Computer Science and Systems Analysis

Computer Science and Systems Analysis

Technical Reports

Miami University *Year* 1992

User Interface Implementation for Network License Management System

Jianli Jiang
Miami University, commons-admin@lib.muohio.edu

MIAMI UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE & SYSTEMS ANALYSIS

TECHNICAL REPORT: MU-SEAS-CSA-1992-012

User Interface Implementation for Network License Management System

Jianli Jiang

School of Engineering & Applied Science | Oxford, Ohio 45056 | 513-529-5928

User Interface Implementation for Network License Management System

by

Jianli Jiang
Systems Analysis Department
Miami University
Oxford, Ohio 45056

Working Paper #92-012 08/92

User Interface Implementation for Network License Management System

Presented in Partial Fulfillment of the Requirements
for the Degree of
Master of Science in Systems Analysis
Graduate School of Miami University

Jianli Jiang
Miami University
August 1992

Advisor: Prof. Douglas Troy

Reading Committee: Dr. Alton Sanders
Dr. James Kiper
Dr. Don Byrkett

Table of Contents

Abstract 2

1. Introduction 3
2. Information and Terminology 4
3. Design of the Network License Management System 8
   3.1 Software Modules 8
   3.2 Operation of NLMS 10
4. Requirements for the User Interface and Management Reports 12
   4.1 Requirements 12
   4.2 User Interface Design 13
   4.3 Software Usage Reports 18
5. Microsoft Windows Implementation 19
   5.1 Interface Description 19
   5.2 Programming with MS Windows 20
   5.3 MS Windows Implementation 21
   5.4 Incompatibility between MS Windows and SUN PC-NFS 22
6. X Windows Implementation 23
   6.1 X Windows System 23
   6.2 Programming with X Windows 23
   6.3 X Windows Implementation 24
7. Management Reports 26
   7.1 Types of Reports 26
   7.2 Implementation 26
8. Bugs Discovered and Corrected 28
9. Comparison of Programming in Microsoft Windows and X Windows 28
   9.1 Concepts 29
   9.2 Programming 29
10. Conclusion 30
   10.1 Interactive Design Tools 30
   10.2 Improvements 31

References 33

Appendix 1: An example of Management Reports 34
Appendix 2: Source Code Listings 35

Table of Contents

Abstract	2
1. Introduction	3
2. Information and Terminology	4
3. Design of the Network License Management System	8
3.1 Software Modules	8
3.2 Operation of NLMS	10
4. Requirements for the User Interface and Management Reports	12
4.1 Requirements	12
4.2 User Interface Design	13
4.3 Software Usage Reports	18
5. Microsoft Windows Implementation	19
5.1 Interface Description	19
5.2 Programming with MS Windows	20
5.3 MS Windows Implementation	21
5.4 Incompatibility between MS Windows and SUN PC-NFS	22
6. X Windows Implementation	23
6.1 X Windows System	23
6.2 Programming with X Windows	23
6.3 X Windows Implementation	24
7. Management Reports	26
7.1 Types of Reports	26
7.2 Implementation	26
8. Bugs Discovered and Corrected	28
9. Comparison of Programming in Microsoft Windows and X Windows	28
9.1 Concepts	29
9.2 Programming	29
10. Conclusion	30
10.1 Interactive Design Tools	30
10.2 Improvements	31
References	33
Appendix 1: An example of Management Reports	34
Appendix 2: Source Code Listings	35

Abstract

This paper describes a project to understand and enhance a distributed application – the Applied Science Microlab Network License Management System(NLMS), and to design and implement an improved user interface for that system. The NLMS utilizes a client–server architecture, the TCP/IP network protocol suite, and the Remote Procedure Call (RPC) facility for the program to interface to the network. The new user interface is based on X Windows/Motif for UNIX client and Microsoft Windows for PC client. In addition, management reports are added to the system and expected to provide the package usage statistics to aid in future software purchase plans.

The goals of my project are to do some developmental work in the UNIX environment; to understand more about network programming; to learn how to write distributed applications; and to learn to programming X Windows/Motif and Microsoft Windows. In my opinion, I have accomplished these goals.

1. Introduction

The Applied Science Microlab is a computing laboratory in the School of Applied Science at Miami University that consists of approximately 80 IBM compatible personal computers, running the PC-DOS or MS-DOS operating system. The PCs are networked, and connected to an IBM RISC/6000 file server running IBM's UNIX operating system called AIX. Most of the lab's PC application software is stored on the file server. When a student desires to use a particular application on a PC, he or she selects the application from a menu on the PC which in turn causes the relevant application to be downloaded from the file server and executed on the PC. In general, the student is unaware of the file server.

Many of the PC application software packages are single-user licensed, and the microlab does not own enough single-user licenses to serve every PC in the lab. For example, the microlab may own 30 single-user licenses for word processor XYZ. Thus, to be compliant with the license agreements, no more than 30 students at a time should be permitted to use the XYZ word processor concurrently.

The Applied Science Microlab Network License Management System (NLMS) is a distributed application, developed by Systems Analysis students, that permits enforcement and management of license agreements[Troy, 1991]. NLMS consists of programs that run on the PCs and the file server, and permits counting and tracking of the number of each application under execution at a given time. The NLMS also maintains a data base storing the information of the maximum number of copies licensed for each application. A portion of the NLMS that executes on the PCs is used to query the database part, that runs on the server, to determine if a given PC application may be downloaded and executed. If the application cannot be downloaded because of an insufficient number of available licenses, then the user is informed to try later. If a license is available, the application is downloaded and executed on the PC.

Another portion of the NLMS permits the network manager to manage the database on the server. For example, the manager could examine the number of applications currently executing, change the number of licenses available, or reinitialize the entire database. Additionally, the NLMS collects data about the usage of each software package. In the original version of NLMS, written prior to this project, this component of the NLMS had a

simple command-line user interface, and, although the data existed to provide many useful management reports on software usage, this part of the system was not implemented. These were major weaknesses of the NLMS prior to my project.

The goals of this project are to:

1. Determine the requirements for an improved and user friendly interface for the management portion of the Applied Science Microlab's Network License Management System;
2. Determine the requirements for management reporting;
3. Implement the improved interface on Personal Computers using MS Windows and on UNIX Workstations using X Windows;
4. Implement the management reporting system; and
5. Summarize and compare the MS Windows and X Windows programming efforts.

The last goal, although not directly related to the NLMS, was an important goal in furthering my understanding of different programming environments for user interfaces.

The remainder of the report is organized as follows. Section 2 presents information and terminology on distributed processing and networking that is used in the remainder of the report. Section 3 describes the design of the NLMS. Section 4 describes the requirements for the modifications and improvements to the NLMS. Sections 5 and 6 describes the Microsoft Windows and X Windows systems that were used to implement the improved user interface. Section 7 is a description of the management reporting that was implemented. Section 8 describes some bugs in the original NLMS that were discovered and corrected. Section 9 presents a comparison of programming in MS Windows and X Windows, and Section 10 concludes the report. Appendix 1 contains an example of the management reports, and Appendix 2 is a complete listing of the source code for the revised NLMS system.

2. Information and Terminology

The NLMS is a distributed processing application designed using the client-server model. The client-server model is used to describe a network system in which one or more processes provide services across the network to one or more other processes, and commu-

nication is in the form of request/reply pairs initiated by the client. The term "server" refers to any process that provides services on request and the term "client" refers to any process that uses the services offered by a server[Ames, 1991]. In our NLMS, the server keeps the license information of software packages and their usage information, provides the services of checking the availability of software packages, and answering the queries from clients. The client presents all the functions available in a nice user interface, through which the user can make different kinds of requests to the remote server and get the reply. One centralized server accepts requests from and responds to all the clients.

The Applied Science Network is an Ethernet local area network that supports the following protocols:

1. TCP/IP
2. Sun Remote Procedure Call (RPC)
3. Sun eXternal Data Representation (XDR)
4. Network File System
5. X Windows

TCP/IP (Transmission Control Protocol/Internet Protocol), funded by Defense Advanced Research Projects Agency, was developed to interconnect different subnetworks of various architectures and make them function as a coordinated unit[Cypser, 1991]. TCP/IP Internet Protocol Suite, commonly referred to as TCP/IP, consists of a set of network standards that specify the details of how computers communicate and a set of conventions for interconnecting networks and routing traffic[Comer, 1991]. Internet provides two broad types of services that can be used by any application program: connectionless package delivery service (User Datagram Protocol) and reliable stream transport service (Transmission Control Protocol). TCP/IP protocols define the unit of data transmission, called a datagram, and specify how to transmit datagrams on a particular network[Comer, 1991].

Electronic mail, file transfer and remote login are some of the popular and widespread Internet application services. Unlike other underlying protocols, TCP/IP protocols make these applications more reliable, because the machines at each end (sender and receiver) are involved in the communication directly[Comer, 1991].

In fact, TCP/IP has become one of the standard protocols at the Transport/Network layer in the OSI seven layer model. The relation between TCP/IP and OSI Open System Interconnection reference model is shown in Table 2.1[*PC–NFS Programmer's Toolkit Manual*, 1987].

7 Application	NFS FTP	YP rcp	TELNET rsh
6 Presentation	XDR		
5 Session	RPC		
4 Transport	TCP	UDP	
3 Network	IP (internetwork)		
2 Data Link	Ethernet	IEEE 802.2	
1 Physical	Ethernet	IEEE 802.3	

Table 2.1 OSI Seven Layer Model and TCP/IP Protocol

Remote Procedure Call (RPC) specifies a particular communication style between client and server, which allows a client to call a procedure that a remote server executes. To obtain a service, a client issues a request in the form of RPC. RPC is a paradigm in which a function call, although appearing as a local call, actually results in a network request/reply interaction [Ames, 1991]. Program_number, procedure_number and version_number are used to identify the service that the client desires. The server registers its services with the operating system using these numbers, and the client passes these numbers as parameters when making a RPC call. Figure 2.1 shows what actually happens when a client makes a RPC call[*PC–NFS Programmer's Toolkit Manual*, 1987]. Several RPC protocols have been designed. The one used in our system is based on Sun Microsystems RPC.

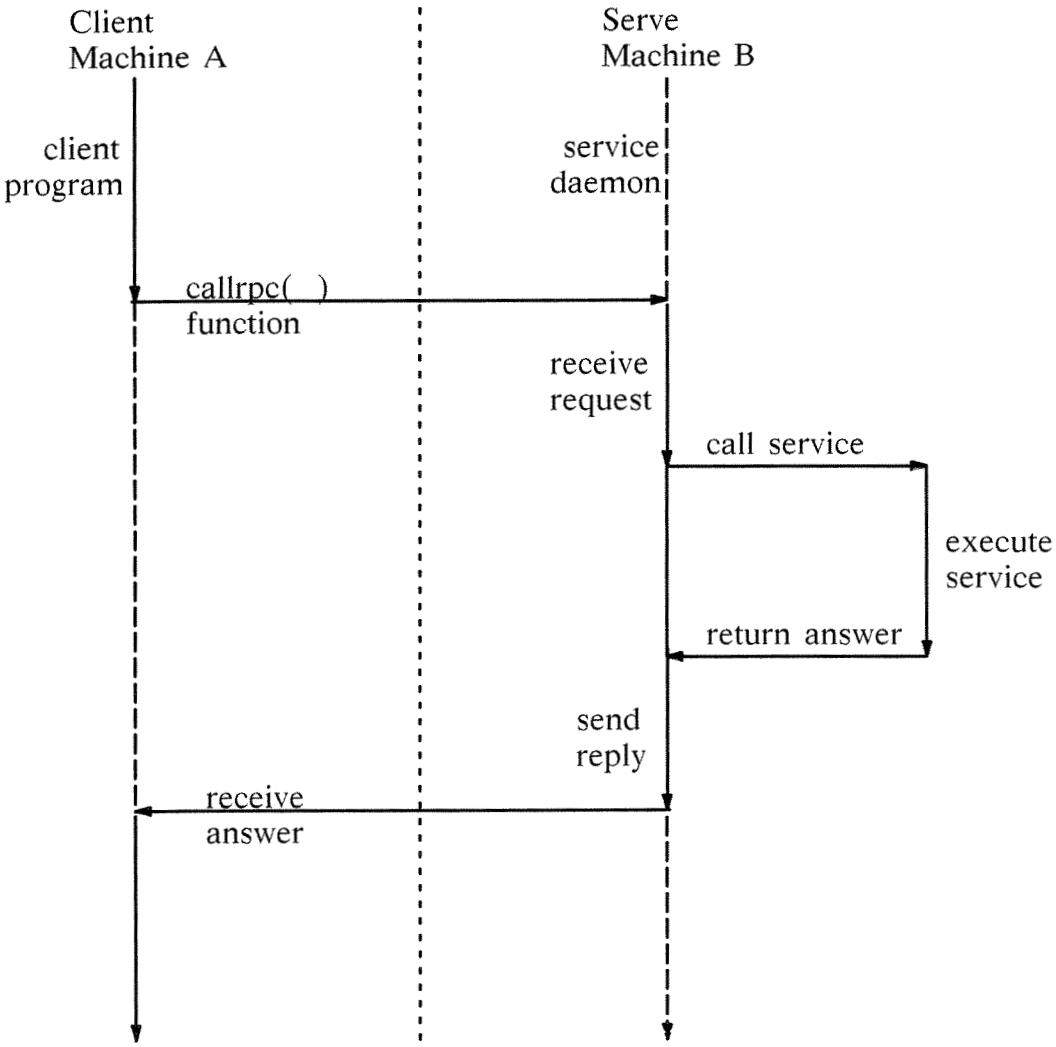


Figure 2.1 Network Communication Mechanism with the Remote Procedure Call

External Data Representation (XDR) specifies standard representations for basic data types, structures and unions. Before data is sent to the network, it is translated from the local host's representation to the standard representation(this process is called encoding); and when it arrives at the receiving host, it is translated into the data representation that receiving site uses(this process is called decoding)[*PC–NFS Programmer's Toolkit Manual*, 1987]. Constructor primitives are the services provided by XDR for encoding and decoding, which allow programmers to translate basic data types as well as user defined complex data types. XDR is important because it guarantees that data will be interpreted correctly in a heterogeneous environment. RPC uses XDR to translate input and output data.

