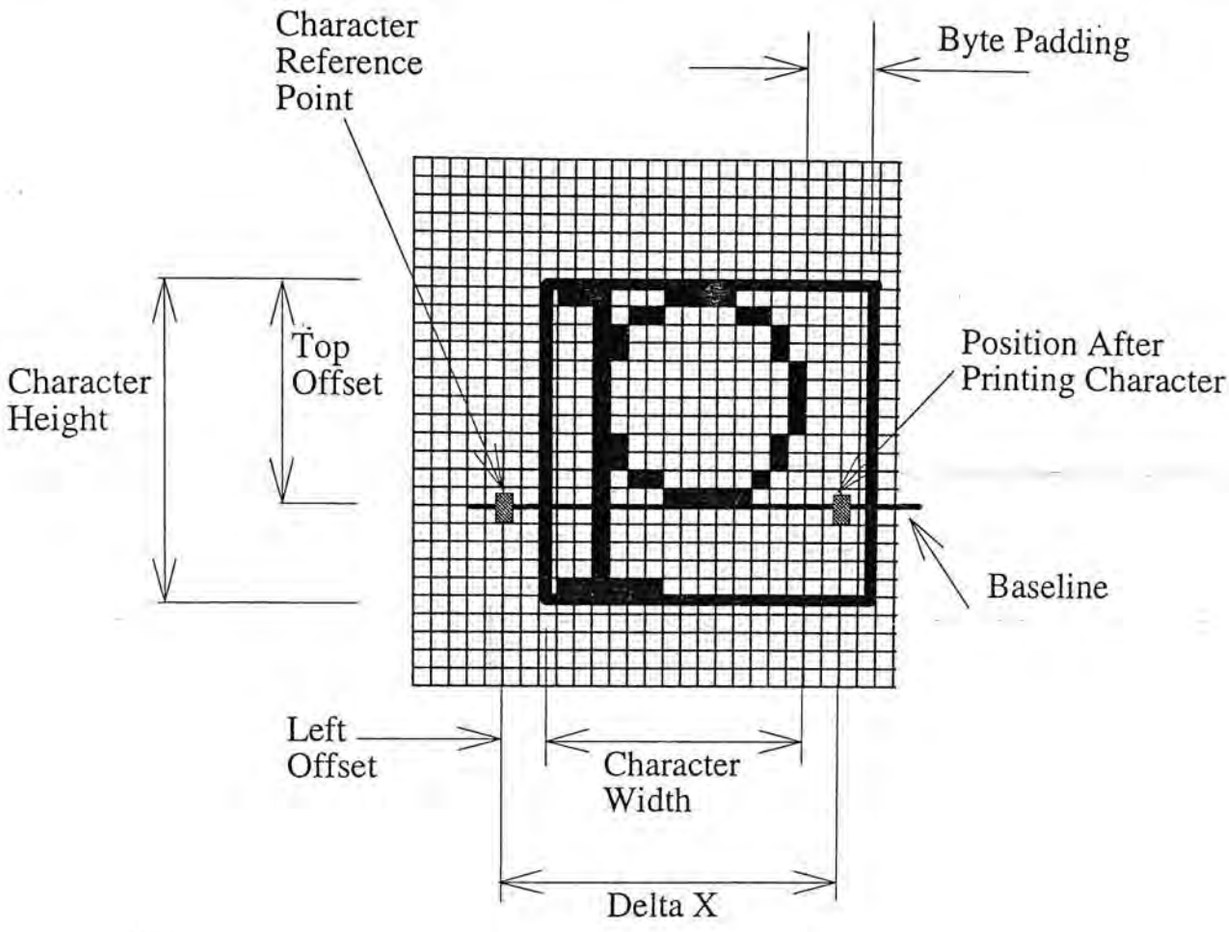


Figure 6-2 Character Descriptor Entries

The Descriptor Size field should be set to 14 for a normal character descriptor. The Continuation field is worth a more detailed explanation. Since PCL limits the size of data transfer in one block to 32767 bytes[6], large characters should be sent in more than one block. The continuation descriptor is used to indicate that the coming data is a continuation of the previous block, not a new block. The format of the continuation descriptor is shown in Table 6-3. Note that for a normal character descriptor, the continuation field is set to 0 while that in the continuation character descriptor is set to 1.

[6] In reference number 2 P. 202, it states that the maximum size of a block is limited 32767 bits, it should be 32767 bytes instead of 32767 bits.

Byte	MSB	LSB
0	Format	Continuation (1)

Table 6-3 Continuation Character Descriptor

6.4.5. Character Bitmap Data

The last step is to sent the character descriptor and the bitmap data to the printer. Before sending the bitmap, the printer driver should inform the printer how many bytes of data are being sent. This size should include the size of the character descriptor. For example, if a bitmap of 64 bytes is being downloaded together with a normal character descriptor, the printer driver should tell the printer that 80 (64+16) bytes are being sent.

The bitmap is sent row by row with the top most row first. The Most Significant Bit of each byte corresponds to the left most pixel of the character. Each row is padded to the nearest byte.

After sending the bitmap data, the process of downloading a character is completed. Each time this character is to be drawn, only the font ID number and character code are needed to inform the printer what bitmap is to be visualized. No bitmap data needs to be sent again.

6.5. New font downloading schemes for double-byte fonts

In this section, two new font downloading schemes for double-byte fonts will be presented. Basically, these new schemes also make use of the soft font creation

method. However, unlike PostScript Level 2 printer, PCL printer cannot handle a soft font with size greater than 245[26]. A Chinese font must be subdivided into several fonts before bitmap data are downloaded to the printer.

6.5.1. Terminology

The following terms are used through out the discussion.

- **driver font id**: the font identity number assigned by the printer driver and used by the application to tell the *printer driver* which font to used

- **printer font id**: the actual font identity number assigned to a font when it is being downloaded to the printer. This id is used to tell the *printer* what font to use. If a font contains more than 245 characters, then it must be divided into several fonts before it can be downloaded to the printer. For each driver font id, there may be more than one printer font id associates with it. It is the printer driver's responsibility to map driver font id to the appropriate printer font id.

- **character internal code**: this is the internal code of the character. This code can be ASCII for English and BIG5 for Chinese, depending on the encoding tables used
- **character print code**: this is the printer representation code of the character. This code is assigned by the printer driver. The application supplies the printer driver with driver font id and character internal code,

which returns a printer font id and the character print code which can be used for actual printing.

Figure 6-3 depicts the mapping defined by the printer driver.

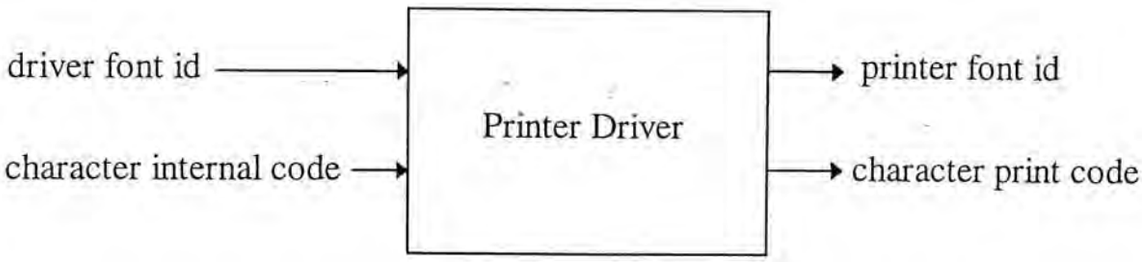


Figure 6-3 Coding Action of the Printer Driver

6.5.2. Underlying Concepts of Algorithm One

Several basic concepts are involved in this font downloading scheme:

- A character bitmap is downloaded only if it is really needed and currently the printer does not have it.
- Double-byte font should be divided into several fonts when it is downloaded to the printer.
- For each bitmap data being downloaded, a **DownLoadFont** table is searched to obtain the printer font id and character print code to be used in the printer.

The **DownLoadFont** table is maintained by the printer driver. It keep track of which character bitmaps have been downloaded to the printer and their respective

font id and character code. The font id is used to select the font when it is needed while the character code tells the printer which character bitmap is to be drawn.

The **DownLoadFont** table is implemented using a linked list. Each node of the list records the status of each downloaded font of the printer, with one node per *printer font*. Entries like driver font id and printer font id are contained in each node which are used to facilitate mapping between two ids. Besides the driver font id, an entry called **minor_font_id** is also stored. It indicates the sequence number of the sub-font of the font with font number driver font id. For example, the first sub-font of the driver font id 1 is 0, the second sub-font of the driver font id 1 is 1 and so on. The driver font id and the minor_font_id together determine the printer font id.

An **Encoding Array** is a also included in each node. This encoding array is used for mapping character internal code to character print code. For each character internal code used, the character internal code is contained in the array. The corresponding character print code is indeed the index of the character internal code in the array. The structure of the DownLoadFont table is shown in Figure 6-4.

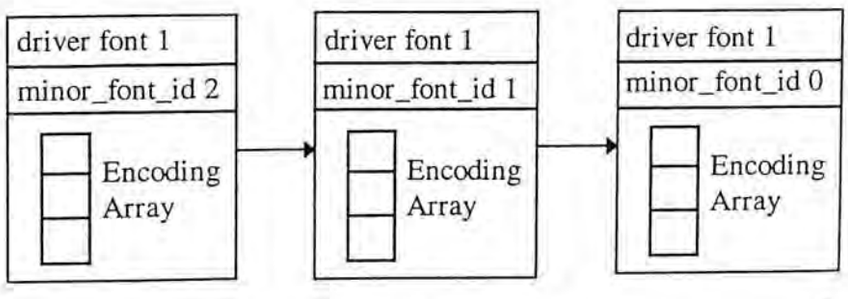


Figure 6-4 Structure of DownLoadFont Table

6.5.3. Algorithm One

Whenever a character is to be drawn, the **DownLoadFont** table is searched. If the respective bitmap has already been downloaded, the printer driver selects the printer font using the font id returned and then instructs the printer to draw the character with the character code returned. If on the other hand, the respective bitmap has not been downloaded, the printer driver checks if the last printer font downloaded has space to accommodate this new character. If it has, the printer driver selects that font and downloads the character bitmap with the unused character code. The printer driver then updates the last printer font record to indicate that one more character is downloaded.

On the other hand, if the last printer font downloaded does not have space to accommodate this character, the printer driver creates a new entry in the DownLoadFont table, creates a new Font Descriptor(see Section 6.4) and sends the character bitmap to the printer. The algorithm is shown in Algorithm 6-1.

```
/* Double bytes fonts downloading algorithm */

for each character in incoming string {
      if (character glyph in printer){
            /* character bitmap already downloaded */
            if ( current_font_id NotEqualTo
                  printer_font_id )
                  set current_font_id to
                        printer_font_id;

            send character_print_code;
            /* send char code directly without sending
                  bitmap again */
      }

      else {
            if (Encoding Array in old_font_entry full OR
                  new font){
                  get printer_font_id;
                  make new font entry;
                  send font descriptor with printer_font_id;
                  set current_font_id to printer_font_id;
            }
            /* old font entry still has room for new
                  character */
            get character_printer_code from Encoding Table;
            create and send character descriptor;
            send character bitmap;
            send character_print_code;
      }
}
```

Algorithm 6-1 Soft Font Downloading Algorithm One

6.5.3.1. Code Mapping

One of the crucial parts to the success of the algorithm is the mapping of driver font id and character internal code to printer font id and character print code. In the abstract sense, the mapping can be viewed as a mathematical function which takes the driver font id and character internal code as arguments and yields the printer font id and character print code.

Let *M(drFID,IntCode)* be the mapping function, where drFID is the driver font id and IntCode is the character internal code.

Since for each pair of arguments (drFID,InCode), a unique result must be obtained (otherwise two different characters will have the same print code in the printer), the function *M* should be injective.

Let prFID be the printer font id and PrnCode be the character print code.

For each pair of argument (drFID,IntCode), the DownLoadFont table is searched. If drFID is found in the table, the node with largest minor_font_id is then searched for an empty entry in the Encoding array to accommodate the IntCode.

If this node has an empty array entry, PrnCode is set to the *INDEX* of this entry in the array and IntCode is put in this entry.

If drFID is not found or the node with largest minor_font_id is full, a new node with driver font id equal to drFID is created. Depending on whether this is the first entry of drFID, minor_font_id of this new node is set to 0 or largest minor_font_id +1. IntCode is then put into the first entry of the Encoding array and PrnCode is set to its index. Since only the node entry with largest minor_font_id needs to be checked, this newly created node is prepended to the DownLoadFont table so that the minor_font_id numbers of the fonts with the same driver font id is arranged in descending order.

Finally, the prFID is calculated using the formula:

prFID= drFID * 256 + minor_font_id

As the maximum value of minor_font_id is 255, the prFID can be viewed as a two digits base 256 number with its most significant digit equal to the drFID and

least significant digit equal to minor_font_id. It is quite obvious that the above formula indeed guarantees that two different drFIDs will not be mapped, in any case, to the same prFID. (Imagine that if two two-digits decimal numbers are equal, then their respective digits must also be equal.)

6.5.3.2. Example

Consider the following example. The content of the DownLoadFont table is as shown in Figure 6-5. A new Chinese character 卓 with driver font id 3 and character internal code A8F4(hex) is being downloaded to the printer. Since the last entry with driver font id 3, in this case minor_font_id 2, is not full, A8F4(hex) is placed in the first unused entry (index number 50) in the Encoding array. After putting character A8F4(hex), the first unused entry is now at index number 51. The DownLoadFont table after putting A8F4(hex) is shown in Figure 6-6.

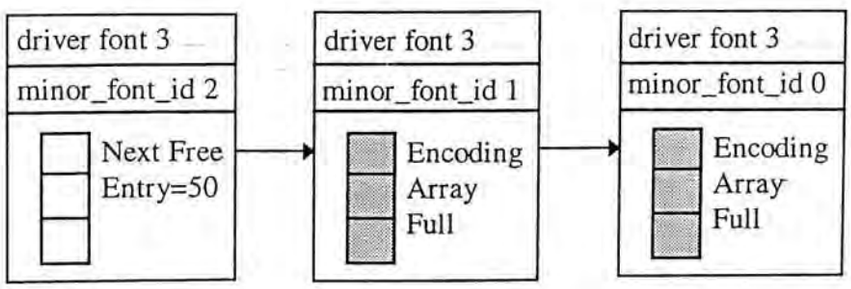


Figure 6-5 Before Adding Character 卓

In this case, the printer font id of the character is 770 ($3 \times 256 + 2$) while its character print code is 50. From now on, these two numbers are used whenever the character A8F4(hex) of driver font id 3 is used.

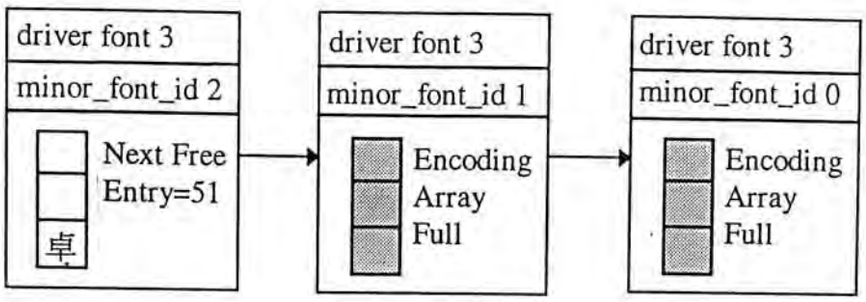


Figure 6-6 After Adding Character 卓

6.5.3.3. Memory Consideration

As the downloaded soft fonts are stored in printer memory, there will be a chance that at the middle of a print job, the memory available is not sufficient to accommodate a new font. In this case, some old fonts should be deleted from the printer memory in order to free space for incoming soft fonts. The printer driver should be responsible for keeping track of the available memory in the printer and deleting some fonts when necessary.

The printer driver maintains the DownLoadFont table using a First-In-First-Out (FIFO) algorithm. Whenever memory is tied, the oldest font is deleted from the printer first and record of this oldest font is removed from the DownLoadFont table. Any later reference to this font entry will fail and bitmap data of characters in this font entry should be downloaded again if they are needed afterward.

The algorithm shown in Algorithm 6-1 is not capable of handling low memory condition. A modified algorithm is now shown in Algorithm 6-2. This new version performs memory checking whenever something is being downloaded to the

printer. It deletes some old fonts when memory available is insufficient for incoming data.

```
/* Version 2 of Soft Font Downloading Algorithm

      Memory Constraint is taking into consideration */


for each character in incoming string {
      if (character glyph in printer){
            /* character bitmap already downloaded */
            if ( current_font_id NotEqualTo
                  printer_font_id )
                  set current_font_id to
                        printer_font_id;

            send character_print_code;
            /* send char code directly without sending
                  bitmap again */
      }

      else {

            if (Encoding Array in old_font_entry full
                        OR new font){
                  get printer_font_id;
                  make new font entry;

                  /* Modified to check memory */

                  if(low memory){
                        delete oldest font;
                        update printer available memory;
                  }
                  send font descriptor with printer_font_id;
                  update printer available memory;
                  set current_font_id to printer_font_id;
            }

            /* old font entry still has room for new
                  character */
            get character_print_code from Encoding Table;
            create character descriptor;

            if(low memory){
                  delete oldest font;
                  update printer available memory;
            }

            send character descriptor;
            send character bitmap;
            send character_print_code;
            update printer available memory record;
      }
}
```

Algorithm 6-2 Version 2 of Algorithm One

6.5.4. Algorithm Two

In Algorithm One, two levels of searching are required. The DownLoadFont table should be searched first for drFID. For each DownLoadFont table entries with the font id equal to drFID, their Encoding Tables must be searched to see if the IntCode is contained in one of them. The searching for IntCode ends only if the IntCode is found or when all the Encoding Array has been searched. It follows that if there are five DownLoadFont entries whose font id is equal to drFID, then in the worse case, five Encoding Tables should be searched.

The main objective of Algorithm Two is to reduce the level of searching to one. As we have seen in Section 6.5.3.1, the code mapping relies on the second level of searching in order to guarantee its injectivity. To eliminate the second level of searching, the mapping algorithm should be changed.