As shown in Table 2.1, the OSI seven layer model, NFS is at the top application layer. NFS (Network File System) is a facility for sharing resources, including files and printers, among network users[*PC-NFS Programmer's Toolkit Manual*, 1987]. SUN's RPC and XDR have been developed and used during the process of implementing SUN's PC-NFS. In NFS, file systems can be mounted from remote machines, and its location is transparent to the users. In our Applied Science Microlab, most of the PC software packages are stored on the server machine, as mentioned in the Introduction. Using PC-NFS, each PC can mount that portion of a filesystem, and read or write it as if it were a local disk.

The X Window System is a network-transparent window system that was designed at MIT in conjunction with DEC. The X Window System Protocol describes the precise semantics of the X11 protocol specification, that is the exact behavior of X[*X Protocol*, 1988]. One important difference between X and many other window systems is that X does not define any particular user interface style. X provides a flexible set of primitive window operations, and a device-independent layer that serves as a base for a variety of interface styles.

The architecture of the X Window System is based on a client-server model. A single process, known as the server, running on a workstation, is responsible for all input and output devices, including keyboard, mouse, and display. Any application that uses the facilities provided by the X server is known as a client. A client communicates with the X server via a network connection using an asynchronous byte-stream protocol[*Xlib Library*, 1988]. X supports many network protocols, such as TCP/IP and DECnet.

Xlib is a low level C subroutine library that application programmers use to interface with X Window System and to write X-based programs[*Xlib Library*, 1988].

The next section describes the overall design of NLMS.

3. Design of the Existing Network License Management System

3.1 Software Modules

The NLMS system is designed based upon the client-server architecture and it can operate on any operating system that supports the Remote Procedure Call, XDR and TCP/IP protocols[Troy, 1991]. For example, the software has been tested on IBM RISC/6000 (AIX), NeXT (Mach), and DEC VAX (Ultrix), and for PCs, Sun PC-NFS (clients only).

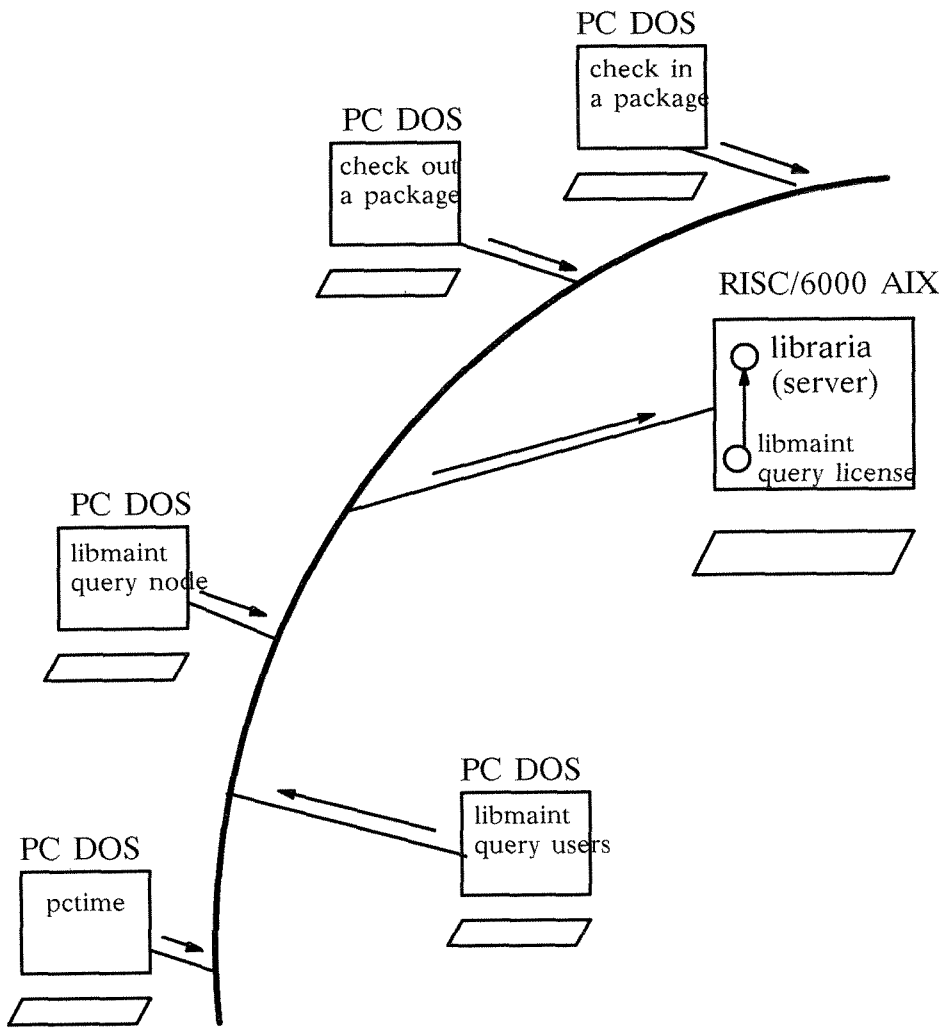


Figure 3.1 The architecture of the NLMS system

Figure 3.1 shows the architecture of the NLMS. As can seen, all of the client programs running on UNIX or PCs send their requests through the network to the server and the server processes each request and sends back a reply.

The programs that made up NLMS prior to my project were:

libraria -- server program that runs on UNIX only. It monitors the license information, counts and tracks the packages being used, and processes the requests from client program "libmaint", "check", or "pctime."

libmaint -- client program that can run on both UNIX and PCs. It is the management portion of the client program, including functions such as query nodes, query license, query time, reset the database, shutdown server pro-

gram "libraria", restart the server, etc.

check -- client program that runs on UNIX and PCs. It is used by a computer node to check out a software package, or check in the software package running on that node.

pctime -- client program runs on PCs. Its purpose is to set the PC's clock to the time of server. This program is used in the program "Sign-in" as well.

3.2 Operation of NLMS

The complete user interface to the existing NLMS is described in the users' guide *"Librarian, A Multi-User License Manager"* [Troy, 1991]. This section presents an overview of the operation of the existing system.

The NLMS server, which is called "libraria", is a program that maintains a database which stores each software package's name and the number of copies purchased. For each software package, the "libraria" keeps track of every computer node which is using each software package. This server program runs as a daemon (background process) on the server, waiting to receive requests from client machines on the network, which are typically PCs but could be a processor on the file server as well. When a request arrives, the server processes the message, and sends back the result. The type of requests include:

1. "check in" or "check out" a software package from client program "check",
2. query the time and date from "pctime",
3. query the usage of a particular software package,
4. query the status of a client machine,
5. change the number of licensed copies of a software package,
6. restart the server program.

Before using a software package, the user needs to obtain permission from the NLMS server by sending a request "check out". Upon receiving a request from a software user, the server checks the database to either reject the request, if the number of users using the software has already reached the maximum number of copies licensed; or grant the request, if the number of users is fewer than the number of copies of the software licensed. When a computer node ceases using a software package, it will inform the NLMS server by sending a

message "check in", and the server will update the software usage information accordingly.

The NLMS server also has many provisions for the management of the network facility. In addition to obtaining the software usage information of the network, an authorized network manager is able to maintain and monitor the network license configuration by sending appropriate requests to the NLMS server.

An important fact to note is that all of the NLMS programs, except for the server (libraria), can be compiled to run on either the PC-DOS clients or the UNIX server. To program the MS-DOS programs, a library of networking functions is required. SUN PC-NFS Programmer's toolkit is a library of C functions used to support RPC-based clients[*PC-NFS Programmer's Toolkit Manual*, 1987]. It provides a relatively easy way to write distributed applications. The AIX operating system, as well as other versions of UNIX such as Ultrix from DEC and Mach from NeXT, comes standard with the RPC library.

The NLMS server "libraria" generates a log, or audit trail, for every transaction that it processed. The names of these files are of the format Month_Date_Weekday.audit. For instance, the audit file for July 4, 1992 is named "Jul_04_Sat.audit."

The audit log shows the time, the client request, the IP address of the client node issuing the request, and the package or node (if applicable) involved. Here are some sample lines stored in an audit file.

01:00:12	SET_MAX	000.000.000.000	Automenu	0
01.00.12	SET_MAX	000.000.000.000	lotus	35
01.00.12	SET_MAX	000.000.000.000	wp51	40
00.00.12	SET_MAX	000.000.000.000	qc	25
09.12.45	CHECK_OUT	134.053.002.051	wp51	
09.24.34	CHECK_OUT	134.053.002.056	qc	
10.12.41	QTIME	134.053.002.161		
11.09.23	CHECK_IN	134.053.002.056	qc	
11.09.23	CHECK_OUT	134.053.002.056	Automenu	
19.12.41	QPACKAGE	134.053.002.067	lotus	
19.12.41	QNODE	134.053.002.230	134.053.002.056	
21.09.23	CHECK_IN	134.053.002.051	wp51	

If a package is SET_MAX to 0, it means that there is no license limit on it.

In the existing system, it is very inconvenient for the network manager to invoke a query or maintenance command using the old libmaint program. In order for the manager to check the usage of a specific software package, one needs to issue a query using the precise name of the software package, as it is stored in the software license data file. Any misspelling of the software package name results in an error message or incorrect response. For example, the following command checks the usage of Microsoft Quick C:

```
libmaint apsrisc query package qc
```

If qc had been spelled as quickC or QC, the command would fail.

It is even more difficult to query a computer node status, since the user has to remember the exact internet address of the machine on the network. For instance, command:

```
libmaint apsrisc query node 134.53.3.56
```

checks the status of the machine whose internet address is 134.53.3.56. Even with exceptional memory, a person might find it difficult to remember more than 10 such machine addresses.

My major goal is to design a user-friendly interface for the program "libmaint." Menus and Dialog boxes are used to improve the quality of the user interface. The manager only needs to select an appropriate entry on the display menu to execute a desired query and, hence, is free from remembering the software packages' names and machine addresses. X Windows for the UNIX version and Microsoft Windows for the PC-DOS version of "libmaint" are chosen to implement the Menus and Dialog boxes.

The second major goal is to implement functions to produce management reports that will aid the Applied Science Microlab manager in monitoring software usage and in making future software purchase decisions.

4. Requirements for the User Interface and Management Reports

4.1 General Requirements

To determine the requirements for the improved user interface for the management portion of NLMS and for the management reports, I interviewed the original designer of the NLMS, Mr. D. Troy; the microlab manager, Ms. S. Baker; and the department chair Dr. A.