The design of Algorithm Two is inspired by the fact that Big5 code is double bytes. Algorithm Two tries to separate the Big5 code into two parts. Instead of defining the mapping by searching, Algorithm Two constructs a mathematical function which maps the incoming code pair (drFID, IntCode) to the printer code pair (prFID, PrnCode). Similarly to Algorithm One, this function much be injective.

Let V be a function which takes two arguments, the drFID and IntCode. Let First_IntCode and Second_IntCode denote the first and second byte of IntCode respectively.

The function V is defined as:

V(drFID, IntCode)

= (drFID x 256 + First_IntCode, Second_IntCode)

That is,

prFID = drFID x 256 + First_IntCode

PrnCode= Second_IntCode

The next step involved is to prove the injectivity of function V.

Suppose that

V(drFID1, IntCode1) = (prFID1, PrnCode1)

V(drFID2, IntCode2) = (prFID2, PrnCode2)

and prFID1 =prFID2

PrnCode1 =PrnCode2

It follows immediately that

Second_IntCode1=Second_IntCode2

Furthermore, from the definition of prFID and the fact that First_IntCode must be less that 256,

prFID mod 256 = First_IntCode

So,

prFID1 = prFID2

$\Rightarrow$ prFID1 mod 256 = prFID2 mod 256

$\Rightarrow$ First_IntCode1 = First_IntCode2

Therefore,

IntCode1 = IntCode2

Furthermore,

$$\begin{aligned} & prFID1 && = prFID2 \\ \Rightarrow\ & drFID1 \times 256 + First\_IntCode1 && = drFID1 \times 256 + First\_IntCode2 \\ \Rightarrow\ & drFID1 \times 256 && = drFID2 \times 256 \\ \Rightarrow\ & drFID1 && = drFID2 \end{aligned}$$

So, it can be concluded that

$$V(drFID1, IntCode1) = V(drFID2, IntCode2)$$

$$\Rightarrow drFID1 = drFID2 \quad \text{and} \quad IntCode1 = IntCode2$$

With the injective function in hand, the design of Algorithm Two is straight forward. The (drFID, First_IntCode) from the incoming code is searched first. If the font entry is already in the DownLoadFont table, the entry in the respective Encoding Array with index equal to Second_IntCode is test to see if the character bitmap is downloaded or not. If the bitmap have been downloaded, the PrnDriver updates this entry, creates the character descriptor and sends the descriptor to the printer followed by the character bitmap.

If the font entry is not in the DownLoadFont table, the PrnDriver checks whether there is enough room in the printer and deletes some old fonts if necessary. It then creates a new font descriptor and sends it to the printer, followed by the

character descriptor and character bitmap. Algorithm Two is depicted in Algorithm 6-3

The DownLoadFont table is updated according to the LRU algorithm. Every time a font is reference, it is moved to the head of the list. Deletion is always made at the end of the list..

```
/* Algorithm Two */

for each character in incoming string {
      if (character glyph in printer){
            /* character bitmap already downloaded */
            if ( current_font_id NotEqualTo
                  printer_font_id )
                  set current_font_id to
                        printer_font_id;

            send character_print_code;
            /* send char code directly without sending
                  bitmap again */
      }
      else {

            if (new font){
                  calculate printer_font_id;
                  make new font entry;

                  /* Modified to check memory */

                  if(low memory){
                        delete oldest font;
                        update printer available memory;
                  }
                  send font descriptor with printer_font_id;
                  update printer available memory;
                  set current_font_id to printer_font_id;
            }

            /* old font entry found */
            calculate character_print_code;
            create character descriptor;

            if(low memory){
                  delete oldest font;
                  update printer available memory;
            }

            send character descriptor;
            send character bitmap;
            send character_print_code;
            update printer available memory record;
      }
}
```

Algorithm 6-3 Algorithm Two

7. Experiment Results and Discussions

In the following sections, the experimental results of several tests will be presented. Two types of tests are performed. In the first type, the cache performance will be examined. The cache is tested with 279 articles with different numbers of characters. In the second test, the two font downloading algorithms are compared with that of MS Windows. In this test, 90 articles with different sizes are used,

7.1. Cache Test

The performance of the cache is measured with four sets of tests. The number of characters cached in the tests is set to 50, 100, 500 and 1000 respectively. In each test, 279 articles are used. For each article, the number of cache hits and cache miss are recorded and the relative hit ratio is calculated.

All four sets of tests are performed on a SPARC10 station running SunOS 4.1.3. The cache is flushed after a single article is finished. Test results are shown in Figure 7-1, Figure 7-2, Figure 7-3 and Figure 7-4. The average hit ratio is shown in Figure 7-5.

In Figure 7-1 and Figure 7-2, the fluctuation of hit ratio among articles is quite large, while in Figure 7-3 and Figure 7-4 the hit ratio, although it still has fluctuates, does have a tendency to increase when the file size increases. This can be explained by the fact that the hit ratio of the cache not only depends on the size of the input file, but also depends on the arrangement of the characters in the file. The

arrangement of characters within the file is usually called the working set of the file.[40]

When the cache size is small, as in Figure 7-1 and Figure 7-2, a long article with high repetition rate still has a low hit ratio because the cache size is to accommodate the working set of the article. When the cache size is large enough, as in Figure 7-3 and Figure 7-4, there is enough room to accommodate the working set of most articles, so the hit ratio is higher.

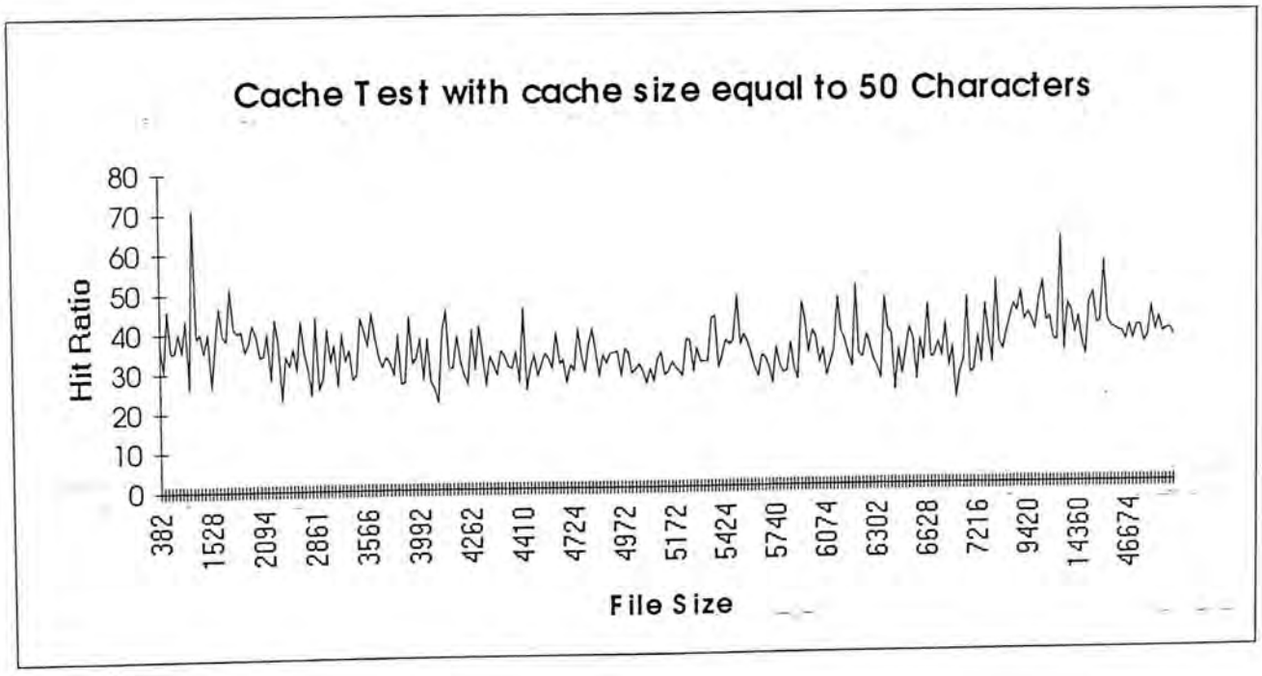


Figure 7-1 Cache Test with a 50-characters cache

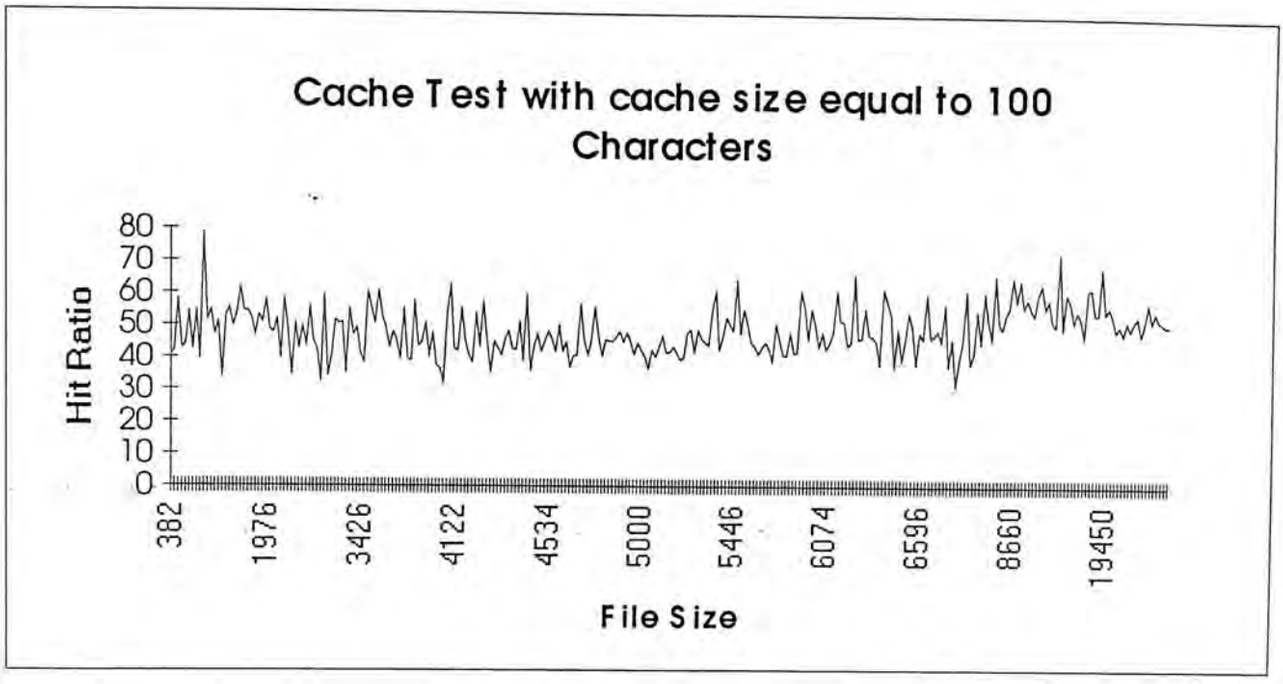


Figure 7-2 Cache test with a 100-characters cache

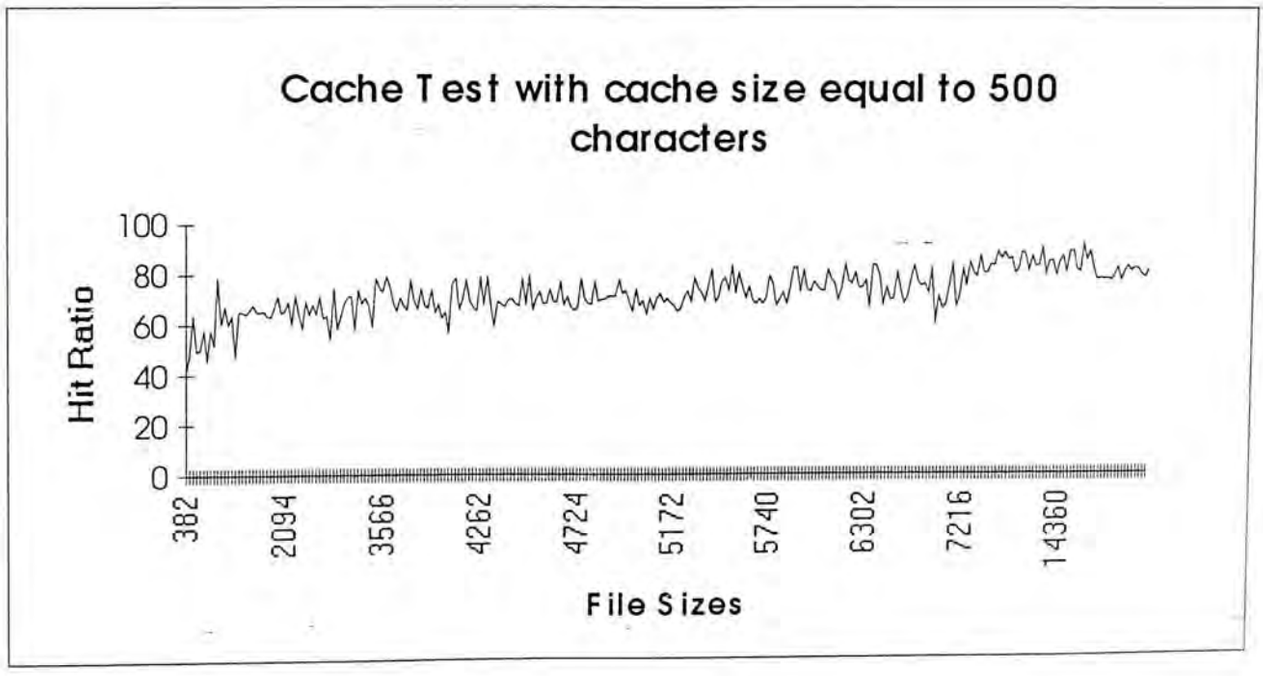


Figure 7-3 Cache Test with a 500-characters cache

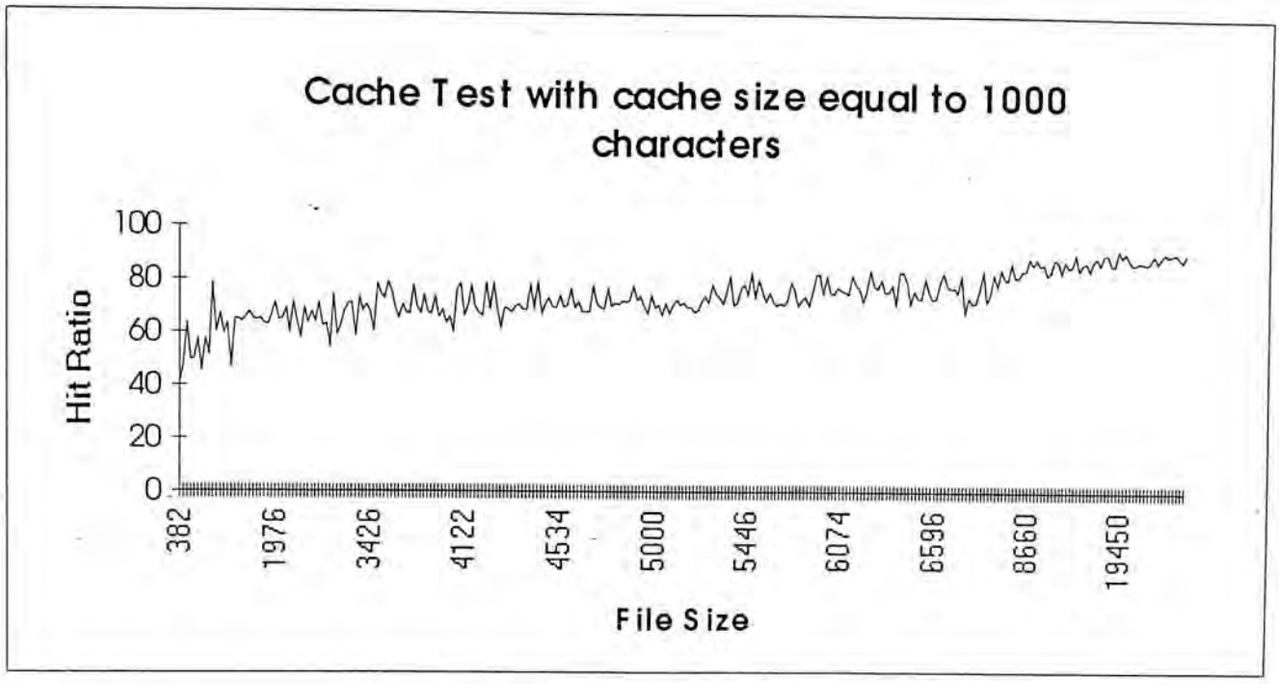


Figure 7-4 Cache test with a 1000-characters cache

As can be seen from Figure 7-5, the average hit ratio increases when the number of characters cached increases. However, the curve shown in Figure 7-5 does not have a fixed slope. Instead, the slope of the curve decreases as the cache size increases. In order words, the marginally cache improvement diminishes with the size of cache.

From Figure 7-5, a 500-characters cache gives a hit ratio of about 70%. This cache size is employed in the Printer Server since further increase in cache size does not improve the hit ratio much.

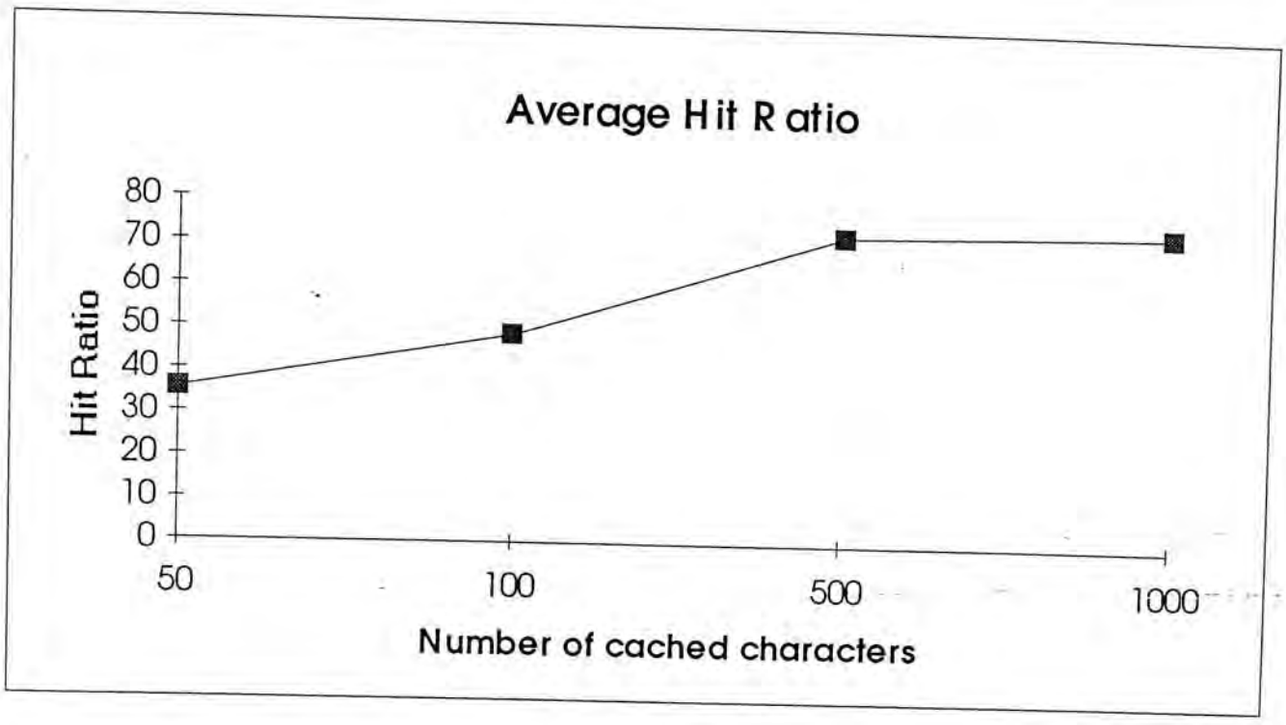


Figure 7-5 Average Hit Ratio with different cache sizes

7.2. Printer Driver Test

The second type of experiment is intended to test the performance of the new printing algorithm for double-byte font. In this experiment, 90 articles are tested and the results are compared with those generated by MS Windows.

A simple typesetting program is used in this test. It major operations include reading Chinese article from file, cutting the long string into several lines so that they can be printed on paper and generating request to the Printer Server for printing. The sizes of the output files generated by the Printer Server are recorded which will be used in the comparison. Fonts of different sizes, namely 10 points, 12 points, 15 points and 18 points are used in the test. Only one Chinese font facename called Sung(仿宋) is used in the test because the facename used does not affect the sizes of output files.

In this test, the Printer Server, Font Server and the typesetting program ran on three different SPARC10 stations running SunOS 4.1.3. The two soft font downloading algorithms are used on two versions of Printer Server and are tested independently. The printer driver is configured for a PCL printer with one mega byte of internal memory. The typesetting program processes the articles one by one, so during the test, the Printer Server serves only one client at a time. In order to compare the algorithms with a currently available product, these 90 articles are also typeset by MS Word 6.0 on MS Chinese Windows 3.1 with a HPPCL5 printer driver using the same Chinese font. All tests of MS Windows are done on a 80486-50MHZ PC.

7.2.1. Testing with 10 points font

As the file size of input file vary from several hundred bytes to a hundred thousand bytes, the variation in output file size is very great. In order to show clearly the different between current printer driver implementations and MS Windows printer driver, the data gathered are divided into five charts. Each chart shows only the data generated from input files whose sizes fall in the specified range.

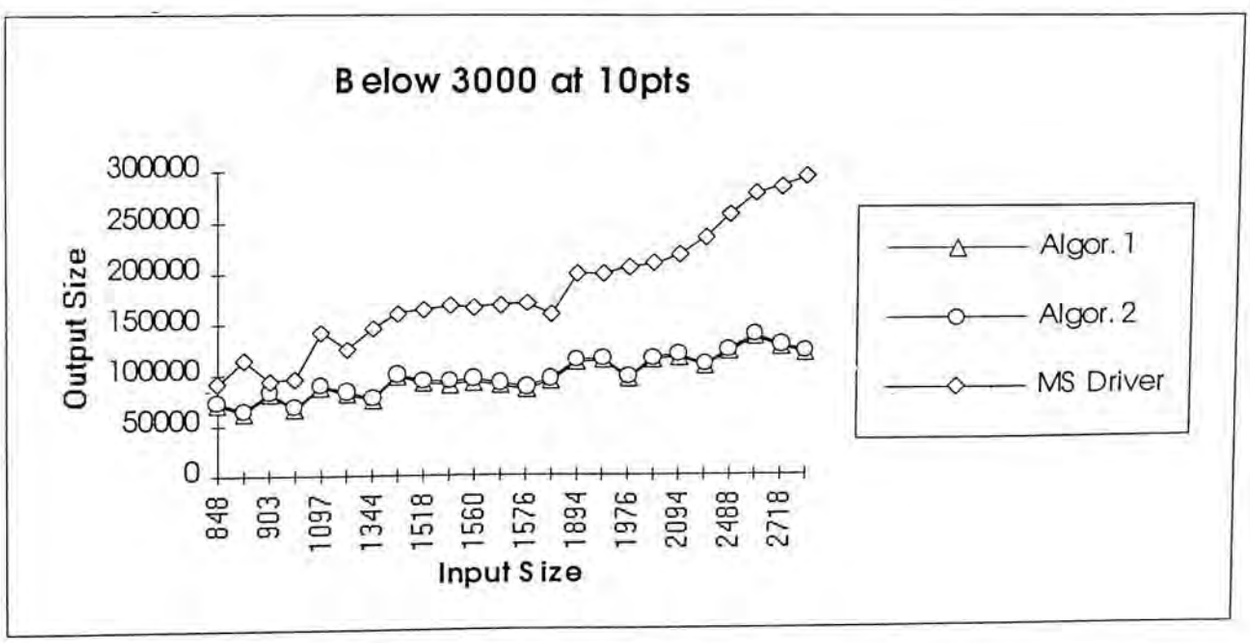


Figure 7-6 Files with sizes below 3000 typeset at 10 points

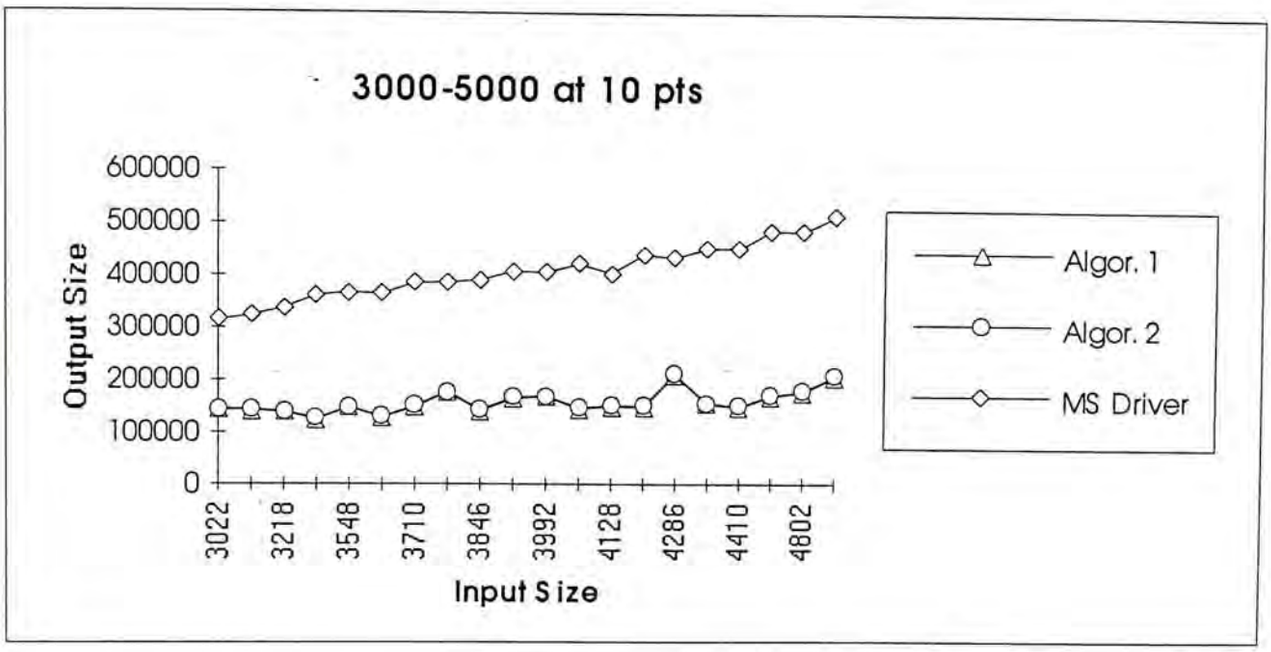


Figure 7-7 Files with sizes between 3000 and 5000 typeset at 10 points

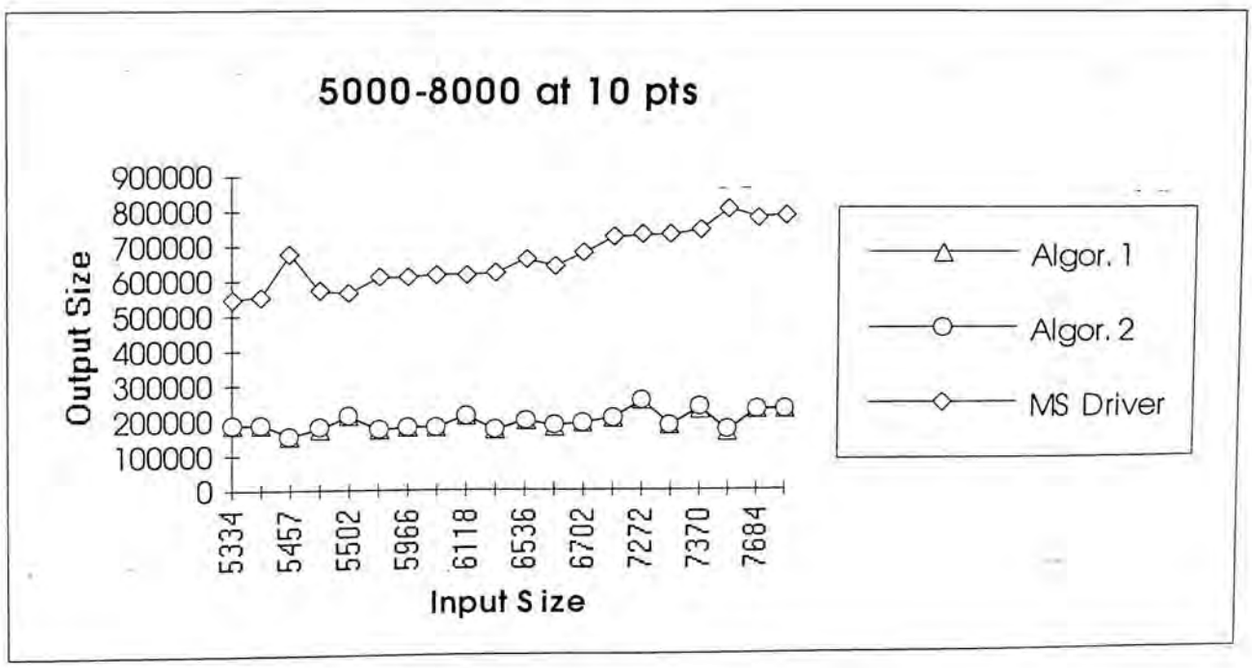


Figure 7-8 Files with sizes between 5000 and 8000 typeset at 10 points

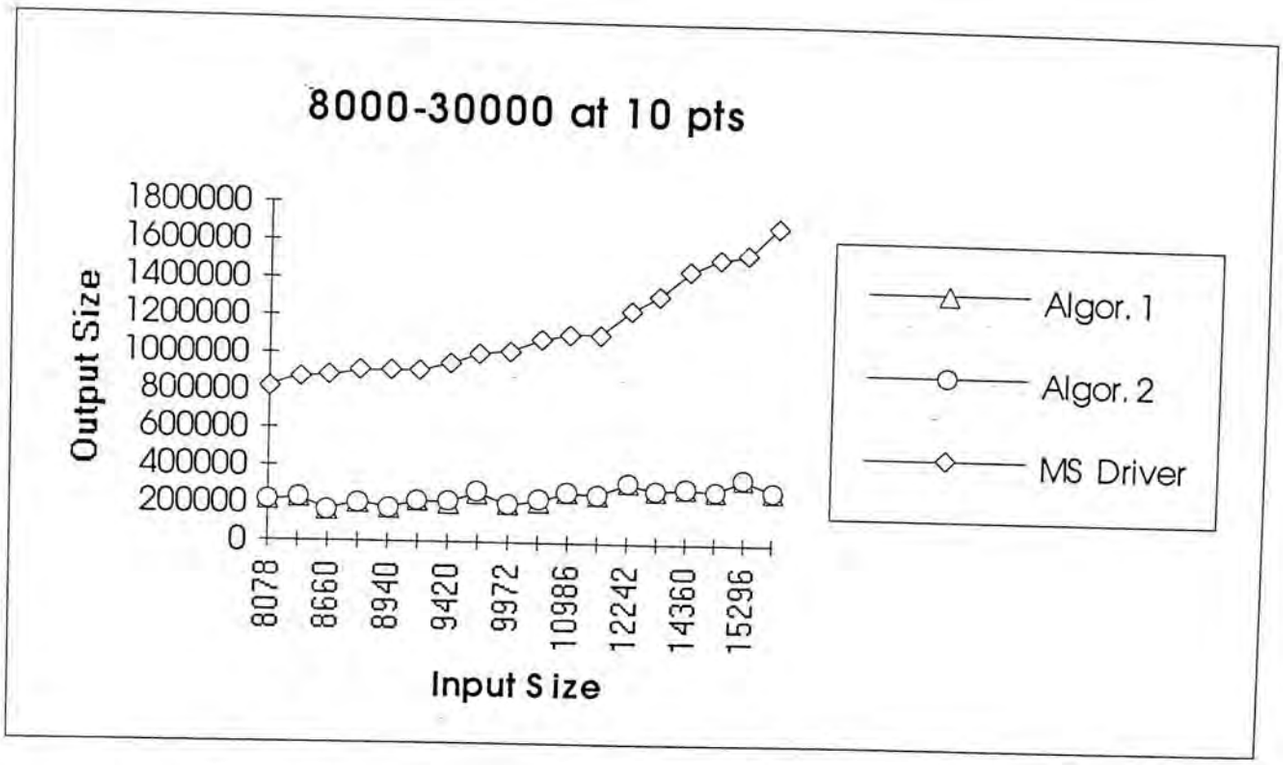


Figure 7-9 Files with sizes between 8000 and 30000 typeset at 10 points

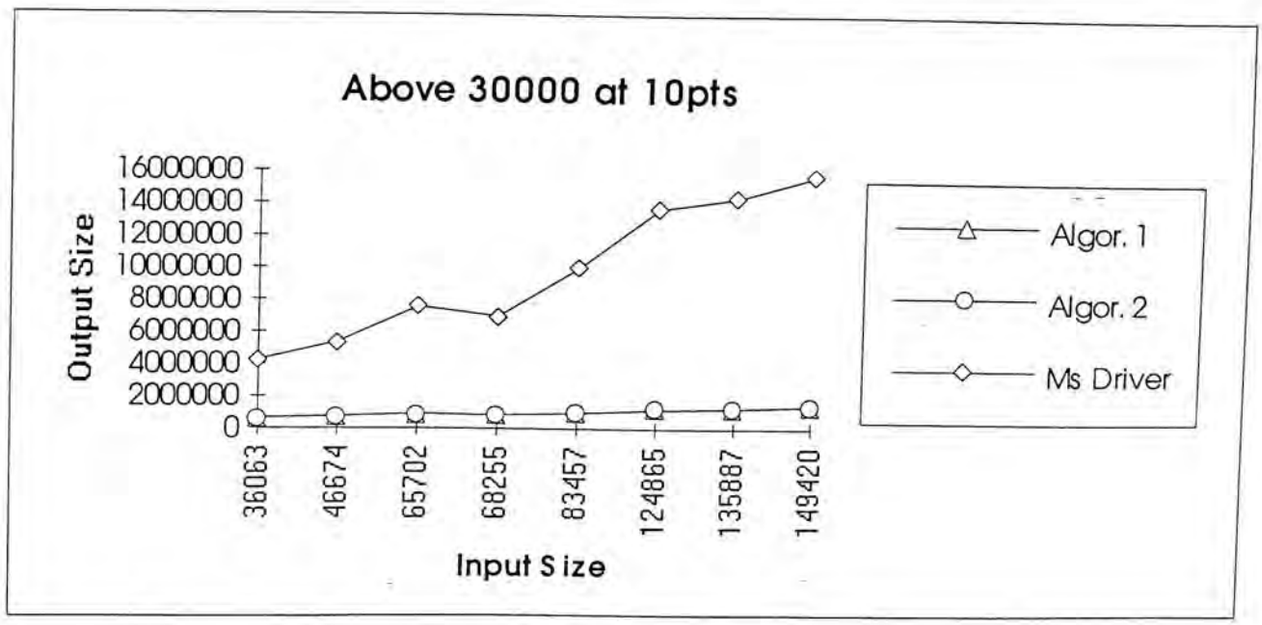


Figure 7-10 Files with size above 30000 typeset at 10 points

7.2.2. Testing with 12 points font

The following charts describe the testing results of the 90 documents typeset at 12 points. Similar to data gathered from files typeset at 10 points, data from this experiment are also divided into five charts. This arrangement will also be applied to the following sections.