Sanders. The following list summarizes their suggestions:

1. The NLMS should continue to run on either DOS or UNIX machines.
2. The maintenance portion should have a menu-driven and window-based user interface.
3. The management reports should be implemented. The reports should provide the information on the maximum number of copies of software packages being used concurrently; the maximum times each package has been checked out during a day; and the accumulative length of time each package has been in use during a day as well.
4. The time interval of the usage reports should be flexible and should have daily, weekly, monthly or semester time frames.
5. The report should take a reasonable amount of time to produce.

4.2 User Interface

User interface has now been recognized as a critically important part of system design. In order to implement a user-friendly interface for the system, I researched some guidelines to good interactive system design. The key goals in user-interface design are to increase the speed of learning, increase the speed of use, and reduce the error rate[Foley, 1990]. It is recommended that the interface present an easy way for the user to understand how to accomplish what he or she wants and the interface be consistent throughout the program[Brown, 1989]. *What you see is what you get* is fundamental to interactive graphics[Foley, 1990].

Following these good interface principles, menus and dialog boxes are introduced to improve user friendliness. Menus allow the user to browse the functions the system provides; and since no typing is involved, the possibility of error occurrence is eliminated. Dialog boxes for input and output enable the system interface to be consistent. It was determined that the main menu should have the following choices: "Query", "Maintenance", "Report", "Special" and "Help."

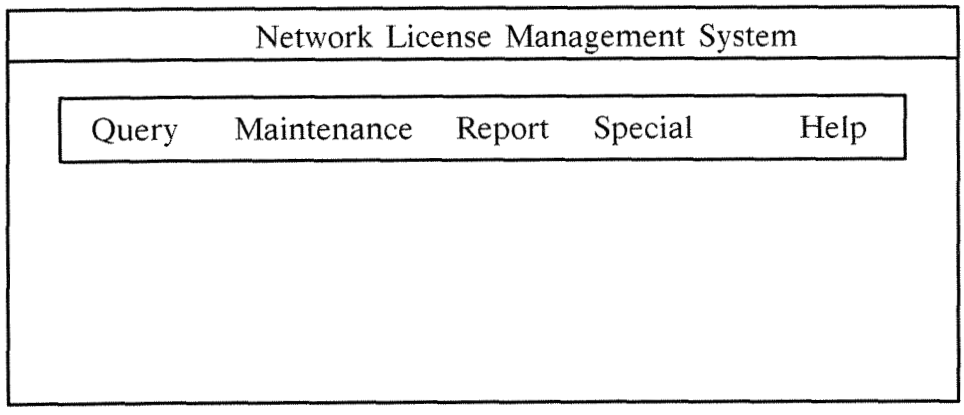


Figure 4.1 Main Window with top level menu

A corresponding pull down menu with detailed options will be displayed for each entry selected. They are shown in the following figures.

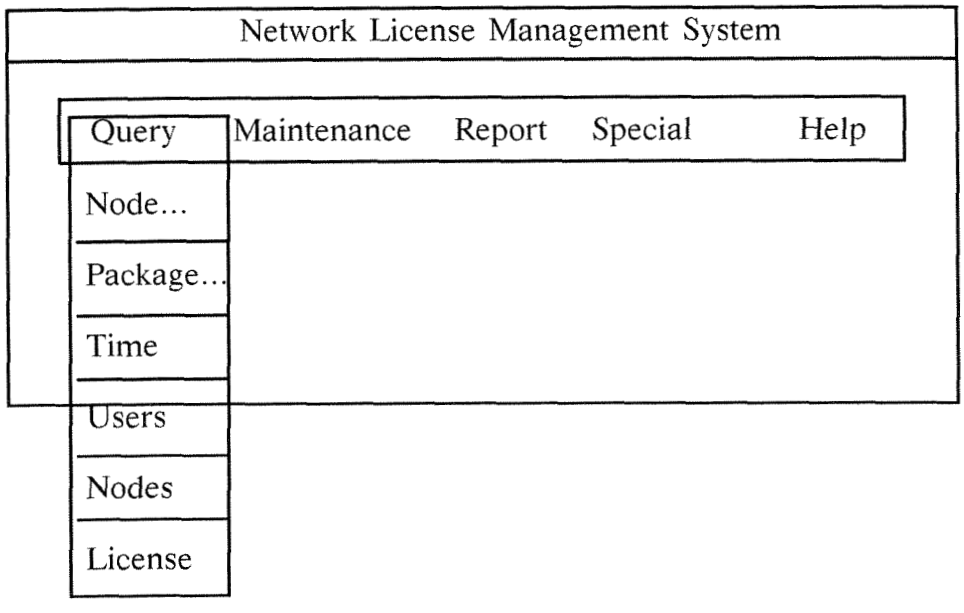


Figure 4.2 When "Query" is selected

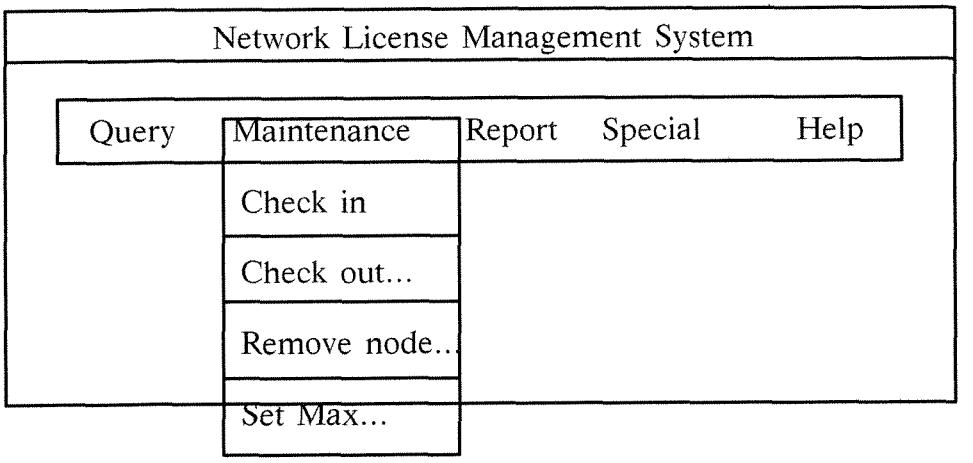


Figure 4.3 When "Maintenance" is selected

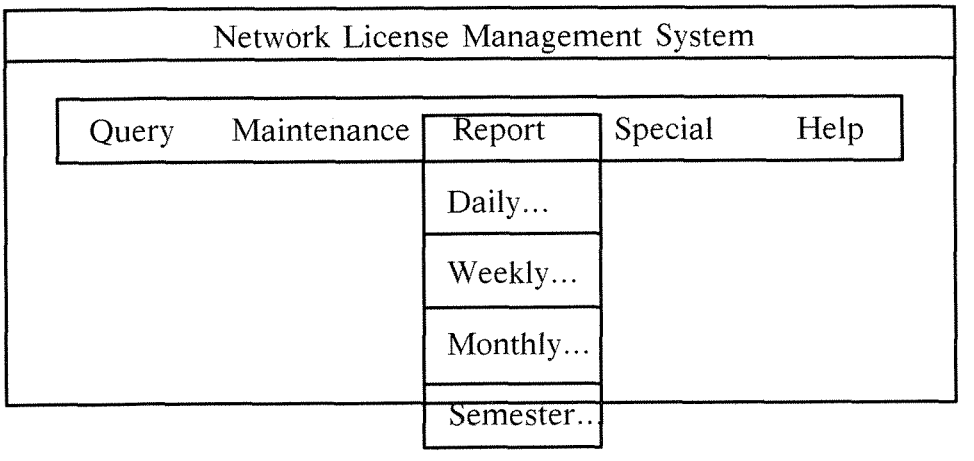


Figure 4.4 When "Report" is selected

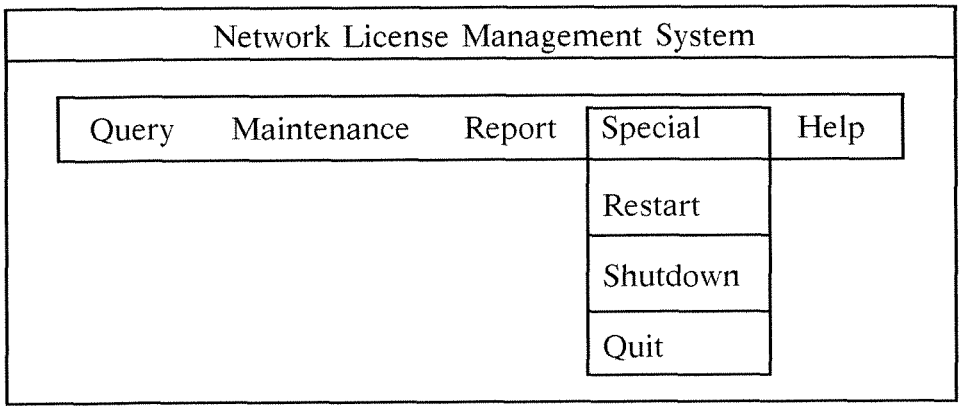


Figure 4.5 When "Special" is selected

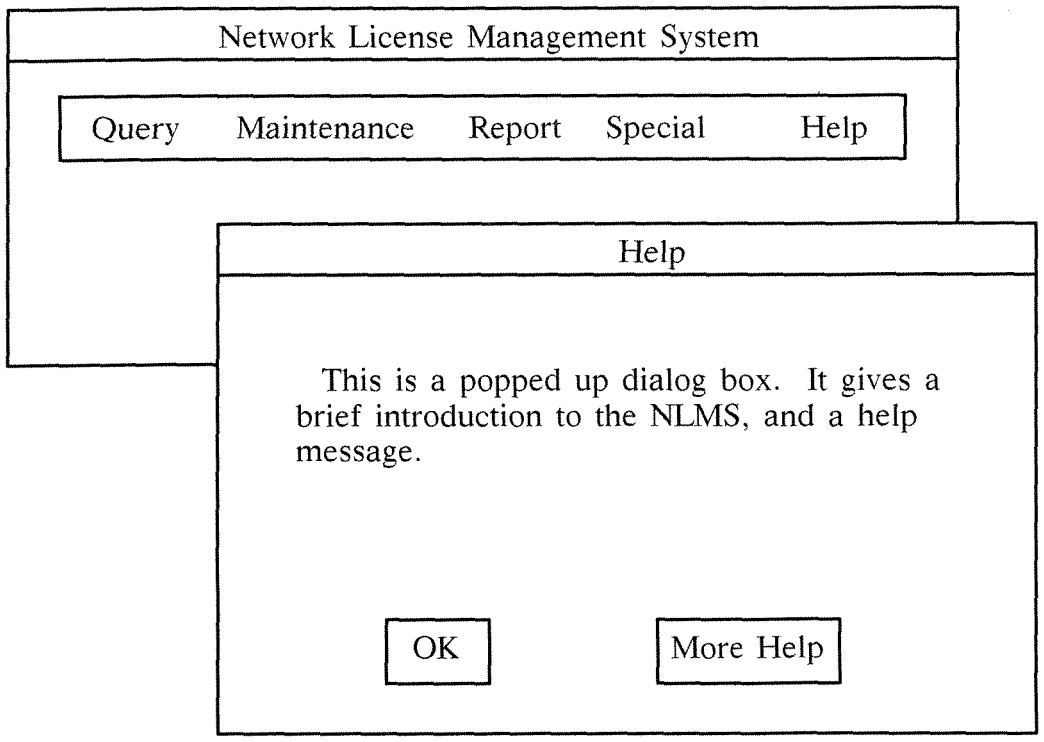


Figure 4.6 When "Help" is selected

There exist two types of menu selections. If a selection not ending with "..." such as "Users" under "Query" is clicked, the corresponding function is executed, and the result is displayed on a popup dialog box. If a selection ending with "..." such as "Node..." under "Query" is clicked, a dialog box will be popped up for further user input. Only after sufficient input is provided, can the desired function be executed.

For example, on the popped up dialog box corresponding to "Query Node", a list of nodes is displayed for the user's choice. This is shown in Figure 4.7. Buttons "APPLY", "CANCEL" and "HELP" are placed at the bottom of the dialog box. A scroll bar is provided if the list is longer than can be display at one time. The user can click the "CANCEL" button to return to the previous main menu window; or click "HELP" for available help; or select an item from the list and click "APPLY" to query the status of the selected node. The result is then presented to the user, as shown in Figure 4.8.