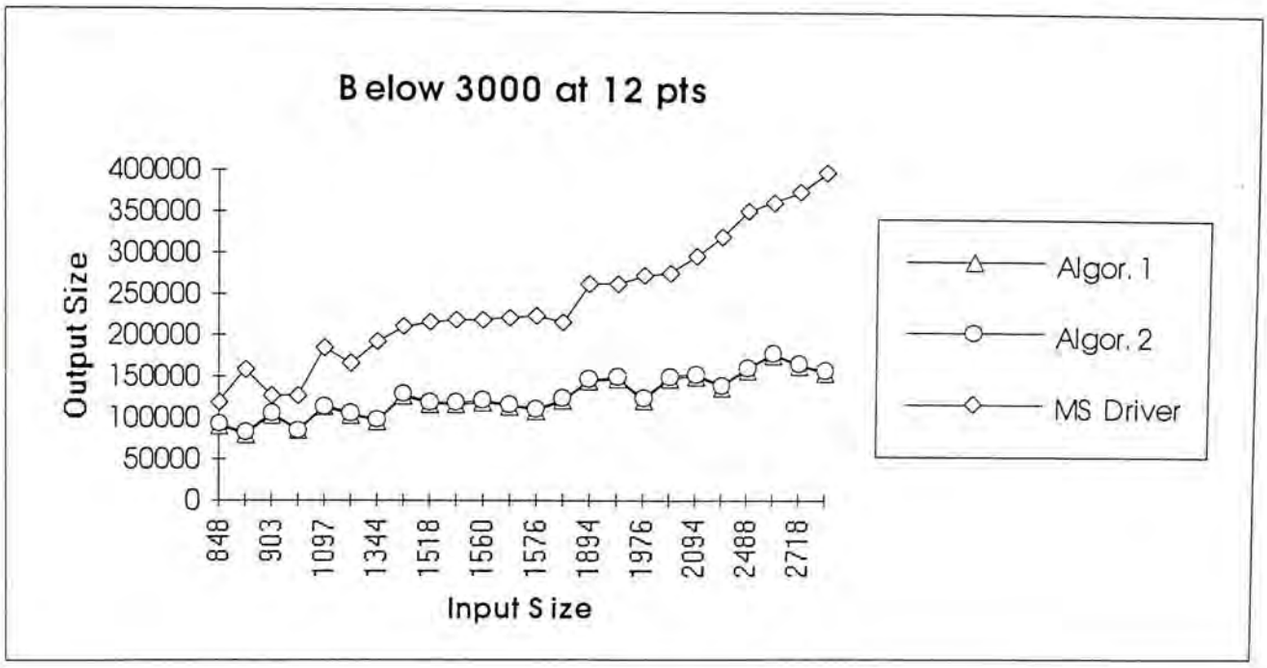


Figure 7-11 Files of sizes below 3000 typeset at 12 points

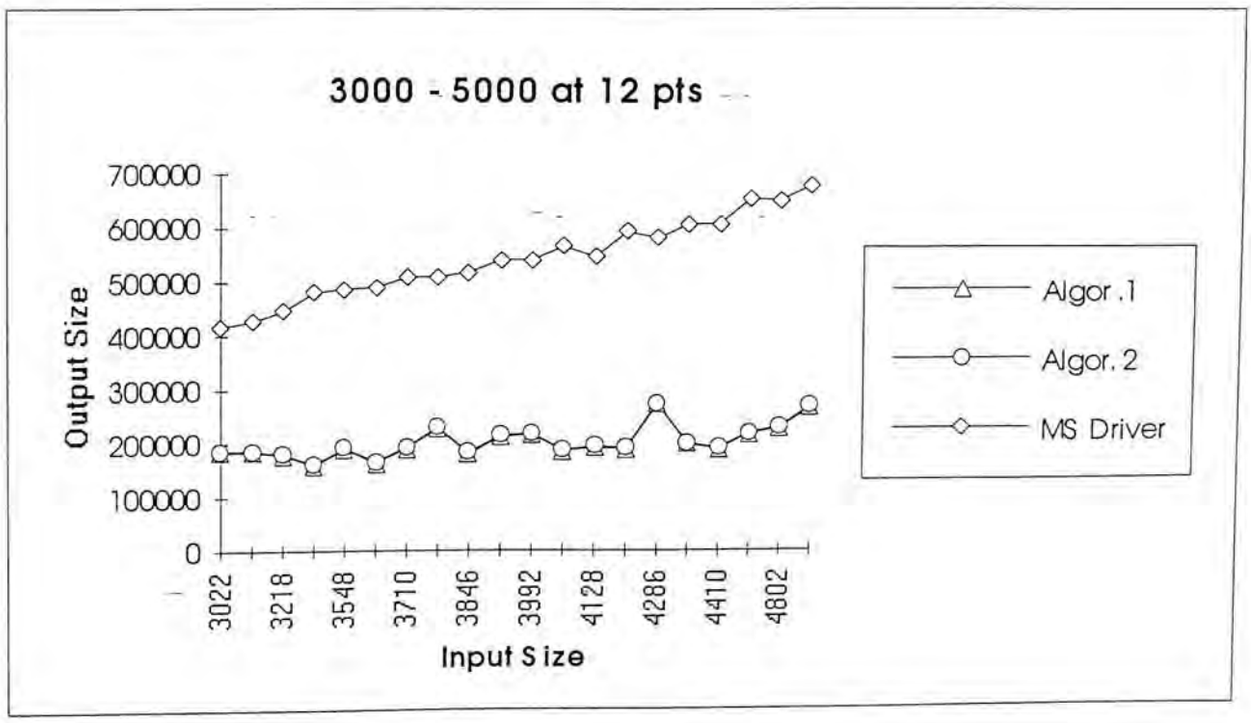


Figure 7-12 Files with sizes between 3000 and 5000 at 12 points

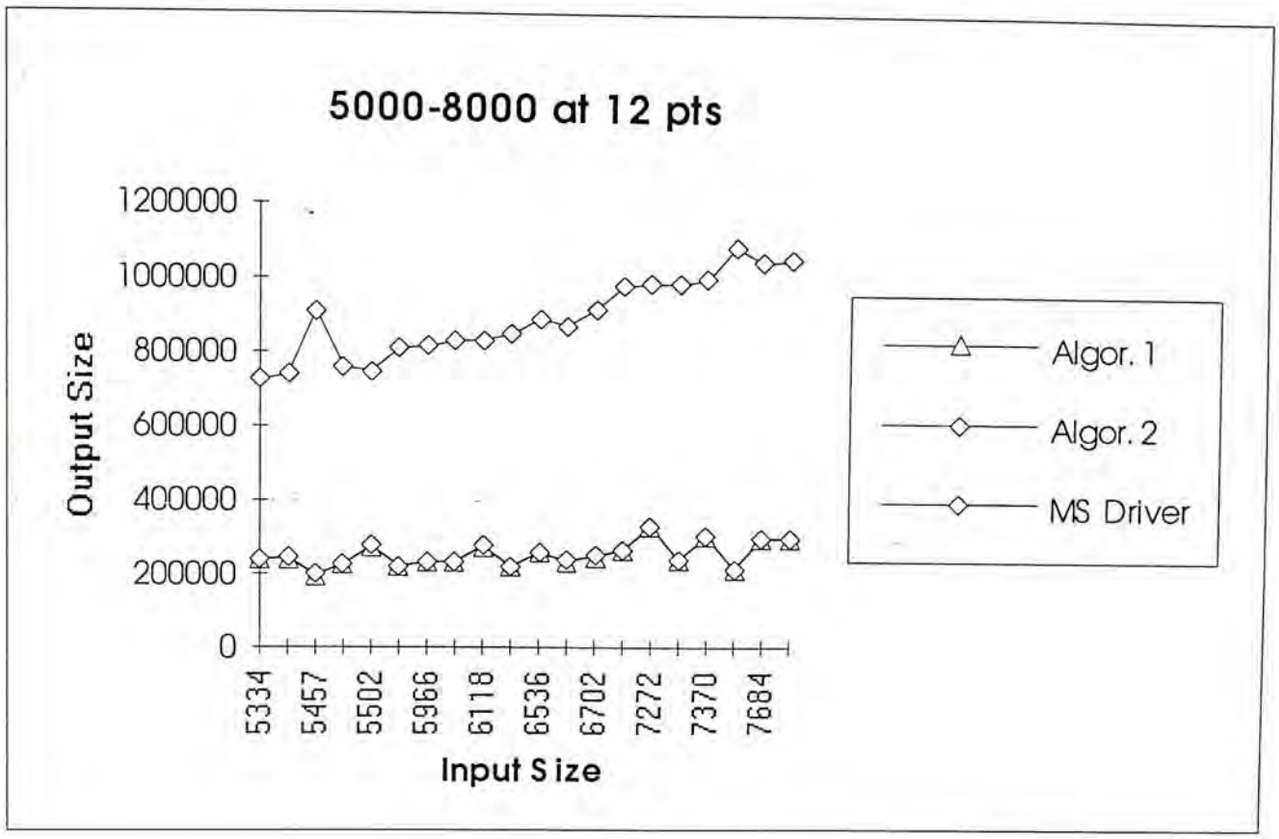


Figure 7-13 Files with sizes between 5000 and 8000 typeset at 12 points

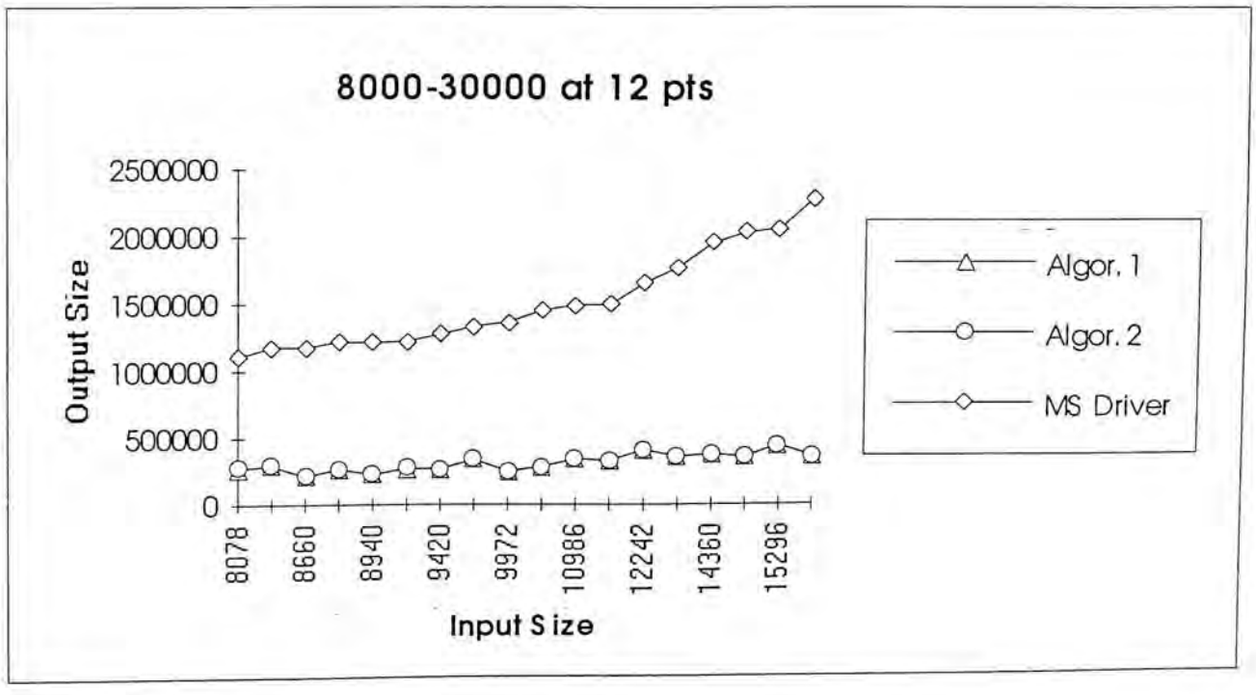


Figure 7-14 Files with sizes between 8000 and 30000 typeset at 12 points

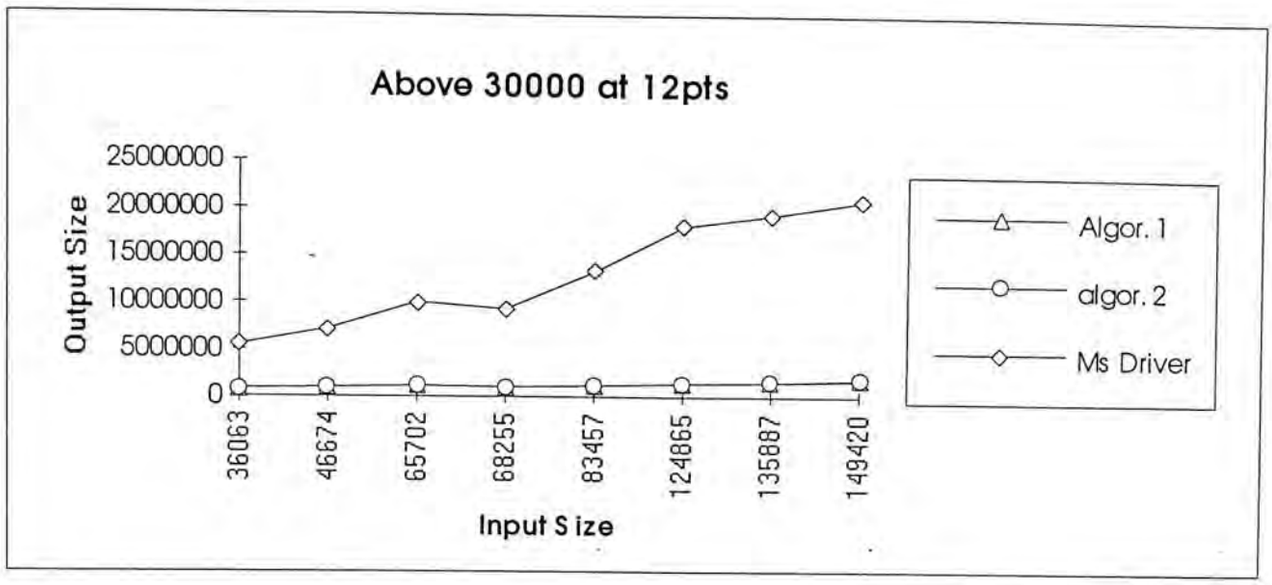


Figure 7-15 Files with sizes above 30000 typeset at 12 points

7.2.3. Testing with 15 points font

The following charts show the experimental results of files typeset at 15 points.