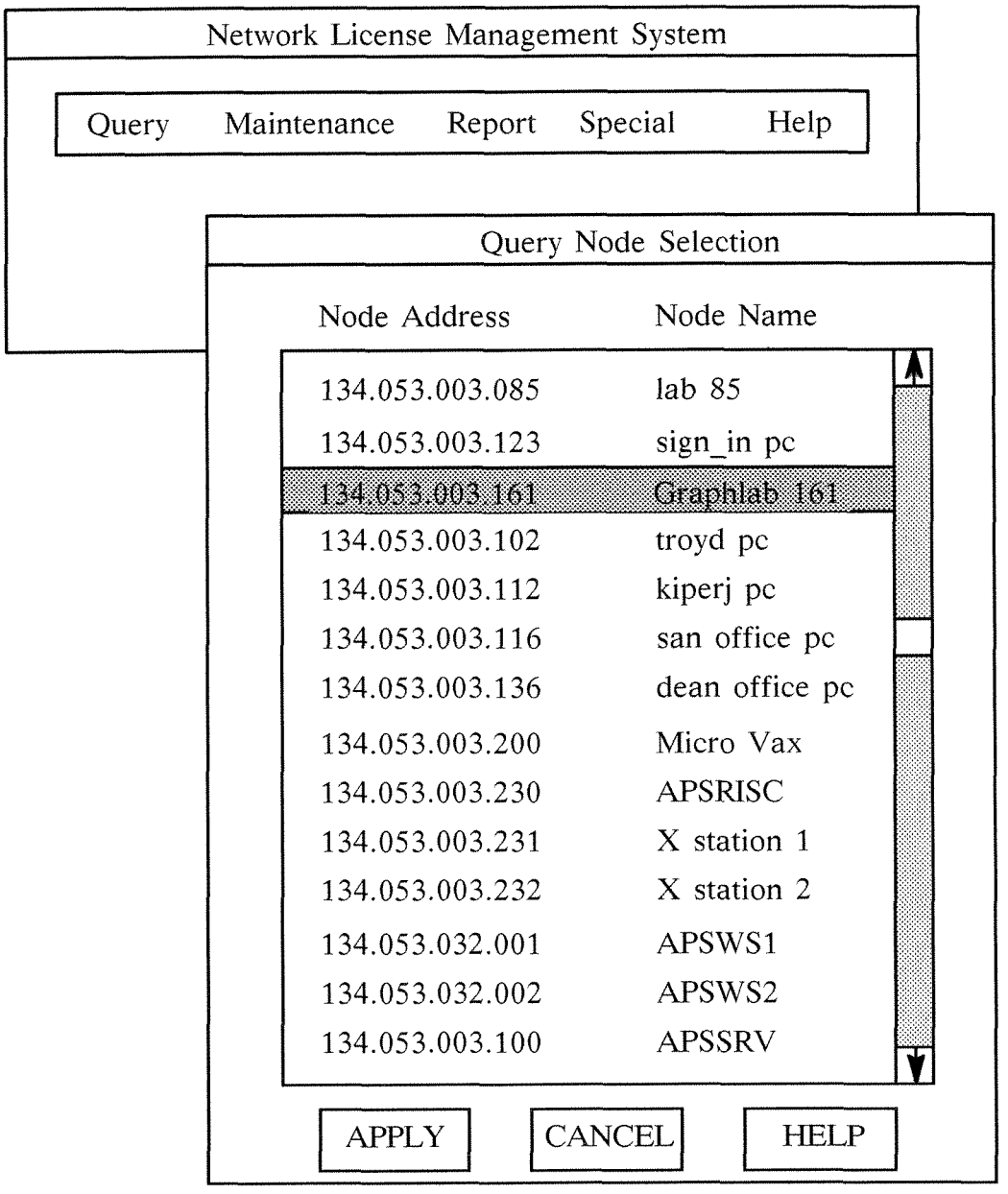


Figure 4.7 When "Node..." Under "Query" is selected

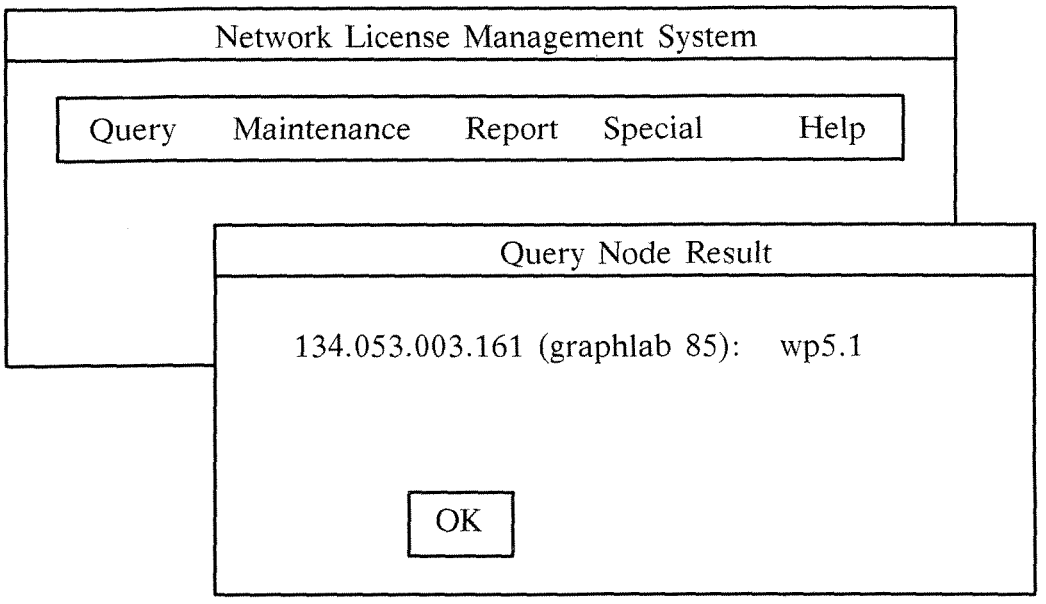


Figure 4.8 Query Node Result

This interface provides a hierarchical interaction between the program and the user, and since it is WYSIWYG (What you see is what you get), the interface is quite intuitive and is very convenient to use.

In this new interface environment, the mouse has become the most important input device. All the functions of the system can be invoked by moving the mouse, pointing to the desired item, and clicking the mouse to select. An item can be a menu item, a selection item or a control button.

The above windowing interface was expected to be implemented on the PC-DOS operating system and the UNIX operating system. Because of different windowing systems available on DOS and UNIX, the interface programs are different for these two versions.

4.3 Software Usage Reports

The usage reports should give the management people an overview of the frequency and duration of use of each software package. It should help people to understand how often and how long the software packages are used by the lab users. For example, in the morning, the lab manager may desire to have a report on the previous day to know if any package had been used at a level near its license limitation. That information can be shown by the

maximum number of packages being checked out concurrently compared to the number of packages licensed. The lab manager may also desire to know how much each package has been used, which can be shown by the total times each package has been checked out (regardless of concurrency) and the total length of time in minutes or in hours that each package has been used.

Similarly, the reports over a week, a month, or a semester can tell which packages are popular, and which were rarely touched during that week, month or semester.

According to the above requirements, the reports have five columns. The first one lists the names of software packages, the second tells how many licensed copies of the packages are owned by our microlab, the third shows the maximum number of copies of the package that were used concurrently during the day, the fourth shows the total times the package was checked out during the day, and the last tells the total time in minutes that the package was in use.

From the figures shown in these reports, the microlab manager can easily tell if a package has been used to its maximum capacity frequently, and hence, to purchase more copies when upgrading it, or to reduce the number of copies, if a package was rarely used over a long period of time.

By providing different time intervals for reporting, the manager may observe a usage pattern and thus enable him or her to make some arrangement of lab users' schedules to further increase the lab utilization.

5. Microsoft Windows Implementation

5.1 Interface Description

Microsoft Windows 3.0 is the windowing interface development tool we chose to use for the PC-DOS implementation. It is a popular product for personal computers and is the dominate windowing system for PC-DOS based systems. MS Windows provides a multitasking graphical-based windowing environment and a consistent appearance and command structure[Petzold, 1990] which makes it easy for the users to learn and use the applications.

The design of the MS Windows program consists of the screens shown in Section 4.2 User Interface, Figure 4.1 through Figure 4.8.

5.2 Programming with MS Windows

Windows has the reputation of being difficult for programmers. The approach to construct a window programs is quite different from a conventional, non-GUI program, and new concepts such as graphical devices, event or message-handling, and object-oriented programming have to be learned and digested[Petzold, 1990].

Programming Windows is especially difficult for beginners. MS-Windows Software Development Kit (SDK) provides a library of C functions that can be used to build a windows application[*Microsoft Windows Software Development Kit, Reference*, 1990]. It has more than five hundred functions and is overwhelming.

Another difficulty is that everything in Windows is interconnected[Petzold, 1990]. If you want to draw anything on the video display, you need a "handle to a device context." To get that, you need a "handle to a window." To have that, you must create a window and be prepared to receive "messages" to the window. To receive and process messages, you need a "window procedure." That is why even a program which only prints "Hello World" has to have all these components and is quite lengthy in its code.

Windows is started as a normal application program running under DOS. After it is loaded, it shares the responsibilities with the operating system for managing the hardware resources of the computer and scheduling processing power among applications[*Microsoft Windows Software Development Kit, Guide to Programming*, 1990]. Since Windows provides a multitasking environment, more than one program can run under Windows concurrently. Most of the time, these programs sit passively awaiting the user input, such as mouse movement, mouse click, or key stroke. Therefore, Windows programs must be event-driven. Unlike conventional programs that only make calls to the operating system to perform certain tasks, the operating system and Windows call and inform the application program of events so that the application can respond to the input events[Petzold, 1990]. For example, when a user resizes a window, Windows sends a message to the program indicating the new window size. Then the program can adjust the contents of its window to reflect the new size.

This is implemented by message passing and handling. Thus, a large portion of the program is the code to process the messages.

As we know, it is quite typical that there are multiple programs running under Windows, each with window(s) and menu bars or dialog boxes, etc. In MS Windows, all these entities are treated as objects. Windows are rectangular objects on the screen, which receive user input from the keyboard or mouse and display graphical output on its surface. An application window usually contains the program's title bar, menu, sizing border, and perhaps some scroll bars. Dialog boxes are additional windows which may contain several additional "child" windows. These child windows take the form of push buttons, radio buttons, check boxes, text entry fields, list boxes, or scroll bars. These windows are seen as objects by the users and are programmed as objects by the programmers as well.

Every window in a Windows program has an associated window procedure. This window procedure is a function that can be either in the program itself or in a dynamic link library. Windows sends a message to a window by calling the window procedure. The window procedure does some processing based on the message and then returns control to Windows[*Microsoft Windows Software Development Kit, Guide to Programming*, 1990].

In order to make the programming job easier and reuse the code of the window procedure, windows are grouped into classes. Every window class has a data structure, its "resources", to define its appearance, and a window procedure to process messages. The use of a window class allows multiple windows to be based on the same window class, and hence, use the same window procedure. In addition, MS Windows provides a rich set of predefined window classes such as menus, dialog boxes, buttons and other commonly used window classes that programmers can use. Thus, you only write window procedures for the windows that are created by your program and based on new window classes to fulfill your application's specific needs. Minimally, a window program needs to have one window procedure for the main window it creates.

5.3 MS Windows Implementation

My program consists of WinMain( ), WinProc( ) and other Dialogbox window procedures to invoke all the functionality of the client side of the NLMS. WinMain( ) is the entry

point of the program. It specifies the name, the window procedure, the icon, the cursor and other characteristics of the window class by filling out the window class structure WNDCLASS. Then it registers the window class with Windows and creates the top window of my program based on that window class. After calling some functions, the newly created window is displayed on the output device, it enters a "while loop" until the program is called to quit. It is the "while loop" that receives messages and dispatches them to the corresponding window procedures that actually process the messages.

WndProc( ) responds to some of the messages, such as the ones that indicate a menu selection or window resizing or window movement, and ignores other messages by passing those messages back to Windows and letting Windows handle them using default procedures. For example, part of the response to WindowCreate (received when the main window is created) is to obtain the starting point of all the other window procedures for later references. All the functionality of the NLMS client program is grouped into menu selections. A unique integer is assigned to each menu selection so that the program can identify each menu selection. The response that handles the reaction to menu selection is either to invoke a dialog box to prompt and get more information from the user or to issue a remote procedure call to the server and then display the result in a dialog box. There is one window procedure for each dialog box.

5.4 Incompatibility between MS Windows and SUN PC-NFS

The implementation of the MS Windows interface could not be completed because of an incompatibility between MS Windows and SUN PC-NFS. It turned out that as soon as our client code would issue an RPC call, the windows system would freeze up and no longer respond to user input. We consulted SUN, and were informed that this is due to a conflict in the way that PC-NFS and MS Windows both attempt to take over the PC's clock interrupts. After consultation with my thesis advisor, we decided that the programming effort to get around this problem was beyond the scope of my project, and we thus abandoned further development of the MS Windows implementation. However, I have done enough programming with MS Windows to be able to make comparisons between it and X Windows.

6. X Windows Implementation

6.1 X Windows System

The X Windows system is an industry-standard, device-independent and distributed software system[Young, 1990]. The device-independent feature of X Windows system provides a convenient and portable programming environment which allows programmers to concentrate primarily on his/her application design instead of worrying about portability issues among different machines.

This feature is achieved through its particular client-server architecture. The server process is responsible for both the input and output hardware and shields all the device-dependent operations. The client may be any application that uses the services provided by the X server. More than one client can be supported by a single X server. Therefore, the X window system is essentially a distributed system and hence it is necessary to have a X window manager to control the sharing of X window resources among multiple users. The X window manager controls the positions of the windows on the screen, and allows the user to move a window, resize a window or switch among windows, etc. It also processes all the requests sent by all the clients and this generally leads to more efficient utilization of the various computer resources.

Furthermore, the X server and the client do not need to be on the same machine provided that the display is a X terminal. For example, I once logged onto a SUN workstation at Ohio State University, telneted to our RISC/6000 machine, started the program and it ran perfectly. In this case, the processing is taking place on our RISC/6000 and the output is displayed on the SUN station. Amazingly, it does not require recompilation to have the interface displayed across the network. I also noticed that the interface components (menus, dialog box, borders, arrows, etc.) had a different look because the display was running a different window manager.

6.2 Programming with X Windows

Although the X protocol is defined at the very low level of network packets and byte-streams, application programmers generally make use of higher level library functions that provide an interface to the base window system. The most widely used low-level inter-

face to X is the C language library known as Xlib. Xlib defines an extensive set of functions that provide complete access and control over the display, windows, and input devices. Based on Xlib, toolkits have been built to provide more convenient ways of programming. Examples of toolkits are InterViews (Stanford University), Andrew (Carnegie Mellon), Xray (Hewlett Packard), etc[Young, 1990].

The toolkit I used is a standard toolkit known as the X Toolkit, which consists of two parts: a layer known as the Xt Intrinsics and a set of user interface components known as widgets. The Xt Intrinsics supports many different widget sets, including OSF/Motif, Open Look and Athena. OSF/Motif 1.0 comes with the IBM RISC/6000 AIX and is the widget set I chose to use. This layered architecture of an X windows system provides portability and flexibility. Figure 6.1 is the application programmer's view of the X Windows system[Young, 1990].

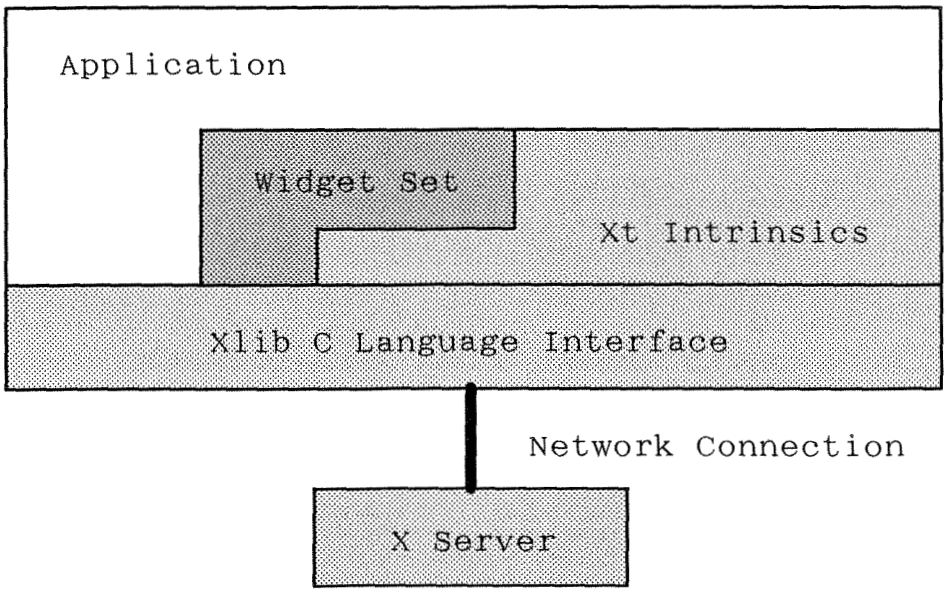


Figure 6.1 The layers of X Windows System from Application Programmer's View

6.3 X Windows Implementation

Applications with a windowing user interface may contain a large while loop, which dispatches all the events. For example:

```
while (TRUE) do
    get next event;
   case event of
     pushed "help" button:  HelpProcessProcedure();
    ...
    end case
end while
```

This sort of loop is called "busy waiting." It is not a desirable feature of a system, since it wastes the precious computer resources. In X Windows/Motif, the Callback function implementation provides an alternative to the "busy waiting" event-processing mechanism, which is an interrupt-driven or event-driven mechanism[*OSF/Motif Programmer's Guide*, 1990]. The system requires an application to register in advance all the actions that will be taken to respond to each event. Therefore a large part of the program is chained event processing procedures to handle user input. A convenient function is available to register a callback function[*OSF/Motif Programmer's References*, 1990].

```
XtAddCallback(button, XmNactiveOK, MenuCallback, QNODE);
```

The above line is a statement in my program after "button," a widget of type BUTTON, is created to represent an option "query node" under the top menu "Query." The XtAddCallback function specifies that when this button is clicked (XmNactiveOK), a procedure MenuCallback( ) is invoked and executed, and QNODE is an integer passed to MenuCallback( ) as a piece of client data. The MenuCallback( ) procedure creates a selection dialog box displaying all the nodes for the user to select. The callback function for this selection dialog box is CallRPC( ). It is in the CallRPC( ) procedure that the remote procedure call is sent after it determines which node is being selected by the user. From a programmer's perspective, it is truly interrupt-driven.

As we can see, in addition to saving the computer resources, the interrupt-driven feature of the X windows system makes an application easier to write soon after you get used to it.

7. Management Reports

7.1 Types of Reports

There are two important requirements that need to be addressed for the statistical reports. One is the time interval for the report. It should be flexible and should not require too much typing to specify. So Mrs. Baker and I decided to have different options, such as daily, weekly, monthly and semester reports. A daily report produces the collective information of usage for any specified day. Weekly, monthly and semester reports give the corresponding statistical report over the specified week, month or semester, respectively.

The second requirement concerns the content of the reports. It should contain relevant information for management. We determined four types of data to support: the number of copies licensed, the maximum copies in use concurrently, the total number of times a package was used during a day, and the total time length a package was in use during a day. The last data is necessary because Mrs. Baker suggested that some packages are checked back in right after being checked out, which causes the total times a package has been checked out to misrepresent the usage. Having this report, the lab manager knows what packages have been used heavily and frequently hit maximum capacity, and what packages have rarely been used by students. This will provide a piece of key information to help in making future purchase plans.

7.2 Implementation

The audit file generated every day records each request from all the clients in the time sequence as it happens during that day. It looks like:

```
00.00.01 SEX_MAX    000.000.000.000 qc      25
09.01.23 CHECK_OUT 134.053.003.056 wp51
```

It shows at time 00.00.01 the package Quick C was set to have 25 licensed copies and at 09.01.23 client node 134.053.003.056, one of the PCs, checked out WordPerfect 5.1. The file for a typical school day has 4000 to 6000 such lines.

Since the file is so large, it would take a long time to produce a report if the audit file were read in and analyzed at the time a report was requested. For example, a monthly or semester report requires 30 or more than 100 such files to be scanned. Also, if a report is

requested more than once for a day, the reading and analysis process for that file is repeated completely.

In order to prevent the scenario that a user has to wait a few minutes to get the report, I took advantage of UNIX's multitasking capabilities. After midnight, the server forks another process, and the newly created process accumulates usage information from that day's audit file.

The algorithm is very simple. At the beginning, an array:

```
struct package_rec
{    char package[16];
     int max_copy;                  /* # of copies licensed */
     int concur_copy;               /* max # of copies used concurrently */
     int times_checkedout;          /* max # of times being checked out */
     int amount_time_in_use;        /* total time in seconds a package in use */
} package_info[MAX_PACKAGES];
```

is initialized. Then the process scans every line of the audit file. The "SET_MAX" is used to set the value of "max_copy" for each package to get the number of license. During of the process of scanning, only "CHECK_OUT" and "CHECK_IN" are read into memory because they are the only information related to lab usage. A sort function is then used to group all the "CHECK_OUT" and "CHECK_IN" information by the client nodes. So for each node, we are able to generate repeated pairs of "check out a package at one time, and check in that package at a later time." The difference in time between the consecutive pair of "check out" and "check in" is assumed to be the time that package has been in use. This accumulates to the total time in seconds that package has been in use. At the same time, concur_copy and times_checkedout is also computed.

This way, a daily usage report is generated and stored in a file for each day. The report file is named as Month_Day_Weekday.report. For instance, Aug_03_Mon.report is the report file name for August third. Thus when a report is called on, only these small summary data files (each contains less than 100 lines) have to be read in and the report can be ready in seconds. An example of the report is shown in Appendix 1.

8. Bugs Discovered and Corrected

In the course of the project, a few bugs were discovered in the old NLMS program in our Microlab. First, the original design of "check", reversed the intuitive meaning of "in" and "out". "Check in" was used when a package was to be requested, and "check out" was used when the user was done with the package. This use of "in" and "out" was confusing, and had to be reversed.

Secondly, since the communication between the client and the NLMS server is implemented via the mechanism of User Datagram Protocol (UDP), and since the maximum length of data communicating between the client and the NLMS server is 1024 bytes in UDP[*PC–NFS Programmer's Toolkit Manual*, 1987], a client query, when issued from a PC and requires communication data longer than 1024 bytes, would fail. Query nodes and query license are two functions whose returned data are longer than 1024 bytes. Therefore, these two functions behave abnormally in the old NLMS when they were invoked from a PC. In order for the communication data between the client and the NLMS server to accommodate more than 1024 bytes long, a different transport protocol is needed. The more reliable and powerful transport protocol TCP is used as the underlying protocol of RPC. Therefore, the network manager can execute the query nodes and query license with the desired result.

To recover from a server failure, the NLMS server maintains an audit file on a daily basis to record all the software usage information. When restarting the NLMS server "libraira" after a crash, the audit file is used to restore the network state back to the one before the failure[Troy, 1991]. The old program failed to read in all the information from the audit file correctly, and thus, the restored state did not reflect the correct software usage in the network. Furthermore, the amount of information kept in the audit files was insufficient to generate the statistical reports. The above limitations and bugs were corrected as a part of my project.

9. Comparison of Programming in Microsoft Windows and X Windows

MS Windows and X Windows/Motif are similar at the high conceptual level and different at the detailed programming level.

9.1 Concepts

They both use an object-oriented approach. Although programmers can use their conventional procedure-oriented language interface, the application program needs to be centered around the concepts of objects because all the interface components are objects. Both systems are event-driven and significant amount of code is required to process the user input events.

I felt that it was very difficult for me to get to a level where I felt comfortable in writing X Window programs, partially because I had to learn X Windows/Motif on my own and I did not have good books on the topic at that time. Another reason is that no debug tools were available. Quite often, a small mistake in the program caused the windows in my program to disappear with no information about the error. It was thus very time consuming to find errors and made me less likely to experiment.

9.2 Programming

At the coding level, programming MS Windows is more tedious than programming X Windows/Motif. One reason is that Motif is built on top of X lib and Xt Intrinsics and thus provides a rich set of high level functions to assist with the implementation of the interface. Another reason is that it assumes a "window manager" is running at the client workstation to help manage some of the hardware resource, especially the X terminal. The X window manager handles what is actually displayed on the screen taking into account the layout and overlapping of the windows. In MS Windows, the application program has the responsibility of repainting the screen when the window moved to another position, resized, or some portion of your window became visible because other windows were moved away.

The logic or steps of writing programs in Motif is quite straight-forward. You have a pattern to follow: create an interface component by creating a widget of suitable class, specify its parent widget, define its appearance using some of its resources, and define its behavior using the callback function(s). A Widget is the object; the call back function is the procedure to handle the user input.