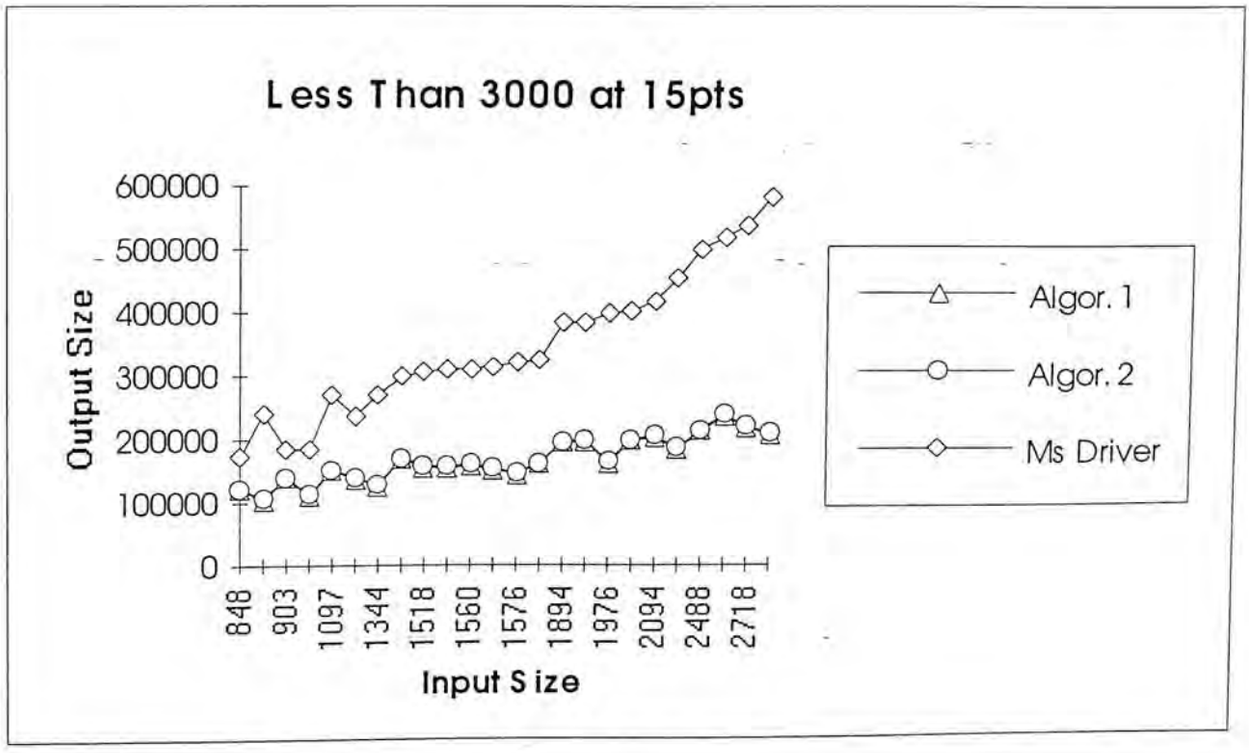


Figure 7-16 Files with sizes below 3000 typeset at 15 points

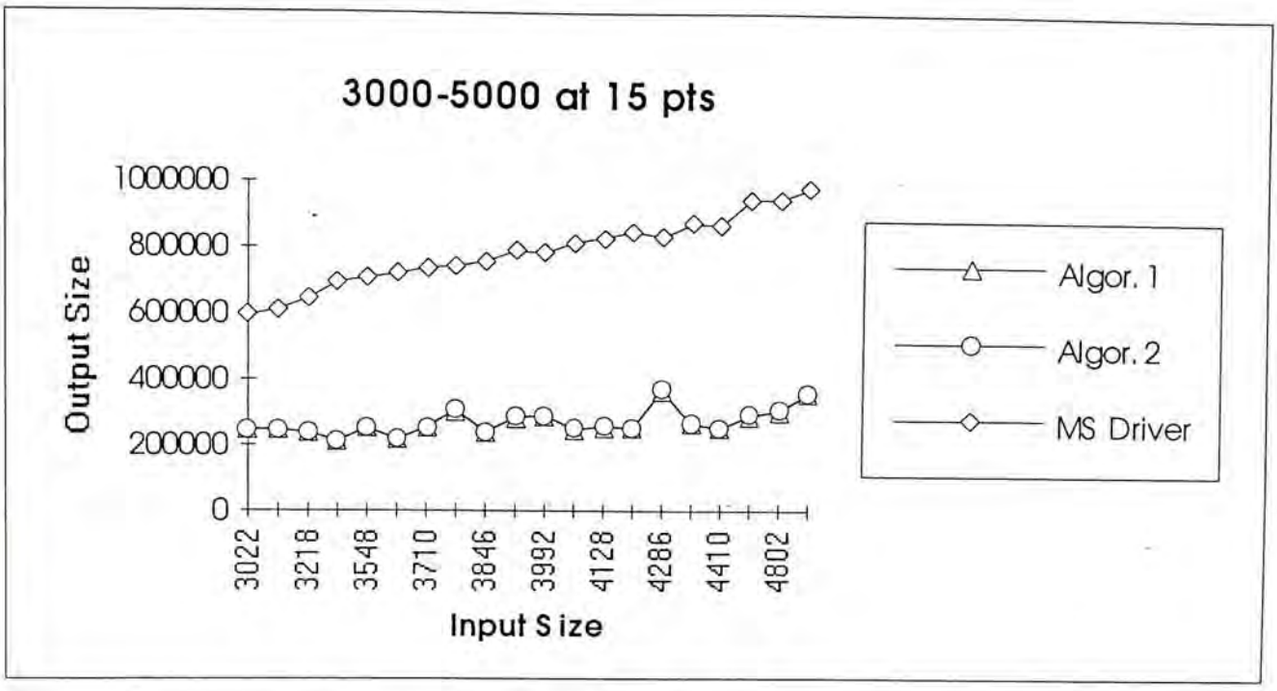


Figure 7-17 Files with sizes between 3000 and 5000 at 15 points

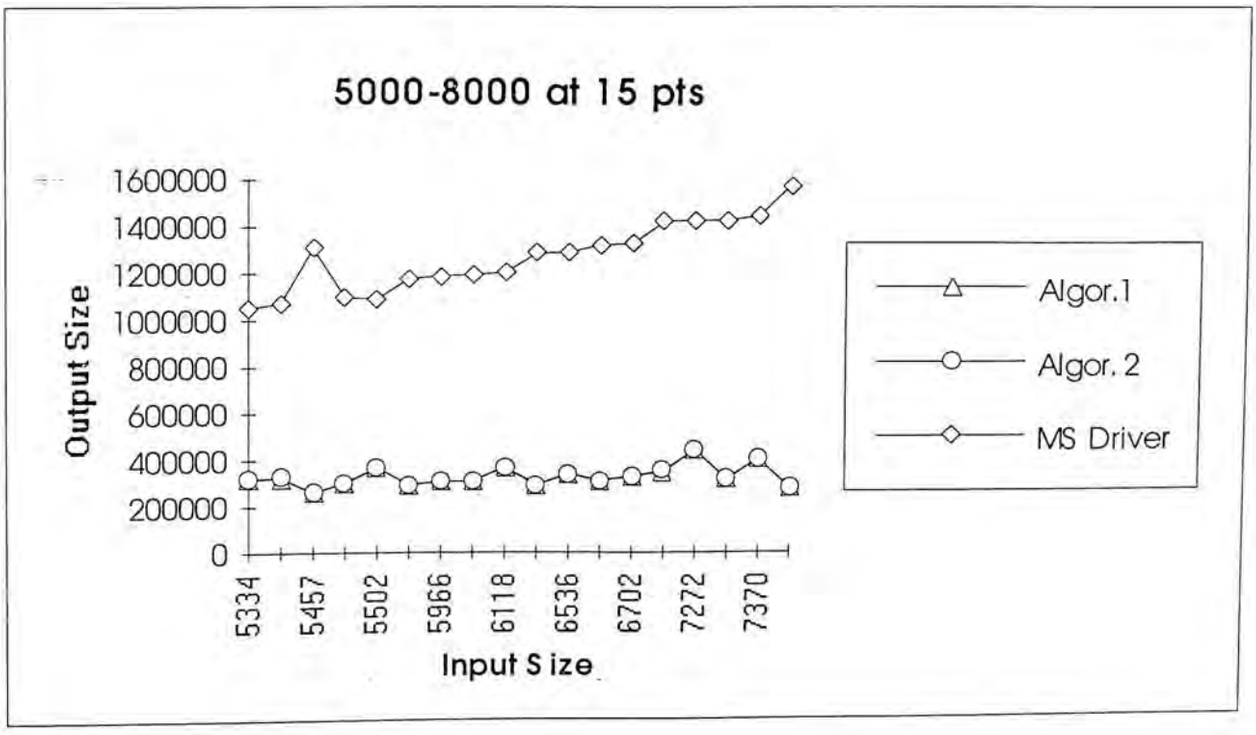


Figure 7-18 Files with sizes between 5000 and 8000 typeset at 15 points

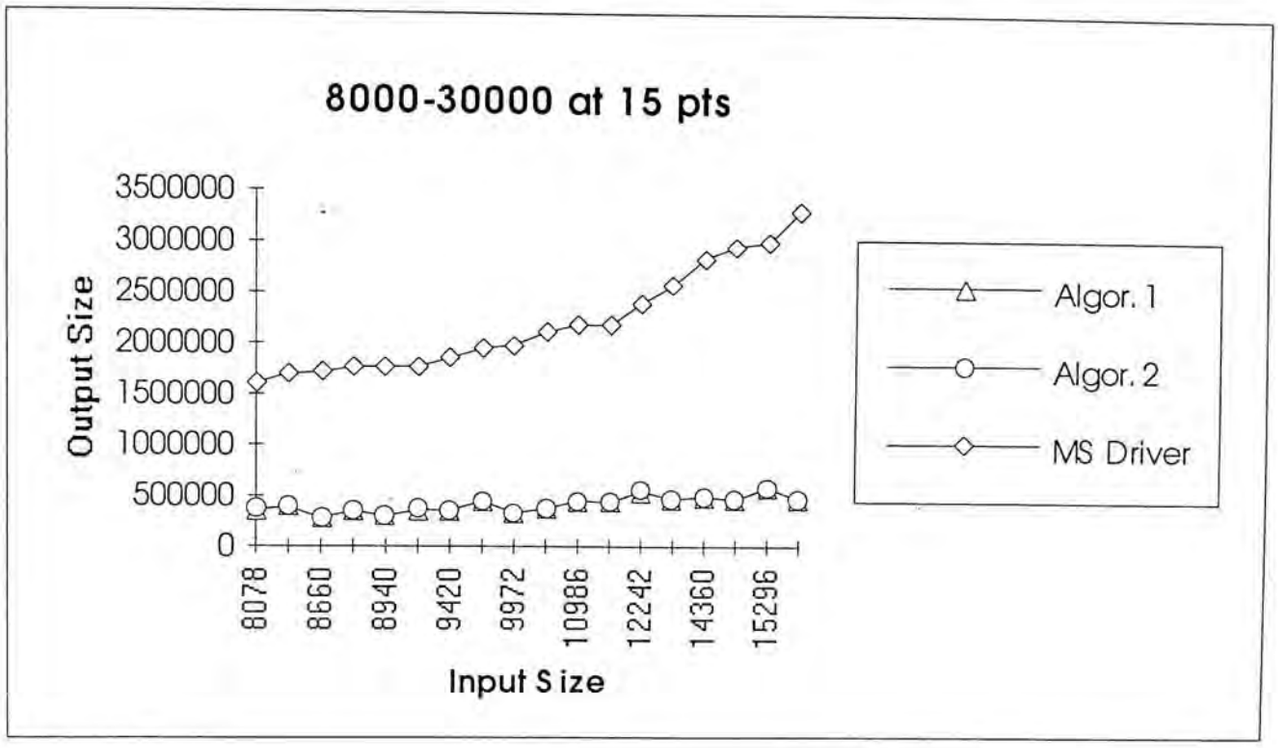


Figure 7-19 Files with sizes between 8000 and 30000 typeset at 15 points

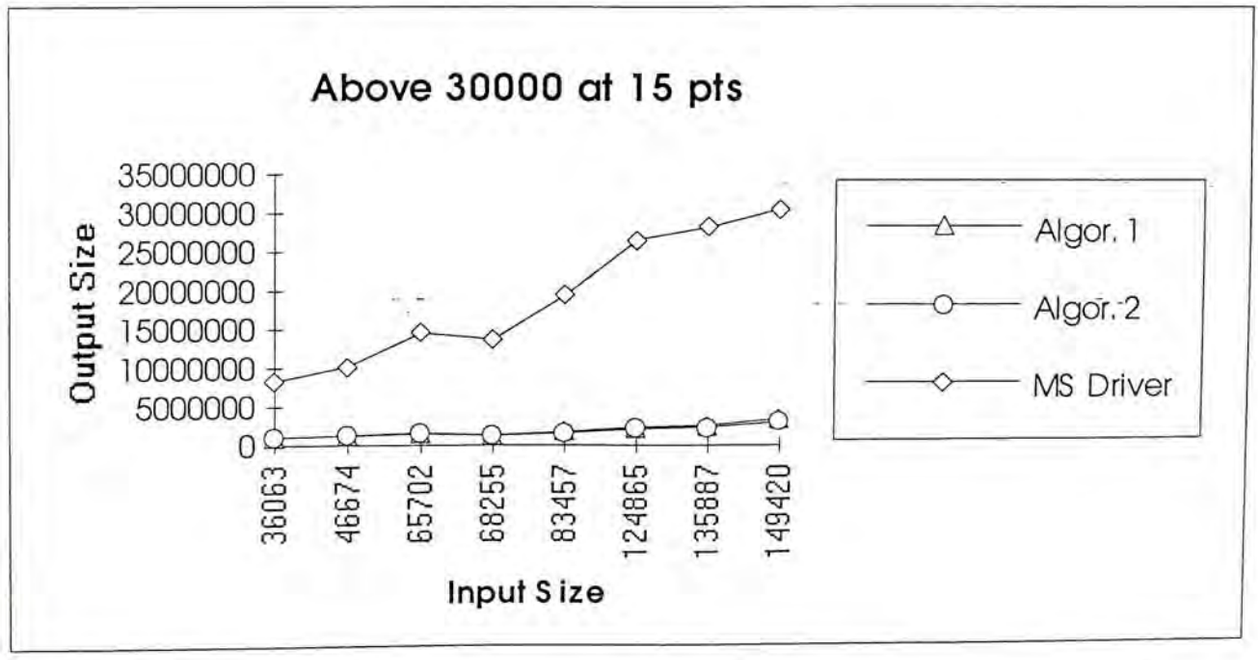


Figure 7-20 Files with sizes above 30000 typeset at 15 points

7.2.4. Testing with 18 points font

The following charts show the testing results of the documents typeset at 18 points.