While in MS Windows, the flow of control is not clear by looking at the code because messages can be generated from window function calls. In order to have a workable window

program, one needs to follow the program structure recommended by Microsoft. For example, after the main window is created, you need to have two statements:

```
ShowWindow(...);
UpdateWindow(...);
```

What they do is to send to the window procedure a message WM_PAINT requiring the application program to paint the newly created window. If these two lines had been missing in your program, you could not get your window on the monitor at all. In the window procedure, part of its code in the big event loop responds to this message:

```
case WM_PAINT:
    hdc=BeginPaint (...);
    GetClientRect (...);
    DrawText (...);
    EndPaint(...);
    return 0;
/* other case blocks */
```

This part of the code is to repaint the content of the window when receiving the message WM_PAINT, which arrives at the time of window creation(ShowWindow( ) procedure call) or later on when the window needs to be repainted.

I personally prefer X Windows/Motif programming environment. If I am going to develop a user interface and I have a choice of either MS Windows and X Windows/Motif, I would choose X Windows/Motif.

10. Conclusion

10.1 Interactive Design Tools

After completing the programming, I have done some research on the tools that can make windows programming easier. If the only tools available to develop a user interface are library functions, the time required to learn, code, and test is a big concern, and programming becomes very tedious. Visual design tools are being developed to boost the productivity of user interface design and implementation under Microsoft Windows. The simplest one is the Dialog box Editor, which lets a user position and size buttons and other

controls within a dialog box. The Editor generates resource script statements and can be used in conjunction with the source code[*Microsoft Windows Software Development Kit, Guide to Programming*, 1990]. More sophisticated tools provide a visual development environment (VDE), which allows users to prototype and design all kinds of interface objects interactively, and generates source code that can be modified and compiled[Petzold, 1992]. But the logic control part of the program still needs to be coded using a programming language and sometimes it can get more complicated in the VDE because the VDE is less flexible. Even so, these tools appear to take some of the pain out of interface design and generally they save some development time[Petzold, 1992].

There are some commercial OSF/Motif GUI builders available, similar to VDE. One is called Interactive Design Tools that allow a user to specify the interface components and generate corresponding source code. For example, TeleUSE from Telesoft claims to be a full-featured User Interface Management System that prototypes, designs, and implements GUIs using an object-oriented methodology[Hogan, 1992]. It is reported that the develop time using these tools to build interfaces can be cut by fifty to eighty percent compared to directly using Motif function calls.

Since I do not have first hand experience with these tools, I can only assume that they would help out with the tedious part of development, and still need coding to produce the flow of control and to put all the interface components together.

10.2 Improvements

The usage reports can be improved further. Right now, the reports only present all the figures to show the actual usage of all the software packages owned by the lab. If we can define the criteria for high usage rate (above 90% or 95% of license limitation) and low usage rate (below 50% of license limitation), the reports can flag the packages whose usage falls into these two categories, in addition to providing those usage statistics. Thus the reports would provide information as opposed to just data.

User interface design is a on-going process. The new interface is designed based upon the prototype and current requirements. As users continuing using the system, they may come up with new ideas and new requirements, which can be incorporated and result in

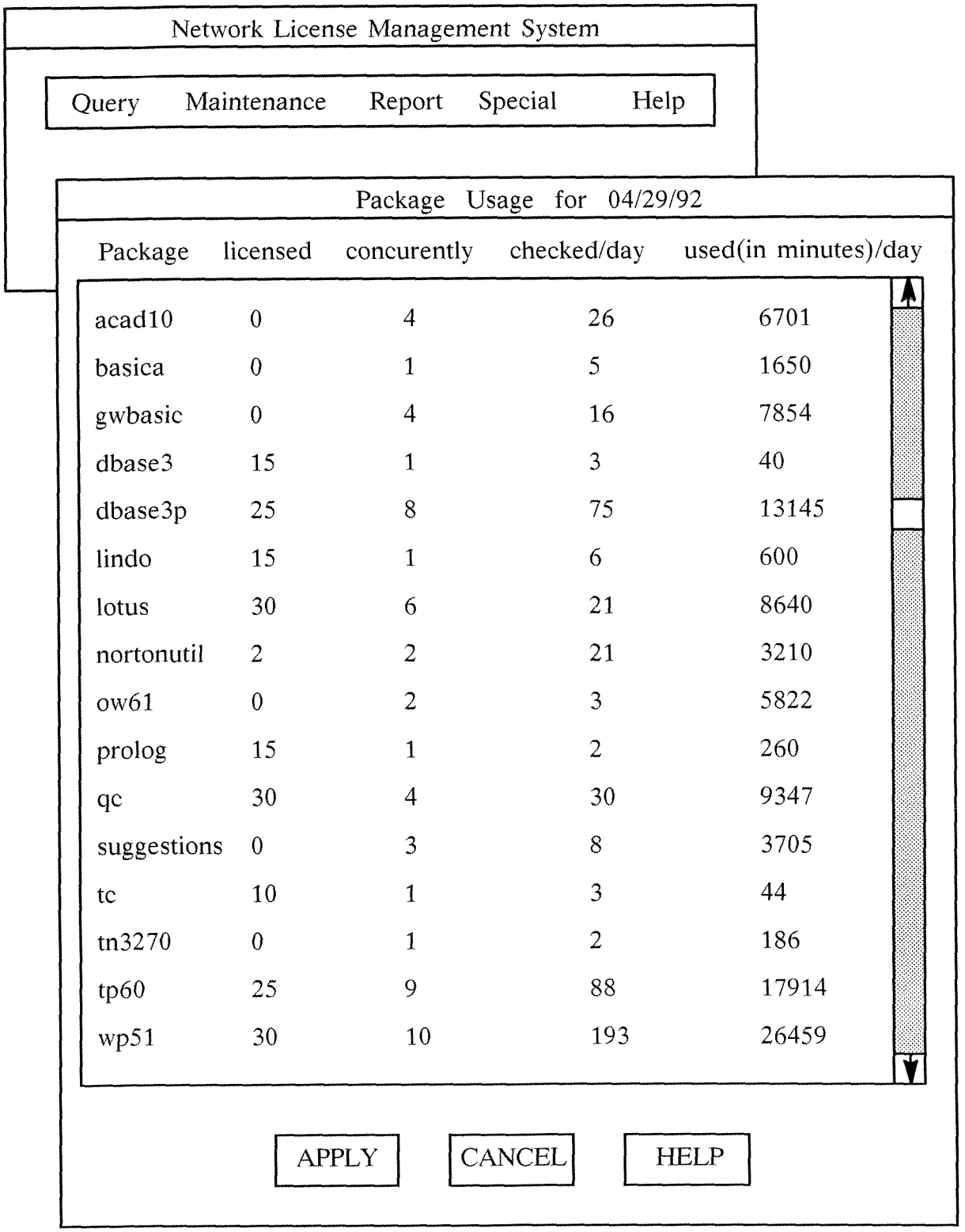
a better design for the user.

The reasons that I chose this project are that I wanted to do some developmental work in the UNIX environment; I wanted to understand more about network programming, especially client–server models of the network; I wanted to learn a way to write distributed applications; and I wanted to learn how to design and develop applications in the two most popular windowing environments, namely Microsoft Windows and X Windows/Motif. I feel that by doing this program, my knowledge has been broadened, and my learning skills and my programming skills have been improved greatly.

References

1. Ames, C., *Overview of RPC*, unpublished report for Distributed Processing and Networking, Systems Analysis, Miami University, 1991
2. Brain, M., *Motif tutorials*, Version 1.1, NetNews (comp.windows.x.motif), 1990
3. Brown, J. & Cunningham, S., *Programming the User Interface*, Wiley, 1989
4. Comer, D. E., *Internetworking with TCP/IP*, Vol 1, Prentice Hall, 1991
5. Cypser, R. J., *Communications for Cooperating Systems OSI, SNA, and TCP/IP*, Addison-Wesley, 1991
6. Foley, J., van Dam, A., Feiner, S. & Hughes J., *Computer Graphics – Principles and Pracitce*, Addison-Wesley, 1990
7. Hogan, T., *TeleUSE*, UNIX Preview, Vol. 10, No. 4, pp 8., April 1992
8. Petzold, C., *Programming Windows, The Microsoft Guide to Writing Applications for Windows 3*, Microsoft Press, 1990
9. Petzold, C., *The Visual Development Environment: More than Just a Pretty Face?* PC Magazine, Vol 11, No 11, pp. 195 – 204, June 1992
10. Sommerville, J., *Software Engineering*, Addison Wesley, 1989
11. Troy, D. A., *Librarian, A Multi-User License Manager*, Miami University, Systems Analysis Department Technical Report 1992-0012, 1991
12. Young, D. A., *The X Window System – Programming and Applications with Xt*, OSF/Motif Edition, Prentice Hall, 1990
13. *Microsoft Windows Software Development Kit, Guide to Programming*, Microsoft Press, 1990
14. *Microsoft Windows Software Development Kit, Reference*, Vol. 1, Microsoft Press, 1990
15. *OSF/Motif Programmer's Guide*, Revision 1.0, Open Software Foundation, Prentice Hall, 1990
16. *OSF/Motif Programmer's References*, Revision 1.0, Open Software Foundation, Prentice Hall, 1990
17. *PC-NFS Programmer's Toolkit Manual*, Sun Microsystems, April 1987
18. *X Protocol*, Ultrix Worksystem Software, Programming Volume 5, Digital Equipment Corporation, 1988
19. *Xlib Library*, Ultrix Worksystem Software, Programming Volume 4, Digital Equipment Corporation, 1988