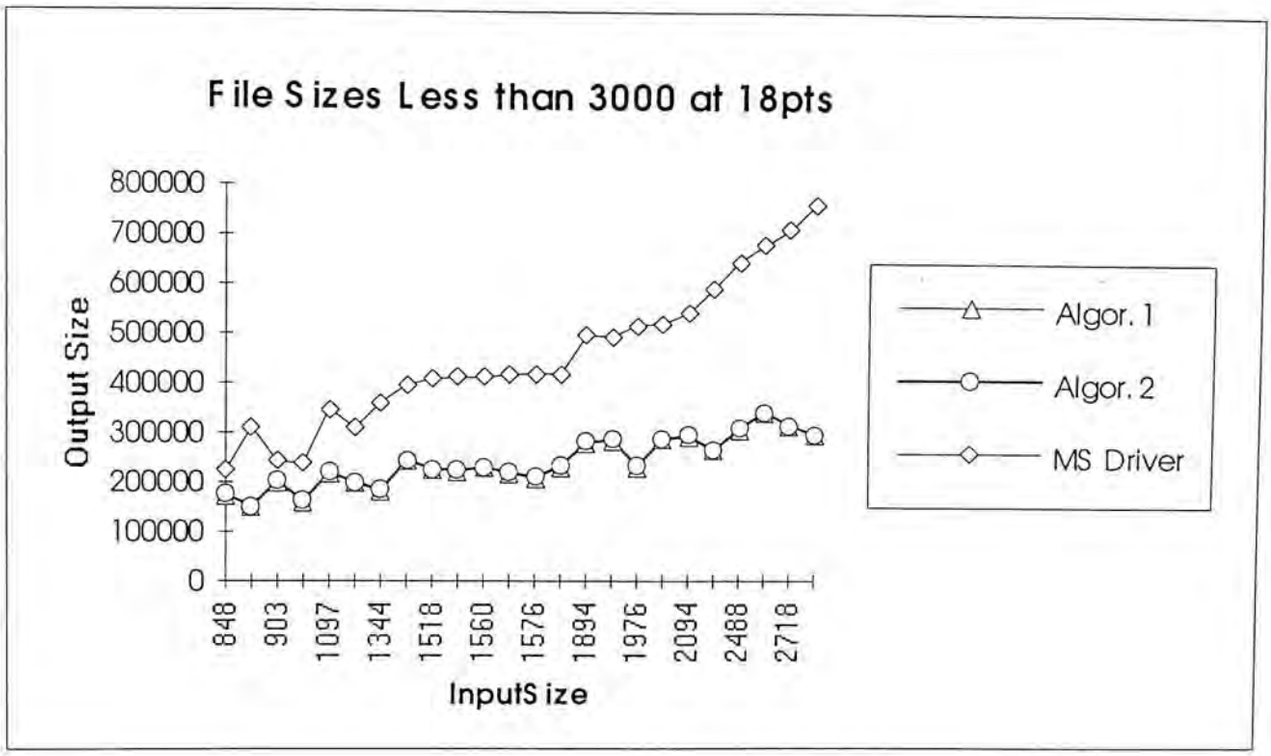


Figure 7-21 Files with sizes less than 3000 typeset at 18 points

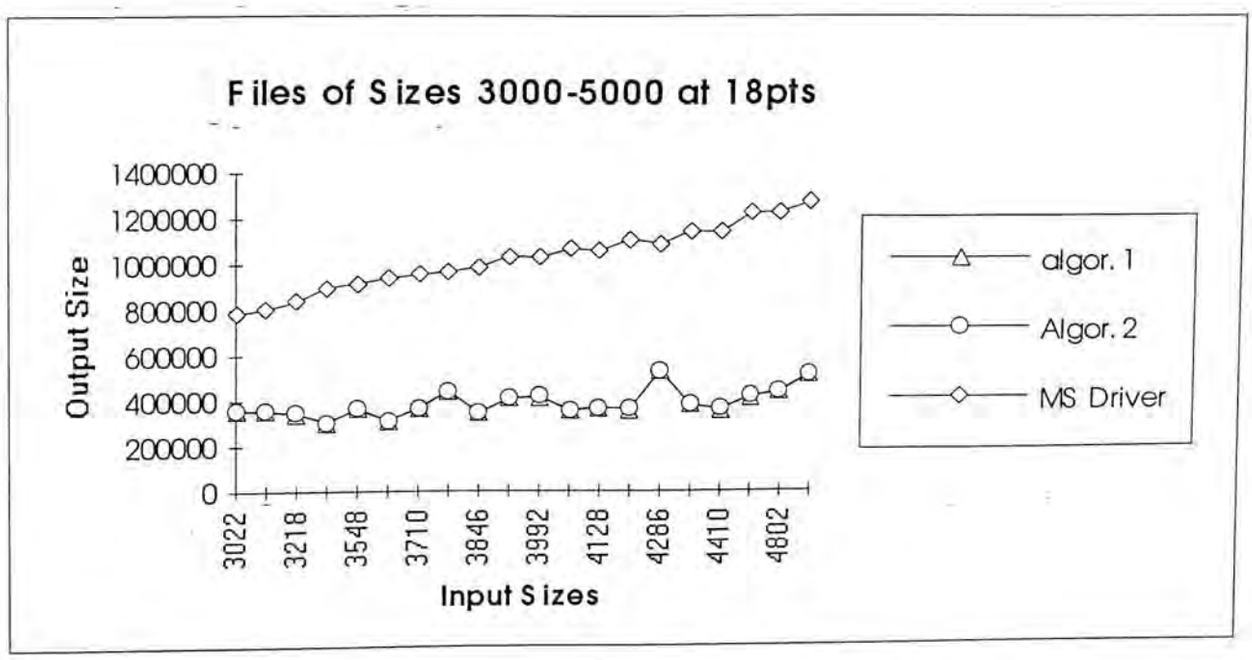


Figure 7-22 Files of sizes between 3000 and 5000 typeset at 18 points

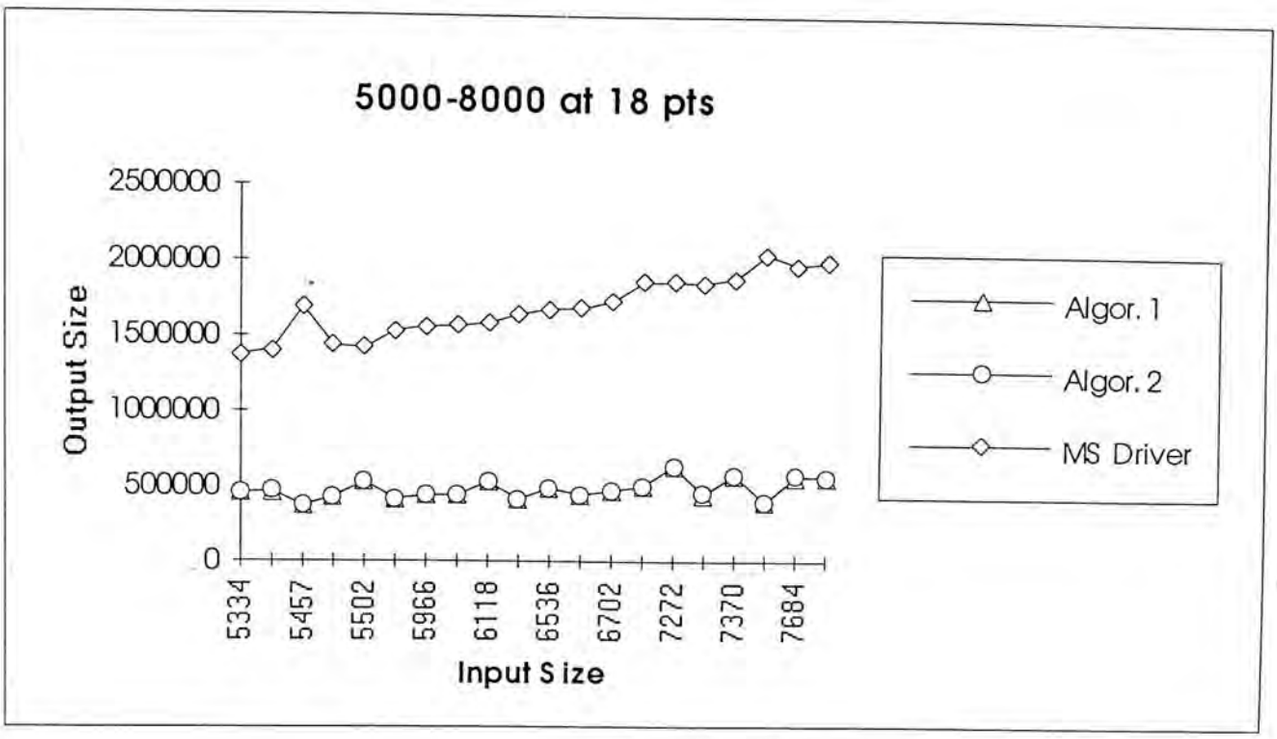


Figure 7-23 Files with sizes between 5000 and 8000 typeset at 18 points

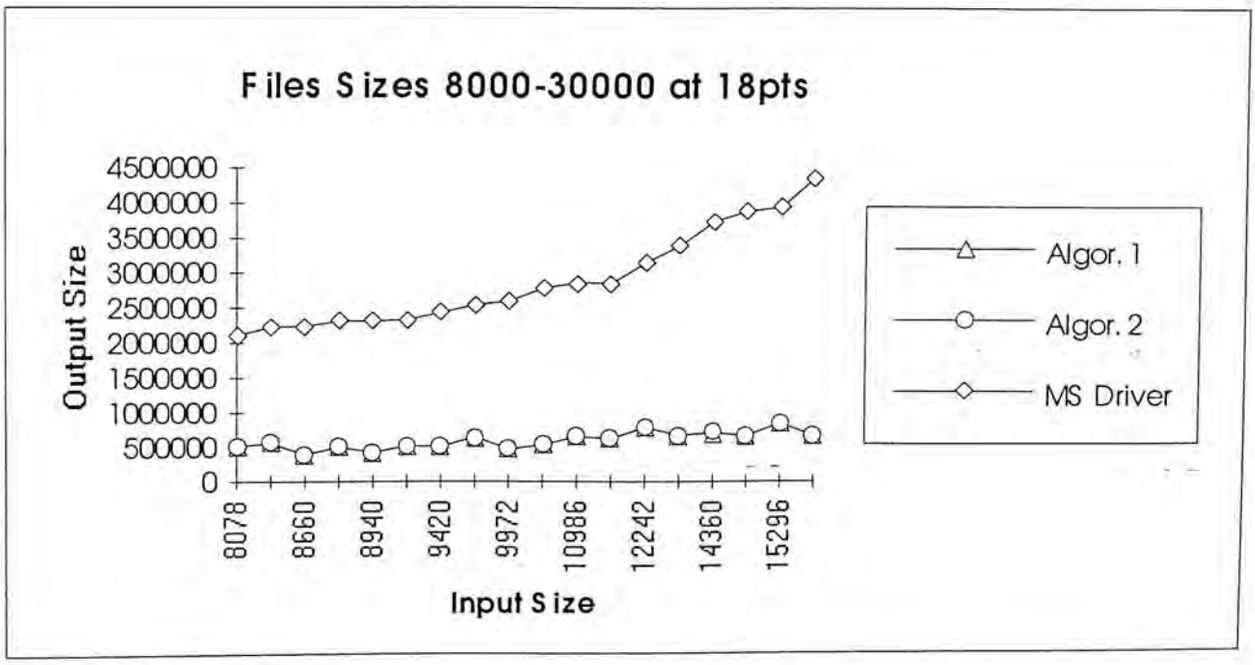


Figure 7-24 Files with sizes between 8000 and 30000 typeset at 18 points

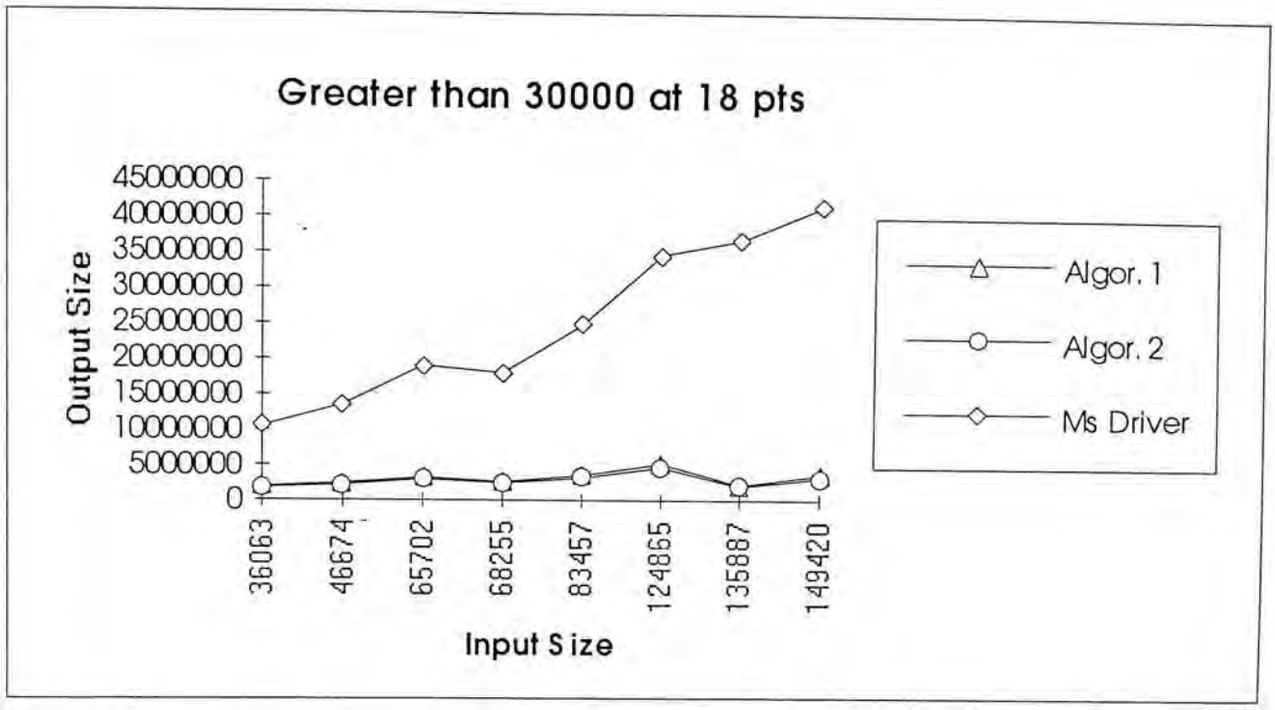


Figure 7-25 Files with sizes above 30000 typeset at 18 points

7.3. Time Measurement

The times required to typeset the 90 articles are also measured. However, the these data are not used in comparing the two algorithms and printer driver from MS Windows for two reasons:

1. The two algorithms are not the key factor as far as the speed of typesetting is concerned since there are network communication and remote font acquisition over head.
2. Besides the network communication issue, the Printer System and MS Windows run on different machines. There is no point in comparing running time of software which run on machines with different speed.

The time measurement given below is just for reflecting the speed of the Printing System. It is not intended for any comparison.

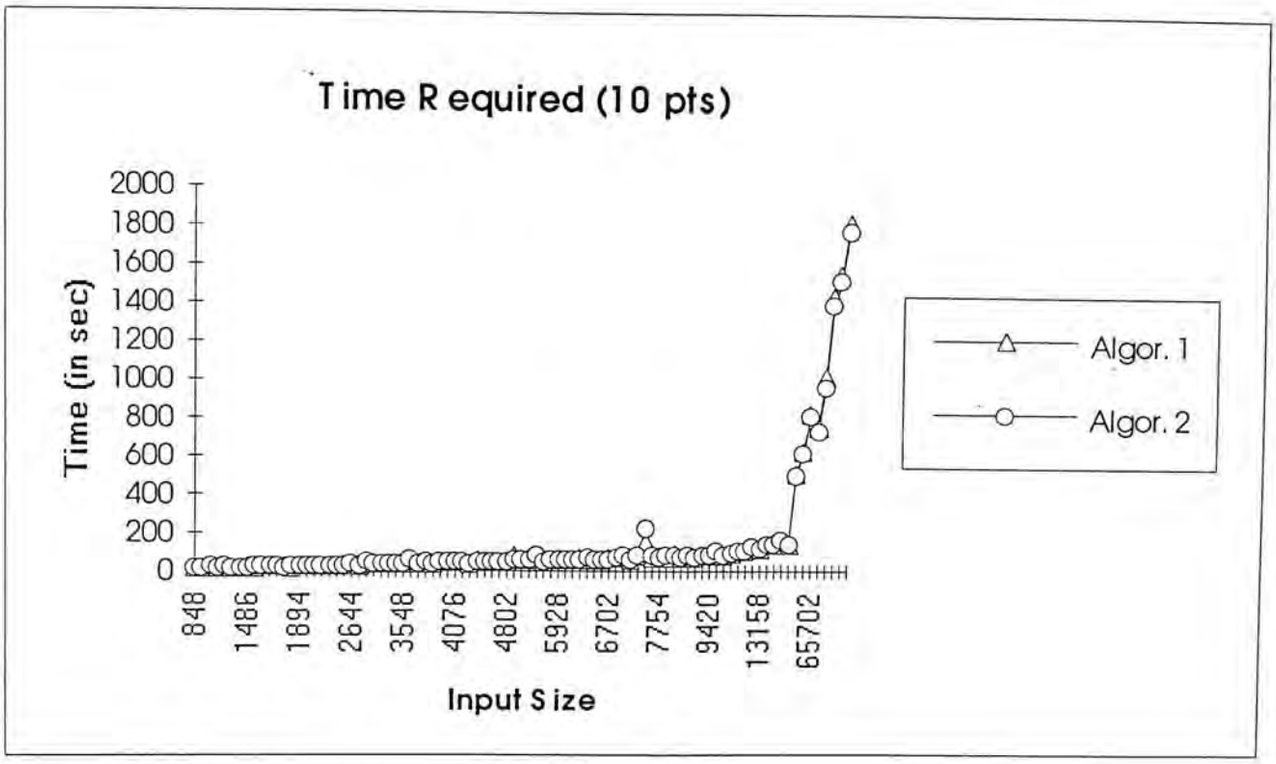


Figure 7-26 Time Required to typeset articles at 10 points

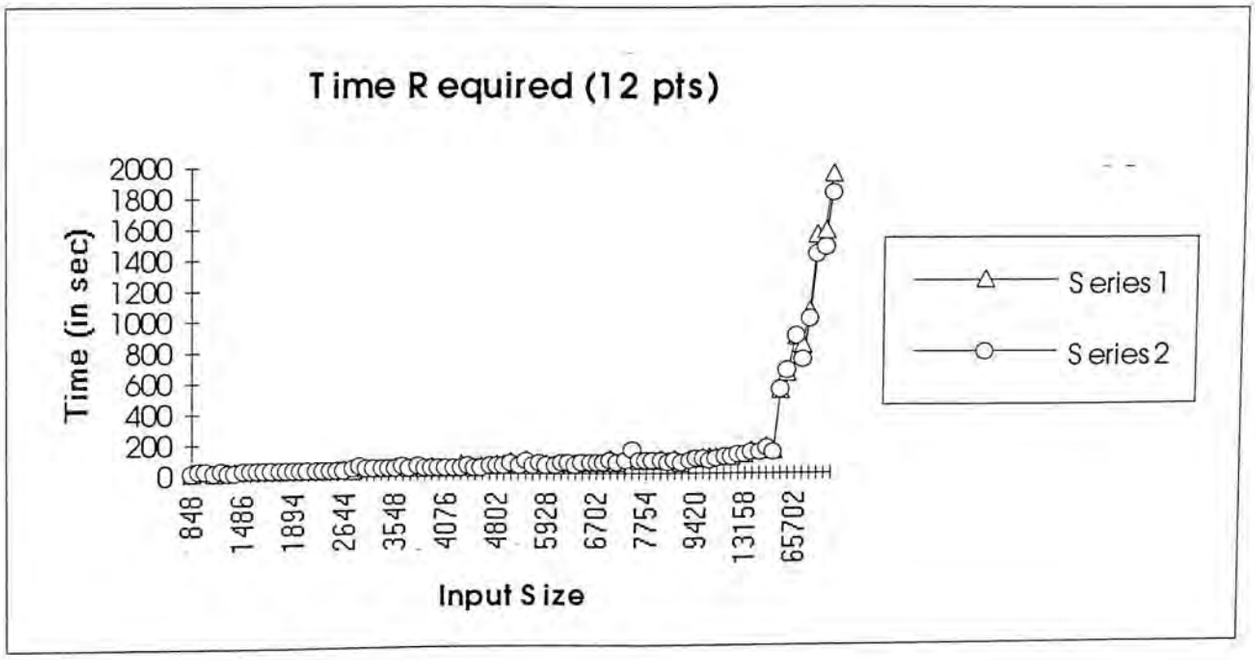


Figure 7-27 Time required to typeset articles at 12 points

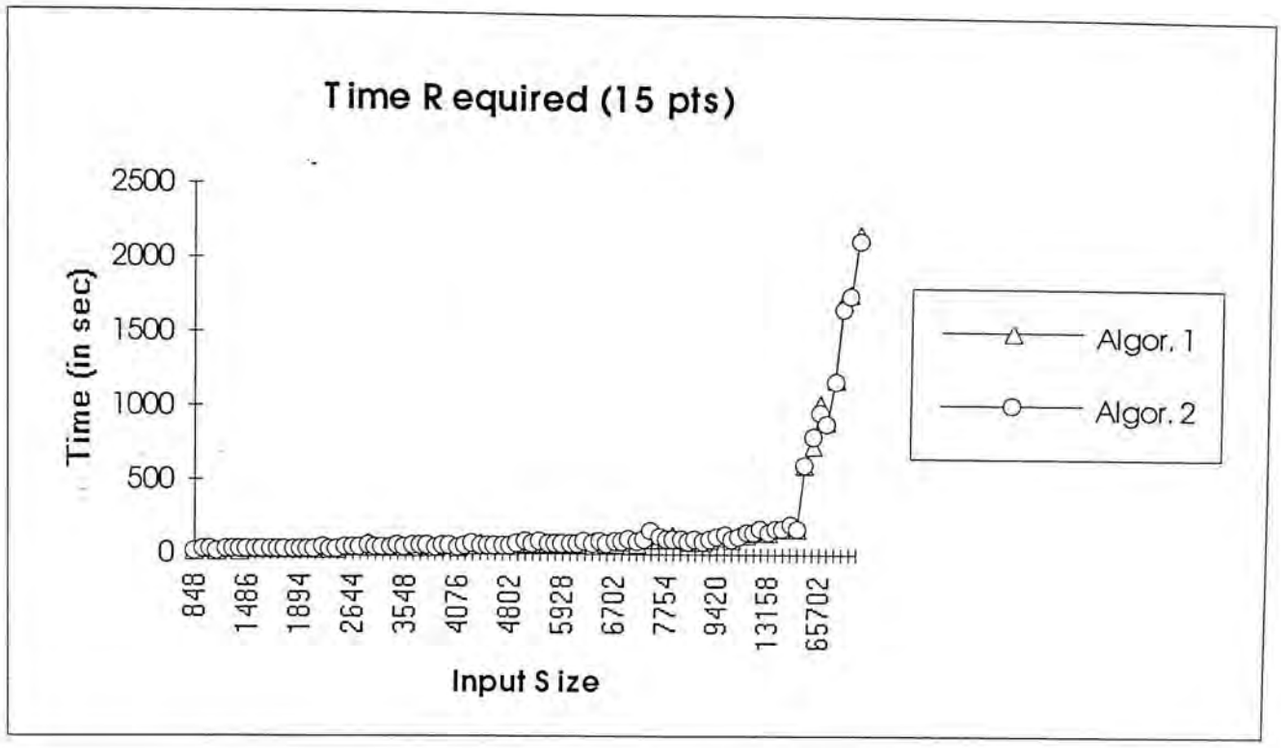


Figure 7-28 Time required to typeset articles at 15 points

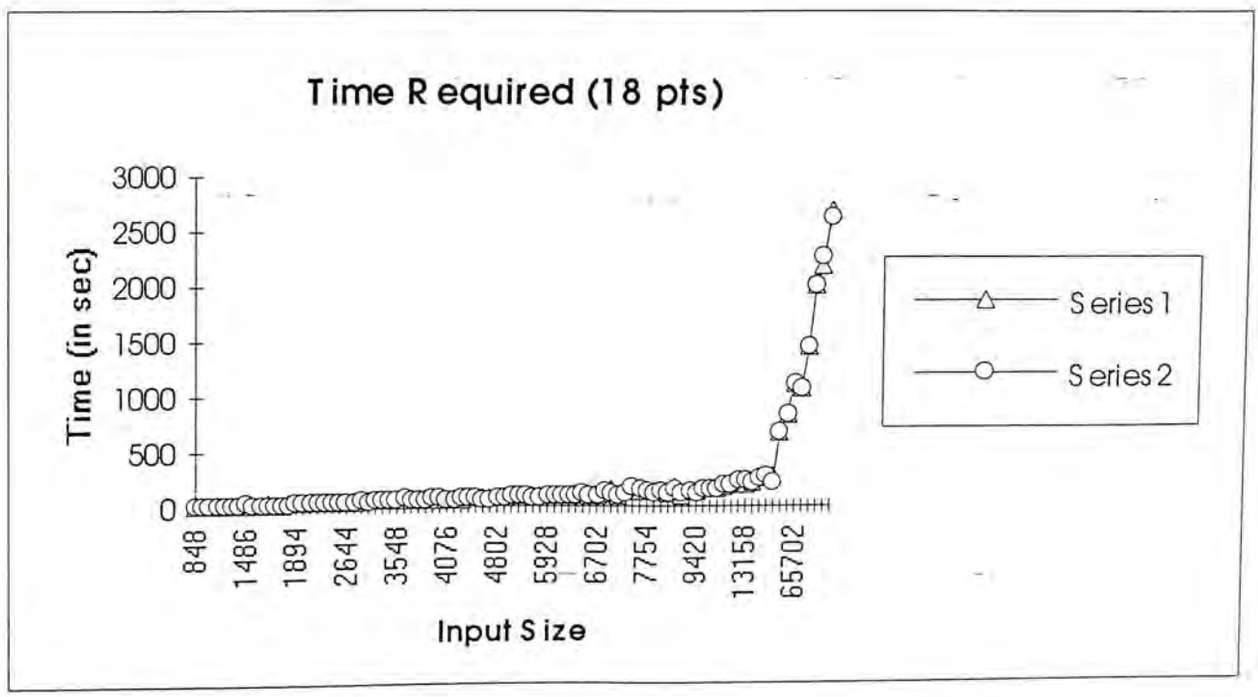


Figure 7-29 Time required to typeset the articles at 18 points

7.4. Discussion

In all the above experiments, the output file size is measured instead of actual printing time. There are two reasons for this decision. First, the 90 articles are tested with 4 different font sizes and 3 different printing algorithm, there are altogether 1080 different combinations. In terms of number of pages, the output generated from the 90 articles ranging from one page to about 110 pages. Assume that on average 50 pages are generated from each combination of experiments, there will be a total of 54,000 pages. If all the output files are printed, there will be a total of 54,000 pages printed. It is a waste of resource to print all pages out and the amount of labor work involved is well beyond the limit of this project.

Second, since the time required to transfer the data from the computer to the printer usually outweigh the time required for the printer to process the data[42], one can infer that the larger the file, the longer the printing time since the transfer time is longer. On the other hand, since the files generated by MS Windows are compressed, so even with the same file size, the time required to print a compressed graphic file is still longer than a file of the same size which contains only soft font and text only.

Lastly, the printing time measured from the selectively printed output files is also helpful in estimating the printing time. From a rough estimation, each page generated by MS Windows contains about 250K bytes of data, i.e. a file of size 20 mega bytes is about 80 pages long. The time required by a LaserJet III to print a page

of compressed graphic is about 40 seconds. So an 80 pages document need about 53 minutes.

The estimations of printing time for files generated by the new algorithms are more complex since the time required not only depends on the output size, but also on the working set of the input document. Very roughly, a page generated by the two algorithms contains 25K to 40K bytes of data and the time required to print a page is about 15 seconds[7].

From the charts shown in the previous sections, it can be observed that the sizes of output files generated by MS Windows increase when the input files increase and when the point size of the font used increases. It is due to the fact that MS Windows print double-byte font as graphics, the larger the input files, the larger the output file sizes.

The files generated from MS Windows are, in almost all the cases, much greater than that of current printer driver implementations. The differences in file sizes are especially high when the input files are large. For example, for a file with about 30,000 bytes , the output file generated when it is typeset at 10 points is about 4 mega bytes while that of current printer drivers are both less than 1 mega. When a file with size of about 120,000 bytes is typeset at 10 points, the file generated by MS Windows is about 11 mega bytes while that of current printer driver algorithms are still less than one and a half mega bytes. The two printer drivers in current

[7] The page sizes given here are calculated by dividing the output file size by the number of pages. While the per-page printing time is observed from printing of 10 files.

implementation do have a great improvement over the printer driver from MS Windows.

From the charts shown above, the performances of the two soft font downloading algorithms are very close to each other. Generally, it is difficult to tell which one is better. However, for long input files with font sizes, the performance of the second algorithm is better, as shown in Figure 7-30 and Figure 7-31.

Algorithm two is also more preferable than algorithm one for its simplicity. In fact, algorithm two is conceptually more simple than algorithm one since it maps the codes by a mathematical function instead of by searching. It is also easier to be implemented than algorithm one.

Furthermore, there is in fact one case that cannot be handled by the current implementation of algorithm one. Since the minor_font_id is always increasing, there will be a chance that the minior_font_id will reach its maximum value 255. At this time, if the minor_font_id increases one more, an error would occur and the injectivity of the algorithm would be destroyed. Additional measures should be employed to guarantee that the minor_font_id should not be greater than 255. One possible solution is to delete all fonts in the printer whose major_font_id is equal to the one whose minor_font_id reaches the upper bound. The deletion of all related fonts is necessary for resetting the minor_font_id to zero. However, this would reduce the hit ratio of the downloaded printer fonts since some fonts are deleted unnecessarily Algorithm two does not suffer from this problem since entries in the DownLoadFont table are independent from each other.

In conclusion, algorithm two is recommended for implementation of Chinese printer driver.

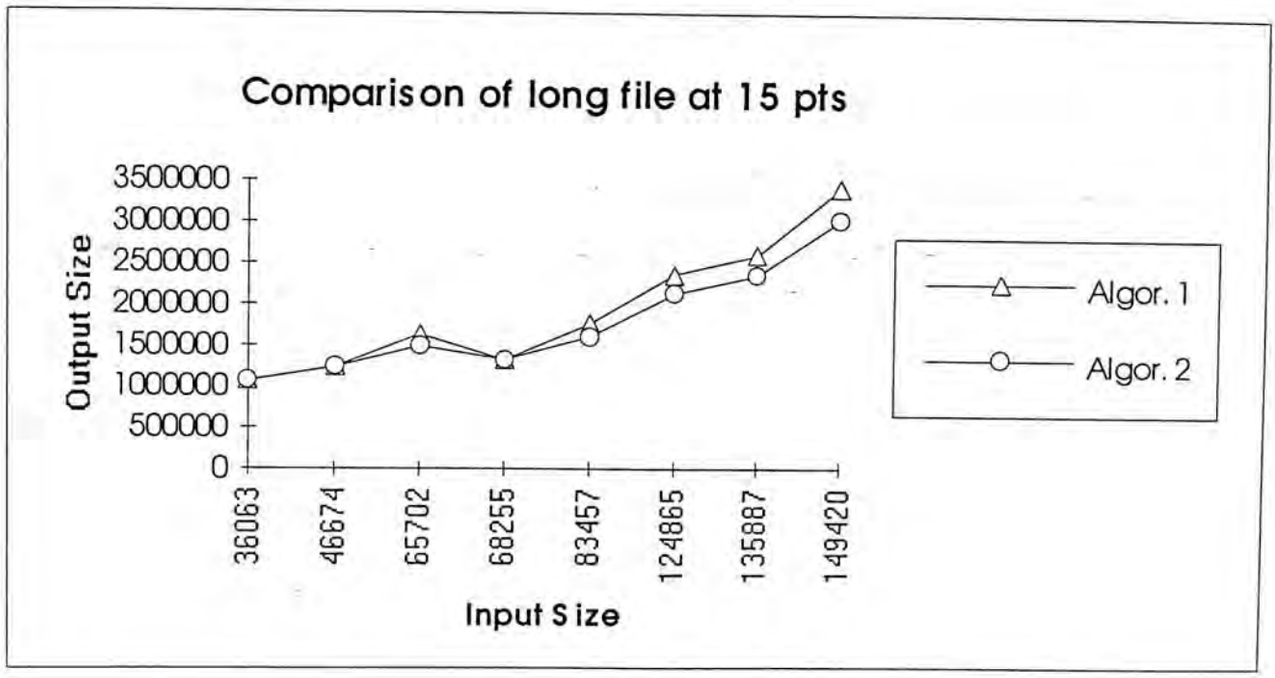


Figure 7-30 Comparing two algorithms using long files at 15 points

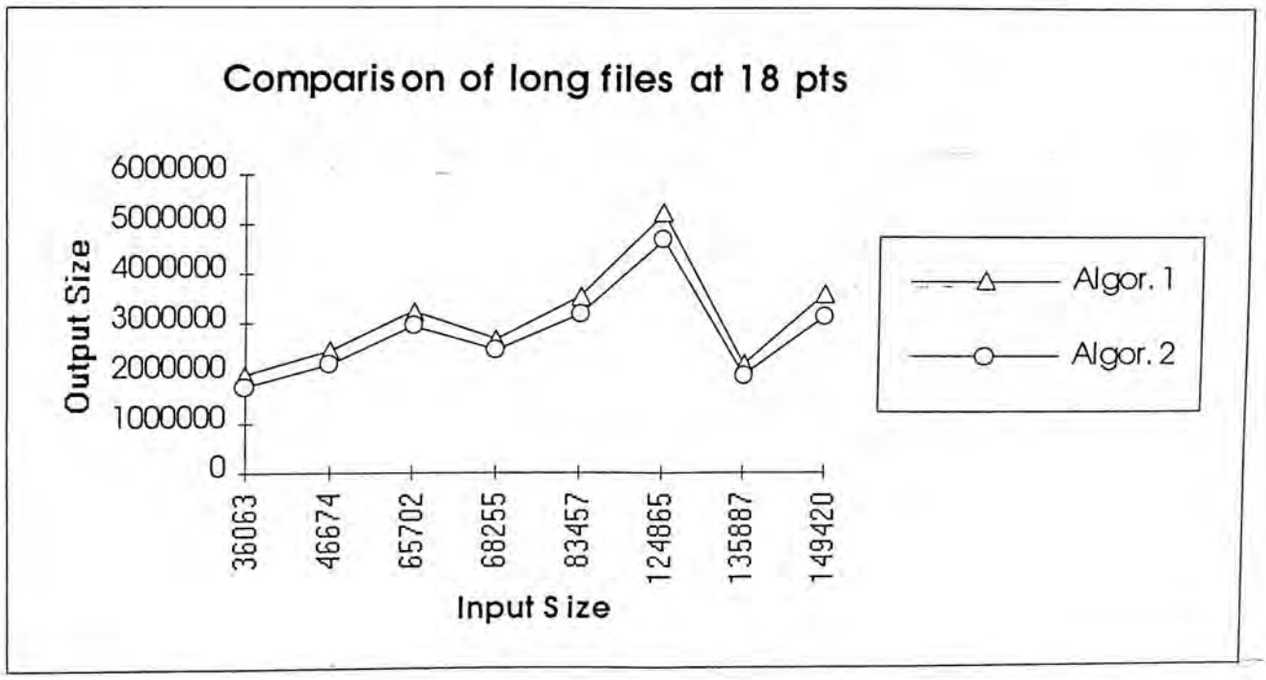


Figure 7-31 Comparing two algorithms using long files at 18 points

The last thing to be discussed is the time measurement. As shown in Figure 7-29, the longest time required is 45 minutes which is the time required to typeset a document with about 150 pages at 18 points. The Printing Subsystem can, on average, typeset 3 pages per minute when only one print client is using the system.

The speed of the system is acceptable because unlike screen display, printing requests are not sent very often. For the requests such as those just wants the metric information of a paragraph of text, the response comes almost immediately. The only time printing requests come in a large batch is when the client wants to perform the actual printing. At this time, the user is usually willing to wait longer before the printing finishes. Furthermore, on the operating system where multi-tasking is supported, the application program can indeed fork a copy of itself to handle printing in the background while letting the user to continue his work.

7.5. Further Improvement

There are several areas in the current implementation that have to be improved to make the system more comprehensive and efficient. First, the requests protocol should be extended to include graphic printing primitives such as line drawing, curve drawing and polygon filling etc.[8]. The text printing system should also be extended to support more complex text printing such as drawing a string with a clipping rectangle and drawing vertical text etc..

On the other hand, the response time of the system increases when the number of clients increases. This is partly due to the fact that in current implementation, the Print Service Protocol is designed to be synchronous. A print client must wait for the reply before it can send another request. If asynchronous protocol is used, the print client can continue its own work without being blocked by the Printer Server. Synchronous protocol is simple and easy to implement. For an experimental system, synchronous protocol is more appropriate. However, on a system where efficiency is of more importance, asynchronous protocol should be used[39].

Efficiency can also be increased by making a clone of the Printer Server. When the limit of the Printer Server is reached, the Printer Server can fork a copy of itself to handle the new coming clients. Making a new copy of the Printer Server improves the response time of the system, however, it also makes the sharing of resources more complex. The font cache should be designed such that it can be shared by several copies of the Printer Server or otherwise, each copy can only use font data store in its font cache.

8. Conclusions

This project started with the idea to enhance the Chinese fonts handling facilities in X Window System and ends up with a Printing System for both single byte and double-byte fonts which can be used by not only the X but also other independent applications.

A Chinese Font Server is firstly implemented to enrich the Chinese fonts source in X. The initial design decision for making the Chinese Font Server support outline fonts has been proved to be adequate. The time needed for the on-the-fly generation of bitmap from outline is acceptable if a demand loading scheme is employed. This is reflected in both the testing of Font Server and the experiments on the Printer Server.

With the Chinese outline fonts provided by the Chinese Font Server, the support of Chinese font printing on a system level is now possible. The Printer Server is built on the foundation of the Chinese Font Server. The distributed architecture of the Printing System provides much flexibility. Application programs can take advantages of the Printer Server as long as they talk the Printing Services Protocol.

By equipping the printer driver with the new Chinese font printing algorithm, the printing speed of Chinese characters is much improved. Although the algorithm has only been implemented for HP LaserJet, it can indeed be used on any printers that support soft font downloading.

The impact of this research is on Chinese only. The system can be extended to other languages with double-byte representation such as Japanese and Korean[18][23]. The font server can be modified to support Japanese and Korean in the fashion as that of Chinese. Moreover, on the Printer Server, the only thing that is dependent on Chinese is the mapping function of Algorithm Two, which depends on the fact that Big5 code is designed such that frequently used characters usually have a small first byte. Algorithm Two can be adapted to other double-byte languages by simply redefining the mapping function.

The story does not end here. With the release of X Window System release 6, a more flexible demand loading scheme can be used for loading fonts. This is in fact good news to all double-byte fonts users. By using the demand loading scheme, the loading time of double-byte fonts can be shorten significantly. This research demonstrates that the idea of a distributed printing system is indeed plausible. Although with only limited facilities, WYSIWYG applications with support for screen display (from X Window System) and for printer output (from Printing System can still be implemented. It is believed that the idea of a distributed printing system will be employed in a system with more comprehensive functionality.

Appendix A. Printer Driver Class

Below is a generic printer driver class defined in the experimental Printer Server. Any printer driver should be a sub-class of the PrnDriver class. Since all method functions defined below are device-specific, subclass of PrnDriver should overload these functions in order to produce output in the format required by a specified printer.

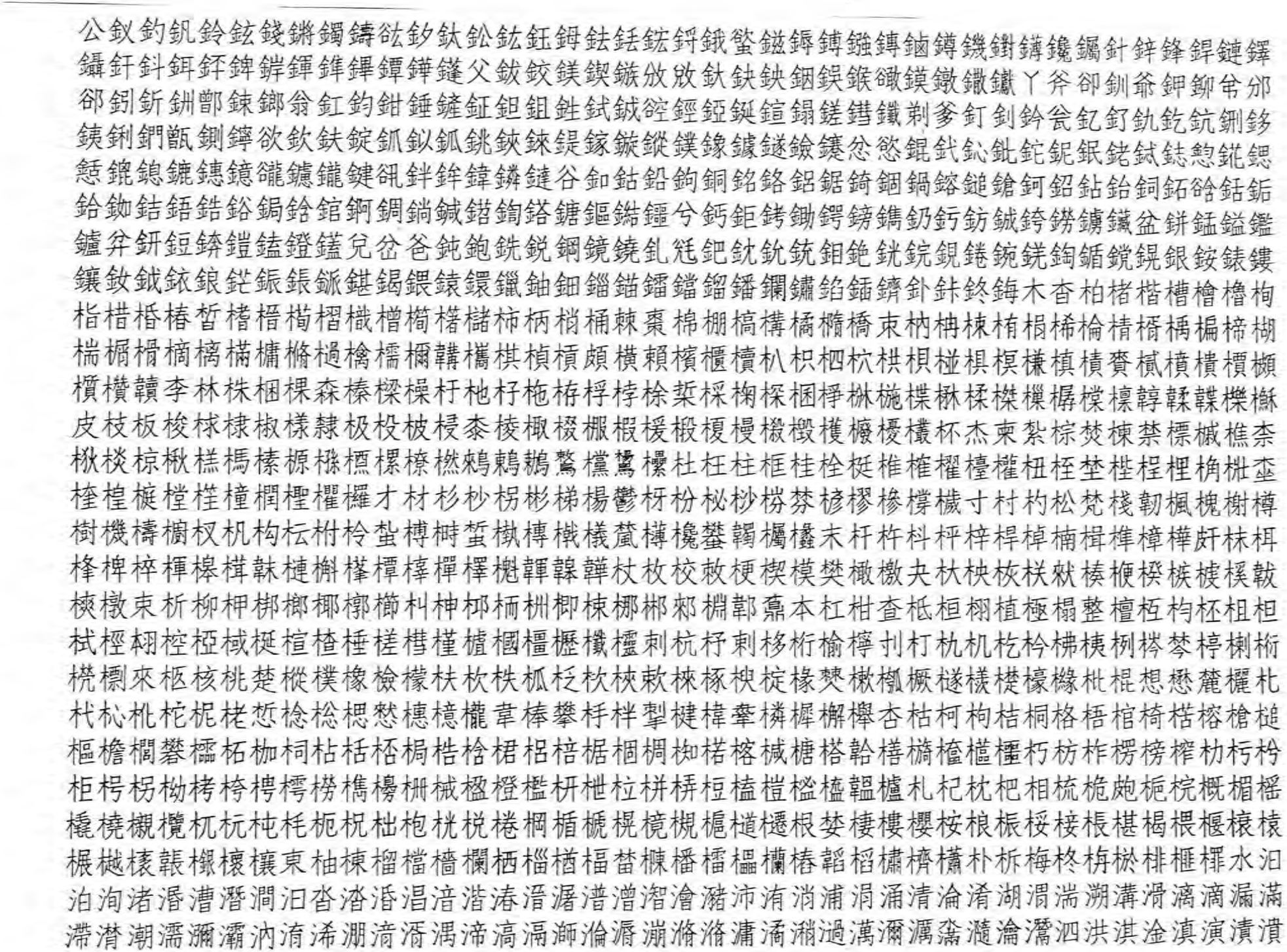
```
class PrnDriver {

  private:

        int setting;
        unsigned short X_res;
        unsigned short Y_res;
        unsigned short p_size;
        unsigned short p_orientation;
        unsigned short num_copy;
        unsigned short pr_Quality;
        unsigned short pr_Duplex;
        int currentfid;
        char *fname;
        unsigned long  mem_available;
        ofstream outf;  // file handle
        DownLoadedFont *head;

  public:

        PrnDriver(char *name);
        virtual ~PrnDriver();
        virtual int PrnInit();
        virtual int PrnGetSetting(unsigned short flag,
                    unsigned short  *xres,
                    unsigned short *yres,unsigned short *ppr,
                    unsigned short *ort,unsigned short *num,
                    unsigned short *dup);

              // get the device mode
        virtual int PrnSetSetting(unsigned short flag,
                    unsigned short  xres,
                    unsigned short yres,unsigned short ppr,
                    unsigned short ort,unsigned short num,
                    unsigned short dup);

        virtual int PrnTextExtent(FnCache *ch,
                    char *name,fsChar2b *str,int str_len,
                    unsigned char id, fsCharInfo **ext_out);

        virtual int PrnDrawString(FnCache *ch,
                    char *name, unsigned char id,
                    fsChar2b *str, int str_len, int x,int y);

        virtual unsigned char PrnOpenFont(FnCache *ch ,
                    char *name);

        virtual unsigned char PrnCloseFont(FnCache *ch ,
                    unsigned char id);
        virtual int PrnEscape(int escCode);
        virtual int SetFont(int, char*);
        virtual int EjectPage();
        virtual int MoveXY (int x, int y);

};
```