Appendix 1 An Example of Management Reports



Appendix 2 Source Code Listings

1. libraria.h
2. libraria.c
3. xface.c

```
/*****************************************************************************/
/*                                                                           */
/*   Include File: LIBRARIA.H                                                */
/*                                                                           */
/*   This header has all the predefined constants, structures, and           */
/*   translation routines used by LIBRARIA, CHECK, PCTIME, XFACE and LIBMAINT*/
/*                                                                           */
/*   Eight translation routines are present:                                 */
/*                                                                           */
/*   xdr_string16       - allows the encoding and decoding of strings whose  */
/*       length does not exceed 15 characters (as in node number, package    */
/*       name).                                                              */
/*   xdr_date           - allows the encoding and decoding of MOST of the tm */
/*       time structure found in <time.h>.                                   */
/*   xdr_software_block - allows the encoding and decoding of package        */
/*       structures.  This is used for the passing of complete package info  */
/*       during queries.                                                     */
/*   xdr_node_block     - allows the encoding and decoding of node           */
/*       structures.  This is used for the passing of complete node info     */
/*       during queries.                                                     */
/*   xdr_node_array     - allows the complete passing of a node array        */
/*       (client_list).                                                      */
/*   xdr_package_array  - allows the complete passing of a package array     */
/*       between APSLIB and LIBMAINT.                                        */
/*   xdr_usage_block    - allows the encoding and decoding of report         */
/*       structures.  This is used for the passing of complete report info   */
/*       during requests.                                                    */
/*   xdr_usage_array    - allows the complete passing of a usage array       */
/*       between APSLAB and LIBMAINT.                                        */
/*                                                                           */
/*****************************************************************************/

#define _BSD
#include <rpc/rpc.h>
#include <sys/socket.h>
#include <netdb.h>
#include <time.h>

#define LICENSE_DB "software.lab"
#define ERRORLOGNAME "license.err"

#define RPC_ERROR      1
#define MAX_ERRORS    10
#define IDLE_MACHINE "automenu"
#define MAX_PACKAGES 150
#define MAX_CLIENTS  150
#define SOFTLIB 550000001
#define SOFTVERS   2

/* User-available functions       */
#define NULL_PROC  0
#define CHECK_IN   1
#define CHECK_OUT  2
#define READ       3

/* Maintenance-level functions    */
#define REMOVE     16
#define QNODE      17
#define QPACKAGE   18
#define QUSERS     19
#define QTIME      20
#define QNODES     21
#define QLICENSE   22

/* Administrative-level functions */
#define SET_MAX  128
```

```
#define SHUTDOWN 129
#define RESTART  130
#define CLEANUP  131

/* Report-related functions */
#define DAILY    256
#define WEEKLY   257
#define MONTHLY  258
#define SEMESTER 259

char weekday[7][4]={"Sun","Mon","Tue","Wed","Thu","Fri","Sat"};
char day[31][3]={"01","02","03","04","05","06","07","08","09","10",
                 "11","12","13","14","15","16","17","18","19","20",
                 "21","22","23","24","25","26","27","28","29","30","31"};
char month[12][4]={"Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct",
                   "Nov","Dec"};
int first_day_of_month[12]={3,6,0,3,5,1,3,6,2,4,0,2};
int no_of_days[12]={31,29,31,30,31,30,31,31,30,31,30,31};


/*****************************************************************************/
/*                                                                           */
/* The following two structures are the building blocks for APSLIB:          */
/* SOFTWARE_BLOCK and NODE_BLOCK.  An array of SOFTWARE_BLOCK contains all   */
/* known software data in the librarian server, and an array of NODE_BLOCK   */
/* contains all current node information.                                    */
/*                                                                           */
/*****************************************************************************/

struct SOFTWARE_BLOCK
{
   char software_package[16];
   short int total_copies;
   short int copies_in_use;
};

 struct NODE_BLOCK
{
   char node_address[16];
   char software_package[16];
   struct SOFTWARE_BLOCK *software_in_use;
};

struct USAGE_BLOCK
{
   char software_package[16];
   short int copies;
   short int concurrent;
   short int checked;
   int time_used;
};

struct DATE
{
   int mday, mon, year, wday;
};

/*****************************************************************************/
/*                                                                           */
/* There are several global variables used by all APSLIB related programs.   */
/* They include:                                                             */
/*                                                                           */
/*      software_list is an array of MAX_PACKAGES elements composed of       */
/*      SOFTWARE_BLOCK.                                                      */
/*      number_of_packages contains the number of SOFTWARE_BLOCKs in use.    */
/*                                                                           */
```

```
/*      client_list is an array of MAX_CLIENTS elements composed of          */
/*      NODE_BLOCK.                                                          */
/*      number_of_clients contains the number of NODE_BLOCKs in use.         */
/*                                                                           */
/*                                                                           */
/*****************************************************************************/

struct SOFTWARE_BLOCK software_list[MAX_PACKAGES];
struct     NODE_BLOCK   client_list[MAX_CLIENTS ];
struct    USAGE_BLOCK    usage_list[MAX_PACKAGES];
int    number_of_packages=1;
int    number_of_clients =0;
int    in_usage_number_of_packages=0;
int    number_in_usage = 0;


/*****************************************************************************/
/*                                                                           */
/* The following six xdr routines are six of eight fundamental               */
/* translation routines used by APSLIB (the others being XDR_INT and         */
/* XDR_STRING).                                                              */
/*                                                                           */
/*****************************************************************************/

xdr_string16(xdrsp,xdr_data)
    XDR  *xdrsp;
    char *xdr_data;
{
    if (!xdr_string(xdrsp,&xdr_data,16)) return (0);
    return (1);
}

xdr_mydate(xdrsp,xdr_data)
    XDR *xdrsp;
    struct DATE *xdr_data;
{
    if (!xdr_int(xdrsp,&xdr_data->mday)) return(0); /* 1-31  day of month */
    if (!xdr_int(xdrsp,&xdr_data->mon )) return(0); /* 1-12  month        */
    if (!xdr_int(xdrsp,&xdr_data->year)) return(0); /* 0-    year - 1900  */
    if (!xdr_int(xdrsp,&xdr_data->wday)) return(0); /* 0-6   day of week  */
    return(1);
}

xdr_date(xdrsp,xdr_data)
    XDR *xdrsp;
    struct tm *xdr_data;
{
    if (!xdr_int(xdrsp,&xdr_data->tm_sec )) return(0); /* 0-59  seconds      */
    if (!xdr_int(xdrsp,&xdr_data->tm_min )) return(0); /* 0-59  minutes      */
    if (!xdr_int(xdrsp,&xdr_data->tm_hour)) return(0); /* 0-23  hour         */
    if (!xdr_int(xdrsp,&xdr_data->tm_mday)) return(0); /* 1-31  day of month */
    if (!xdr_int(xdrsp,&xdr_data->tm_mon )) return(0); /* 0-11  month        */
    if (!xdr_int(xdrsp,&xdr_data->tm_year)) return(0); /* 0-    year - 1900  */
    if (!xdr_int(xdrsp,&xdr_data->tm_wday)) return(0); /* 0-6   day of week  */
    if (!xdr_int(xdrsp,&xdr_data->tm_yday)) return(0); /* 0-365 day of year  */
    return(1);
}

xdr_software_block(xdrsp,xdr_data)
    XDR *xdrsp;
    struct SOFTWARE_BLOCK *xdr_data;
{
    char *string_pointer_1;
    string_pointer_1=xdr_data->software_package;

    if (!xdr_string(xdrsp,&string_pointer_1,16    )) return(0);
```

```
    if (!xdr_short (xdrsp,&xdr_data->total_copies )) return(0);
    if (!xdr_short (xdrsp,&xdr_data->copies_in_use)) return(0);
    return(1);
}

xdr_node_block(xdrsp,xdr_data)
    XDR *xdrsp;
    struct NODE_BLOCK *xdr_data;
{
    char *string_pointer_1,*string_pointer_2;
    string_pointer_1=xdr_data->node_address;
    string_pointer_2=xdr_data->software_package;
    if (!xdr_string(xdrsp,&string_pointer_1,16)) return(0);
    if (!xdr_string(xdrsp,&string_pointer_2,16)) return(0);
    return(1);
}


xdr_node_array(xdrsp,xdr_data)
    XDR *xdrsp;
    struct NODE_BLOCK *xdr_data;
{
    if (!xdr_array(xdrsp,xdr_data,&number_of_clients,MAX_CLIENTS,
                sizeof(struct NODE_BLOCK),xdr_node_block)) return(0);
    return(1);
}

xdr_package_array(xdrsp,xdr_data)
    XDR *xdrsp;
    struct SOFTWARE_BLOCK *xdr_data;
{
    if (!xdr_array(xdrsp,xdr_data,&number_of_packages,MAX_PACKAGES,
                sizeof(struct SOFTWARE_BLOCK),xdr_software_block)) return(0);
    return(1);
}

xdr_usage_block(xdrsp,xdr_data)
    XDR *xdrsp;
    struct USAGE_BLOCK *xdr_data;
{
    char *string_pointer_1;
    string_pointer_1=xdr_data->software_package;

    if (!xdr_string(xdrsp,&string_pointer_1,16    )) return(0);
    if (!xdr_short (xdrsp,&xdr_data->copies )) return(0);
    if (!xdr_short (xdrsp,&xdr_data->concurrent)) return(0);
    if (!xdr_short (xdrsp,&xdr_data->checked)) return(0);
    if (!xdr_int   (xdrsp,&xdr_data->time_used)) return(0);
    return(1);
}

xdr_usage_array(xdrsp,xdr_data)
    XDR *xdrsp;
    struct USAGE_BLOCK *xdr_data;
{
    if (!xdr_array(xdrsp,xdr_data,&in_usage_number_of_packages,MAX_PACKAGES,
                sizeof(struct USAGE_BLOCK),xdr_usage_block)) return(0);
    return(1);
}
```

```
/*************************************************************************/
/*                                                                       */
/*   Program: LIBRARIA.C                                                 */
/*                                                                       */
/*   Purpose: NLMS server program                                        */
/*                                                                       */
/*   Port History: This program runs on NeXT, DEC MicroVax II, and       */
/*                 IBM RS/6000                                           */
/*                                                                       */
/*   Purpose: This is the Applied Science software librarian.  It accepts */
/*            requests from microcomputer-based clients, and returns the */
/*            appropriate responses.  Requests are as follows:           */
/*                                                                       */
/*            RPC request     action                                     */
/*            -----------     ------------------------------------------ */
/*            SHUTDOWN        This function performs a controlled shutdown */
/*                            of the librarian server.  All files are    */
/*                            closed and all socket connections are torn */
/*                            down.                                      */
/*                                                                       */
/*            RESTART         This function re-initializes all relevent data */
/*                            structures (client_list and software_list are */
/*                            zeroed) and the site agreement file (software. */
/*                            lab) is re-read.                           */
/*                                                                       */
/*            CHECK_OUT        The client-PC is requesting to check out  */
/*                            (gain possession of a copy) a software     */
/*                            package.  If the number of copies in use is */
/*                            less than the site agreement maximum for the */
/*                            package, the librarian returns a zero to the */
/*                            client-PC, indicating that it can use the  */
/*                            package.  If the maximum number of copies are */
/*                            in use, a non-zero is returned.            */
/*                                                                       */
/*            CHECK_IN        The client-PC is signalling that it has    */
/*                            finished using a software package, and the */
/*                            number of copies in use for that package is */
/*                            decremented. The default package, IDLE_MACHINE */
/*                            is then checked in.  In the event of two   */
/*                            consecutive CHECK_OUTs, an automatic CHECK_IN */
/*                            is implied.                                */
/*                                                                       */
/*            REMOVE          Occasionally a node will die (as in a hardware */
/*                            failure) without releasing a software package. */
/*                            This function removes a node from the data- */
/*                            base and decrements the total usage on the */
/*                            software it was using at the time of death. */
/*                                                                       */
/*            SET_MAX         In the event that the software librarian is */
/*                            running, and the system administrator wishes */
/*                            to change the site agreement file, two options */
/*                            exist - SHUTDOWN the system and edit the file, */
/*                            or issue a SET_MAX and have the file rewritten */
/*                            automatically.  SET_MAX will fail if the new */
/*                            site requirement is less than the number of */
/*                            copies in use.                             */
/*                                                                       */
/*            QNODE           Given a node number, this function returns the */
/*                            package being used on that node.  It is    */
/*                            convenient when monitoring a group of PCs. */
/*                                                                       */
/*            QPACKAGE        This query returns the number of copies in use */
/*                            for a given software package as well as the */
/*                            site agreement restriction.                */
/*                                                                       */
/*            QUSERS          This query returns the number of active and */
```

```
/*                            idle nodes registered in the network. Idle */
/*                            nodes are determined by checking the usage of */
/*                            AutoMenu.                                  */
/*                                                                       */
/*            QTIME           One of the nice features of a centralized  */
/*                            server is the ability to coordinate several */
/*                            aspects - such as a network time.  This    */
/*                            function returns the time of day to the    */
/*                            client-PC.                                 */
/*                                                                       */
/*            QNODES          This query returns every node address using */
/*                            the server and the package currently being */
/*                            executed.                                  */
/*                                                                       */
/*            QLICENSE        This query returns every software package in */
/*                            the librarian database with the maximum    */
/*                            allowable package usage (the site license  */
/*                            agreement) and the actual number of copies in */
/*                            use.                                       */
/*                                                                       */
/*            DAILY           This request produces a usage report for a spe-*/
/*                            cified day.                                */
/*            WEEKLY          This request produces a usage report for a spe-*/
/*                            cified week.                               */
/*            MONTHLY         This request produces a usage report for a spe-*/
/*                            cified monty.                              */
/*            SEMESTER        This request produces a usage report for a spe-*/
/*                            cified semester.                           */
/*************************************************************************/

#include "libraria.h"
#include <stdio.h>
#include <fcntl.h>

/*************************************************************************/
/*                                                                       */
/* There are several global variables used by the rest of the program.   */
/* They include:                                                         */
/*                                                                       */
/*      audit_file is the file containing the librarian-generated audit, and */
/*      error_log contains the librarian-generated error messages.       */
/*                                                                       */
/*      starting_date contains the time-dependent audit file name.  Every */
/*      time an audit is generated, starting_date is compared to the system */
/*      date.  If they do no match, a new audit trail is created and     */
/*      starting_date is updated.                                        */
/*                                                                       */
/*      days_in_service contains the number of days that the server has been */
/*      running without shutdown or crash.  Special things must occur on the */
/*      zeroeth day - the system must be restarted.                      */