Note that the above printer driver supports the soft font downloading algorithm described in Chapter 6. Subclass of PrnDriver could ignore the *DownLoadFont* entry if it does not want to used it to keep track of the information of the downloaded fonts.

Note that the implementations of these method functions are up to the printer driver and are not restricted by the function declarations.

Appendix B. Sample Ouput

頭是中配合改裝的程式
可將各檔案的儲存位置重新編排使每一個檔案都在連續的磁區即使
壓縮後的磁碟機一樣適用原附在今隨
而來並有些微改進是以隱藏式記憶體的觀念將部分主記憶
體或擴充記憶體作為磁碟機資料存取的緩衝區利用記憶體的快速存
取速度彌補磁碟機較慢的存取速度以提高電腦處理效率
多套啓動組合為了配合不同的使用狀況
使用者有時必須反覆地修改系統啓動時兩個關鍵性檔案與
提供多套啓動組合可讓使用者將上述兩個檔案
編寫成特定的選擇式內容附帶地可以指定螢幕顏色內定選擇項與等
候時間系統啓動時即以設定之名稱供使用者選擇而採用對應的啓動組
合予以執行此外當系統啓動後出現時按下
可以不執行這兩個檔案的啓動指令直接進入系統若按下則系統
將逐一顯示各指令詢問使用者是否要執行它最後並詢問
是否執行
附帶特點
此外還有些特點值得一提
使用提示的加強苳漁完備操作方便的線上查詢
救回已刪除檔案迟供暗哨式
追蹤式與標準式等三種層次的刪除保護與
救回方式讓無意中刪除或刪除後悔意的檔案得以盡可能地救回
指令供移動檔案之用尤以同一磁碟機之檔案移動無論大小都
將非常快速同時也可用於變動目錄的名稱
指令可以刪除整個次目錄包括其下的各個檔案與次目錄
指令可應用於成批執行檔執行時依使用者按鍵如
或進行不同的處理
類似的可以詳
盡地檢視系統中各種硬體軟體的相關資訊
配上通訊埠或平行埠的連線
可以讓一台直接使用另一台的硬式磁碟機或報表機
可以有效地管理膝上型電腦或筆記型電腦的電源分配使電池的
延續使用時間得以增長達四分之一
增設的工具箱含防治病毒
硬碟備存
及救回已刪除檔案其作用與
版本完全相同而且相容
系統安裝
推出較低廉的只要有早期的
或即可安裝安裝程序非常簡單
預備兩張與開機磁片同型的空白磁片寫上標籤
將第一片磁片放入或磁碟機對應地鍵入
或
依照螢幕顯示的指示逐步完成安裝步驟
安裝完成後原作業系統的所有檔案將被移到的次目錄
萬一在安裝的過程中出了問題或是安裝後想回復原有作業系統可使
用安裝過程中備儲的磁片解決
不便與缺失
最後且來談談不便與缺失之處
由於進行磁碟機壓縮非常小心對已存有大量檔案的磁碟
機進行壓縮耗時甚久的硬碟在已使用就需一



圓圓曲
圓圓曲
吳偉業
吳偉業
鼎湖當日棄人問，破敵收京下玉關。
鼎湖當日棄人間，破敵收京下玉關。
慟哭六軍俱縞素，沖冠一怒爲紅顏
慟哭六軍俱縞素，沖冠一怒為紅顏
紅顏流落非吾戀，逆賊天亡自荒宴。
紅顏流落非吾戀，逆賊天亡自荒宴。
電掃黃巾定黑山，哭罷君親再相見
電掃黃巾定黑山，哭罷君親再相見
相見初經田竇家，侯門歌舞出如花。
相見初經田竇家，侯門歌舞出如花。
許將戚里空侯伎，等取將軍油壁車
許將戚里空侯伎，等取將軍油壁車
家本姑蘇浣花里，圓圓小字嬌羅綺。
家本姑蘇浣花里，圓圓小字嬌羅綺。
夢向夫差苑里游，宮娥擁入君王起
夢向夫差苑里游，宮娥擁入君王起
前身合是采蓮人，門前一片橫塘水。
前身合是采蓮人，門前一片橫塘水。
橫塘雙槳去如飛，何處豪家強載歸
橫塘雙槳去如飛，何處豪家強載歸
此際豈知非薄命，此時只有淚沾衣。
此際豈知非薄命，此時只有淚沾衣。
熏天意氣連宮掖，明眸皓齒無人惜
熏天意氣連宮掖，明眸皓齒無人惜
奪歸永巷閉良家，教就新聲傾坐客。
奪歸永巷閉良家，教就新聲傾坐客。
坐客飛觴紅日暮，一曲哀弦向誰訴
坐客飛觴紅日暮，一曲哀弦向誰訴
恨殺軍書抵死催，苦留后約將人誤。
恨殺軍書抵死催，苦留后約將人誤。
相約恩深相見難，一朝蟻賊滿長安
相約恩深相見難，一朝蟻賊滿長安
可憐思婦樓頭柳，認作天邊粉絮看！
可憐思婦樓頭柳，認作天邊粉絮看！
遍索綠珠圍內第，強呼絳樹出雕闌
遍索綠珠圍內第，強呼絳樹出雕闌
若非壯士全師勝，爭得蛾眉匹馬還？
若非壯士全師勝，爭得蛾眉匹馬還？
蠟炬迎來在戰場，啼妝滿面殘紅印。
蠟炬迎來在戰場，啼妝滿面殘紅印。
專征簫鼓出秦川，金牛道上車千乘
專征簫鼓出秦川，金牛道上車千乘
斜谷云深起畫樓，散關月落開妝鏡。
斜谷云深起畫樓，散關月落開妝鏡。
教曲伎師憐尚在，浣紗女伴憶同行。
教曲伎師憐尚在，浣紗女伴憶同行。

X Window System Printing Extension

圓圓曲
吳偉業
鼎湖當日棄人間，破敵收京下玉關。
慟哭六軍俱縞素，沖冠一怒爲紅顏
紅顏流落非吾戀，逆賊天亡自荒宴。
電掃黃巾定黑山，哭罷君親再相見
相見初經田竇家，侯門歌舞出如花。
許將戚里空侯伎，等取將軍油壁車
家本姑蘇浣花里，圓圓小字嬌羅綺。
夢向夫差苑里游，宮娥擁入君王起

X Window System Printing Extension

圓圓曲
吳偉業
鼎湖當日棄人間，破敵收京下玉關。
慟哭六軍俱縞素，沖冠一怒為紅顏
紅顏流落非吾戀，逆賊天亡自荒宴。
電掃黃巾定黑山，哭罷君親再相見
相見初經田竇家，侯門歌舞出如花。
許將戚里空侯伎，等取將軍油壁車
家本姑蘇浣花里，圓圓小字嬌羅綺。
夢向夫差苑里游，宮娥擁入君王起

X Window System Printing Extension

圓圓曲

吳偉業

鼎湖當日棄人間，破敵收京下玉關。
慟哭六軍俱縞素，沖冠一怒為紅顏
紅顏流落非吾戀，逆賊天亡自荒宴。
電掃黃巾定黑山，哭罷君親再相見
相見初經田竇家，侯門歌舞出如花。
許將戚里空侯伎，等取將軍油壁車
家本姑蘇浣花里，圓圓小字嬌羅綺。
夢向夫差苑里游，宮娥擁入君王起

X Window System Printing Extension

圓圓曲

吳偉業

鼎湖當日棄人間，破敵收京下玉關。
慟哭六軍俱縞素，沖冠一怒為紅顏
紅顏流落非吾戀，逆賊天亡自荒宴。
電掃黃巾定黑山，哭罷君親再相見
相見初經田竇家，侯門歌舞出如花。
許將戚里空侯伎，等取將軍油壁車
家本姑蘇浣花里，圓圓小字嬌羅綺。
夢向夫差苑里游，宮娥擁入君王起

X Window System Printing Extension

圓圓曲

吳偉業

鼎湖當日棄人間，破敵收京下玉關。
慟哭六軍俱縞素，沖冠一怒為紅顏
紅顏流落非吾戀，逆賊天亡自荒宴。
電掃黃巾定黑山，哭罷君親再相見
相見初經田竇家，侯門歌舞出如花。
許將戚里空侯伎，等取將軍油壁車
家本姑蘇浣花里，圓圓小字嬌羅綺。
夢向夫差苑里游，宮娥擁入君王起

圓圓曲
吳偉業

鼎湖當日棄人間，破敵收京下玉關。
慟哭六軍俱縞素，沖冠一怒爲紅顏
紅顏流落非吾戀，逆賊天亡自荒宴。
電掃黃巾定黑山，哭罷君親再相見
相見初經田竇家，侯門歌舞出如花。
許將戚里空侯伎，等取將軍油壁車
家本姑蘇浣花里，圓圓小字嬌羅綺。
夢向夫差苑里游，宮娥擁入君王起
前身合是采蓮人，門前一片橫塘水。
橫塘雙槳去如飛，何處豪家強載歸
此際豈知非薄命，此時只有淚沾衣。
熏天意氣連宮掖，明眸皓齒無人惜
奪歸永巷閉良家，教就新聲傾坐客。
坐客飛觴紅日暮，一曲哀弦向誰訴
恨殺軍書抵死催，苦留后約將人誤。
相約恩深相見難，一朝蟻賊滿長安
可憐思婦樓頭柳，認作天邊粉絮看！
遍索綠珠圍內第，強呼絳樹出雕闌
若非壯士全師勝，爭得蛾眉匹馬還？
蠟炬迎來在戰場，啼妝滿面殘紅印。
專征簫鼓出秦川，金牛道上車千乘
斜谷云深起畫樓，散關月落開妝鏡。
教曲伎師憐尚在，浣紗女伴憶同行。

References

[1] Apple Computer, Inside Macintosh, Volume II, Addison-Wesley, 1985

[2] Bennett Steven J. & Randall Peter G., The LaserJet III Companion, Brady Publishing, 1991.

[3] Bigelow Charles, Font Design For Personal Workstations, Byte January 1985, pp. 255-6

[4] Chen C. K. and Moon Y. S., Modification of X Window System Font Server to Support Chinese Outline Fonts, Proceedings of International Conference on Chinese Computing' 94, pp. 249-257

[5] Coplien James, Advanced C++, Programming styles and idioms, Addison Wesley, 1992.

[6] Debry Roger and Griffee Alan W, Management of multiple font technologies in a distributed system environment, Raster Imaging and Digital Typography, 1988.

[7] Finseth Craig A., The Craft of Text Editing Emacs for the Modern World, Springer-Verlag,1991.

[8] Foley James & Dam Andries van, Computer Graphics Principles and Practice, Addison Wesley, 1990.

[9] Flowers Jim, X Logical Font Description Conventions Version 1.5 Public Review Draft, X Consortium Inc.

[10] Fuchs D.R. and Knuth D.E., Optimal Prepaging and Font Caching, ACM Trans. on Prog. Lang. and Sys., Vol.7,1, 1985 pp. 62-79

[11] Fulton Jim, The X Font Service Protocol ver. 1.0, Network Computing Devices, Inc., 1991.

[12] Griffee A. W. and Casey C.A., An introduction to typographic fonts and digital font resources, IBM System Journal, Vol 27, No 2, 1988.

[13] Hersch Roger D. Introduction to font rasterization, Raster Image and Digital Typography Proceedings of the International Conference, Ecole Poytechnique Federale Lausanne, 1989. pp1-13.

[14] Heller Martin, Strengthening the ties that bind, Windows, Vol 4, No.3, 1993. pp 129-134.

[15] Hewlett Packard, PCL Printer Language Technical Quick Reference Guide, Hewlett Packard,1990.

[16] Huang Jack Kai-tung, The Input and Output of Chinese and Japanese Characters, IEEE Computer, January 85, pp. 18-24

[17] Israel Elias and Fortune Erik, The X Window System Server, Digital Press, 1992.

[18] Iwamoto Hitoshi and Cai Rong, Handling Several Character Sets on the X Window System, Center of International Cooperation for Computerization.

[19] Jones Oliver, Introduction to The X Window System, Prentice Hall, 1989.

[20] Kawabata A. and Marinescu Dan C., Font Cache Design Issues in a Distributed Electronic Publishing System for Japanese Language Documents, Oki Technical Review Vol. 58.

[21] Lemeke Dave, Font server implementation overview, Network Computing Devices, Inc,. 1991.

[22] Martin James, Principles of Object-Oriented Analysis and Design, Prentice Hall, 1993.

[23] Matsuda Ryouichi, Processing Information in Japanese, IEEE Computer, January 85, pp. 27-34

[24] Meyer Bertrand, Object-Oriented Software Construction, Prentice Hall, 1988.

[25] Meyer Brian and Doner Chris, Programmer's Introduction to Windows 3.1, Tech Publication Pte. Ltd, 1992.

[26] McGilton Henry and Campione Mary, PostScript by Example, Addison-Wesley Publishing Company, 1992.

[27] Microsoft Corporation, Microsoft Windows Device Driver Kit, Device Driver Adaptation Guide, 1992.

[28] Microsoft Corporation, Microsoft Windows Device Driver Kit, Printer and Fonts Kit, 1992.

[29] Microsoft Corporation, TrueType Font Files ver. 1.00, Microsoft Corporation, 1992.

[30] Moore Geroge, An Introduction to Digital Typography using TrueType, Microsoft Corporation.

[31] Nye Adrian, Xlib Programming Manual, O'Reilly & Associates, Inc., 1992.

[32] Packard Keith & Lemke David, The X Font Library, X Consortium, Inc., 1991.

[33] Rubinstèin Richard, Digital Typography: An Introduction to Type and Composition for Computer System Design, Addison-Wesley, 1988.

[34] Scheifler Robert W, X Window System Protocol MIT X Consortium Standard, X Version 11 Release 5, X Consortium Inc.

[35] Scheifler Robert W. and Gettys James, X Window System, The Complete Reference to Xlib, X Protocol, ICCCM, XLFD, Digital Press, 1992.

[36] Scheifler Robert W. and Gettys James, The X Window System, ACM Transactions on Graphics, Vol. 5, No. 2, 1986, pp 79-109.

[37] Stevens W. Richard, UNIX Network Programming, Prentice Hall,1990.

[38] Stroustrup Bjarne, The C++ Programming Language, Addison Wesley, 1991.

[39] Tanenbaum Andrew S., Computer Networks, Prentice Hall, 1989.

[40] Tanenbaum Andrew S., Modern Operating System, Prentice Hall, 1992.

[41] X Consortium Inc., Font Server sample implementation.

[42] Weise David and Adler Dennis, TrueType and Microsoft Windows Version 3.1, Microsoft Corporation.

[43] Young Douglas A., X Window Systems Programming and Application with Xt, Prentice Hall, 1989.

[44] Young Douglas A., Object-Oriented Programming with C++ and OSF/Motif, Prentice Hall, 1992.

[45] 倚天資訊股份有限公司,倚天中文系統,松崗電腦圖書資料有限公司

[46] 呂學鈿,中文字形環境 Microsoft Window 3.0中文版,倚天雜誌 Aug, 1991.

[47] 韓世昌,剖析 Microsoft Windows 3.0 中文版,倚天雜誌 , Aug, 1991.

[48] 貝貴琴,張學濤,漢字頻度統計-速成識讀優選表, 電子工業出版社, 1988.

[49] 賴洋助, 林任烈,UNIX系統與X視窗之中文化,工業技術研究院

電腦與通訊工業研究所 系統軟體部

[50] 趙敏 安博, Windows窗口環境下的漢字信息管理系統,北京天晨新技術開發中心