/*                                                                       */
/*      number_of_system_errors contains the number of non-fatal system  */
/*      errors that have occurred since system start.  If this number    */
/*      exceeds MAX_ERRORS, the system is terminated.                    */
/*                                                                       */
/*************************************************************************/

int    number_of_system_errors;
int    days_in_service=0;
FILE  *audit_file, *error_log;
char   starting_date[11];
char   audit_file_name[20];
int    child_pid = 0;

/*************************************************************************/
/*                                                                       */
```

```
/* Procedure "open_error_log"                                                */
/*                                                                           */
/* This procedure opens the error log file ("apslib.errors").  If an error   */
/* during this procedure, we have one valid option - terminate the program.  */
/* This should be the first procedure called by this program.                */
/*                                                                           */
/*****************************************************************************/

open_error_log()
{
   if ((error_log=fopen(ERRORLOGNAME,"w"))==NULL)
       {
          printf("\nFATAL ERROR: Unable to write error log file.\n");
          exit(1);
       }
}


/*****************************************************************************/
/*                                                                           */
/* Procedure "system_error"                                                  */
/*                                                                           */
/* This procedure, given a pointer to an error message, prints the time and  */
/* the error message in the error log file and increments the number of      */
/* overall system errors.                                                    */
/*                                                                           */
/*****************************************************************************/

system_error(error_message)
   char *error_message;
{
   time_t time_of_error;
   char error_message_time[21];

   time(&time_of_error);
   strncpy(error_message_time,ctime(&time_of_error),20);
   error_message_time[20]='\0';
   fprintf(error_log,"%20s%s\n",error_message_time,error_message);
   fflush(error_log);
   number_of_system_errors++;
   if (number_of_system_errors==MAX_ERRORS) exit(1);
}


/*****************************************************************************/
/*                                                                           */
/* Procedure "initialize_software_list"                                      */
/*                                                                           */
/* This procedure initializes all MAX_PACKAGES software_list entries.  The   */
/* software package name is set to null, the total copies for that package   */
/* is set to zero, and the copies in use for the package is set to zero.     */
/*                                                                           */
/* Software_list[0] always is set to contain "AutoMenu", the equivalent of   */
/* a lab PC in an idle state.                                                */
/*                                                                           */
/*****************************************************************************/

initialize_software_list()
{
   int counter;
   for (counter=0; counter<MAX_PACKAGES; counter++)
       {
          *(software_list[counter].software_package)='\0';
          software_list[counter].total_copies=0;
          software_list[counter].copies_in_use=0;
       }
```

```
   strcpy(software_list[0].software_package,IDLE_MACHINE);
   number_of_packages=1;
}


/*****************************************************************************/
/*                                                                           */
/* Procedure "initialize_client_list"                                        */
/*                                                                           */
/* This procedure initializes all MAX_CLIENTS client_list entries.  The node */
/* address for each client list is set to 000.000.000.000, and the software  */
/* in use is set to software_list[0] ("AutoMenu").                           */
/*                                                                           */
/*****************************************************************************/

initialize_client_list()
{
   int counter;
   for (counter=0; counter<MAX_CLIENTS; counter++)
       {
          strcpy(client_list[counter].node_address,"000.000.000.000");
          strcpy(client_list[counter].software_package,"");
          client_list[counter].software_in_use= &software_list[0];
       }
    number_of_clients=0;
}


/*****************************************************************************/
/*                                                                           */
/* Function "find_software_package"                                          */
/*                                                                           */
/* This function, given a package name and an addition flag, returns an      */
/* integer corresponding to its location in software_list.  If the package   */
/* can not be found, and the addition flag is TRUE, an attempt is made to    */
/* add the package name into the software_list.  If addition is attempted,   */
/* but all array elements are in use, a -1 is returned (indicating that the  */
/* the client-PC will remain idle), and an error log is generated.           */
/*                                                                           */
/*****************************************************************************/

int find_software_package(package_name,add_package)
    char *package_name;
    int  add_package;
{
    int software_location=0;
    int software_found=FALSE;
    while ((software_location<number_of_packages) && (software_found==FALSE))
          {
             if (strcmp(software_list[software_location].
             software_package,package_name)!=0) software_location++;
             else software_found=TRUE;
          }

    if ((software_found==FALSE) && (number_of_packages<MAX_PACKAGES))
         if (add_package==TRUE)
            {
               software_found=TRUE;
               number_of_packages++;
               strcpy(software_list[software_location].software_package,
                      package_name);
               software_list[software_location].total_copies=0;
               software_list[software_location].copies_in_use=0;
            }

   if ((software_found==FALSE)) software_location= -1;
```

```
   if ((software_found==FALSE) && (add_package==TRUE))
        system_error("software_list array limit reached.");

   return (software_location);
}

int find_package_in_usagelist(package_name,add_package)
    char *package_name;
    int  add_package;
{
    int software_location=0;
    int software_found=FALSE;
    while ((software_location<in_usage_number_of_packages)
           && (software_found==FALSE))
          {
             if (strcmp(usage_list[software_location].
             software_package,package_name)!=0) software_location++;
             else software_found=TRUE;
          }

    if ((software_found==FALSE) && (in_usage_number_of_packages<MAX_PACKAGES))
         if (add_package==TRUE)
            {
               software_found=TRUE;
               in_usage_number_of_packages++;
               strcpy(usage_list[software_location].software_package,
                      package_name);
               usage_list[software_location].copies=0;
               usage_list[software_location].concurrent=0;
               usage_list[software_location].checked=0;
               usage_list[software_location].time_used=0;
            }

   if ((software_found==FALSE)) software_location= -1;
   if ((software_found==FALSE) && (add_package==TRUE))
        system_error("software_list array limit reached.");

   return (software_location);
}

/*****************************************************************************/
/*                                                                           */
/* Function "find_node address"                                              */
/*                                                                           */
/* This function, given a node address and an addition flag, returns an      */
/* integer corresponding to its location in software_list.  If the node      */
/* can not be found, and the addition flag is TRUE, an attempt is made to    */
/* add the node address into the client_list.  If addition is attempted,     */
/* but all array elements are in use, a -1 is returned (indicating that the  */
/* client-PC will remain idle), and an error log is generated.               */
/*                                                                           */
/*****************************************************************************/

int find_node_address(node_address,add_node)
    char *node_address;
    int  add_node;
{
    int node_location=0;
    int node_found=FALSE;

    while ((node_location<number_of_clients) && (node_found==FALSE))

          {
             if (strcmp(client_list[node_location].node_address,
                 node_address)!=0) node_location++;
             else node_found=TRUE;



          }

    if ((node_found==FALSE) && (number_of_clients<MAX_CLIENTS))
         if (add_node==TRUE)
            {
               node_found=TRUE;
               number_of_clients++;
               strcpy(client_list[node_location].node_address,node_address);
               strcpy(client_list[node_location].software_package,IDLE_MACHINE);
               client_list[node_location].software_in_use= &software_list[0];
               software_list[0].copies_in_use++;
            }

    if ((node_found==FALSE)) node_location= -1;
    if ((node_found==FALSE) && (add_node==TRUE))
        system_error("client_list array limit reached.");

    return (node_location);
}


/*****************************************************************************/
/*                                                                           */
/* Function "read_software_configuration"                                    */
/*                                                                           */
/* This procedure attempts to read in the software configuration file        */
/* ("LICENSE_DB") and set the maximum number of usable copies for each       */
/* software package.  In the event that the file is not present, or the file */
/* is corrupt, a system error message is flagged and the function is         */
/* terminated with a return value of 1.  If the file is successfully read,   */
/* 0 is returned.                                                            */
/*                                                                           */
/*****************************************************************************/

int read_software_configuration()
{
   FILE *configuration_file;
   int   total_copies,exit_condition,elements_read,software_location;
   char  long_package_name[80],software_package[16],audit_description[60];

   if ((configuration_file=fopen(LICENSE_DB,"r+"))==NULL)
      {
         system_error("LICENSE_DB is not present.  Unlimited usage granted.");
         return(1);
      }
   long_package_name[0]='\0';

   elements_read=0;
   while (!feof(configuration_file))
         {
            elements_read=fscanf(configuration_file,"%2d%s",
                          &total_copies,long_package_name);
            if (elements_read==2)
                {
                   strncpy(software_package,long_package_name,15);
                   software_package[15]='\0';
                   sprintf(audit_description,"000.000.000.000 %-15s %3d",
                   software_package,total_copies);
                   generate_audit("SET_MAX",audit_description);
                   software_location=
                            find_software_package(software_package,TRUE);
                   if (software_location!= -1)
                      software_list[software_location].total_copies=total_copies;
                }
            else
               if (elements_read!=EOF)
```

```
                {
                  system_error("Unable to read LICENSE_DB configuration.");
                  return(1);
                }
        }
   fclose(configuration_file);
   return(0);
}


/*****************************************************************************/
/*                                                                           */
/* Function "rewrite_software_configuration"                                 */
/*                                                                           */
/* If the system adminisitrator opts to take the safe approach to updating   */
/* the LICENSE_DB configuration file via SET_MAX, this procedure is called   */
/* to rewrite the new software configuration to disk.  If the file can not   */
/* be rewritten we return a value of one, and if it can, we return a value   */
/* zero.  If the function fails, we proceed onward, hoping that the          */
/* administrator does not issue a RESTART command (which will force a        */
/* re-reading of the file).                                                  */
/*                                                                           */
/*****************************************************************************/

int rewrite_software_configuration()
{
   FILE *configuration_file;
   int counter;

   if ((configuration_file=fopen("temp.lab","w"))==NULL)
      {
        system_error("LICENSE_DB configuration can not be rewritten.");
        return(1);
      }
   for (counter=0;counter<number_of_packages;counter++)
        fprintf(configuration_file,"%2d %s\n",
                software_list[counter].total_copies,
                software_list[counter].software_package);
   fclose(configuration_file);
   if (rename("temp.lab\0",LICENSE_DB)== -1)
      {
        system_error("LICENSE_DB configuration can not be rewritten.");
        return(1);
      }
   return(0);
}


/*****************************************************************************/
/*                                                                           */
/* Function "regstr"                                                       */
/*                                                                           */
/* Passed the addresses of a node_address and a package_name, this routine   */
/* attempts to register that node for use of the requested program.  If the  */
/* node is still clinging to another package, that package is released, and  */
/* its current usage is decreased.  If the node is new, or the software is   */
/* not present in the "LICENSE_DB" file, there is the possibility that       */
/* either client_list or software_list is filled to capacity.  In that case, */
/* a two is returned to the calling routine.  If the requested package has   */
/* the maximum amount of copies in use, one is returned.  If the requested   */
/* package is already registered, a negative one is returned.  If the        */
/* package can be registered, the package's usage is incremented, and a zero */
/* is returned.                                                              */
/*                                                                           */
/*****************************************************************************/
```

```
int regstr(node_address,package_name,caller,copies_in_use)
    char *node_address;
    char *package_name;
    int  caller;
    int  *copies_in_use;
{
    int software_location=0;
    int node_location=0;
    char audit_description[40];

    struct SOFTWARE_BLOCK *software_to_check_in;
    struct SOFTWARE_BLOCK *software_to_check_out;

    software_location=find_software_package(package_name,TRUE);
    if (software_location==-1) return(2);

    node_location=find_node_address(node_address,TRUE);
    if (node_location==-1) return(2);

    software_to_check_out = &software_list[software_location];
    software_to_check_in  = client_list[node_location].software_in_use;

    if (strcmp(software_to_check_in->software_package,package_name)!=0)
       {
          if ((software_list[software_location].total_copies)>0)
             if (software_list[software_location].total_copies ==
                 software_list[software_location].copies_in_use) return(1);
          if (strcmp(client_list[node_location].software_package, "") != 0)
          {
             if (caller == CHECK_OUT || caller == CHECK_IN)
             {
                sprintf(audit_description,"%-15s %-15s",
                        node_address,software_to_check_in->software_package);
                generate_audit("CHECK_IN",audit_description);
              }
              (software_to_check_in->copies_in_use)--;
          }
          (software_to_check_out->copies_in_use)++;
          *copies_in_use = software_to_check_out->copies_in_use;
          strcpy(client_list[node_location].software_package,
                 software_to_check_out->software_package);
          client_list[node_location].software_in_use=software_to_check_out;

       }
   else return(-1);
   return(0);
}


/*****************************************************************************/
/*                                                                           */
/* Function "set_max"                                                        */
/*                                                                           */
/* Given a software info block, and a package addition flag, this routine    */
/* resets the site license maximum for the package.  If the package is new,  */
/* and the package array is filled, a negative one is returned.  If the site */
/* license maximum is successfully reset, a zero is returned, and if the     */
/* present number of copies in use for the package exceed the new maximum,   */
/* a one is returned and the package information is left unchanged.          */
/*                                                                           */
/*****************************************************************************/

int set_max(software_info,add_package)
    struct SOFTWARE_BLOCK *software_info;
    int add_package;
{
```

```
    int software_location,library_reply;

    software_location=find_software_package(software_info->software_package,
                                            add_package);
    if (software_location==-1) library_reply= -1;
    else
       {
          library_reply=0;
          if ((software_list[software_location].copies_in_use<=
               software_info->total_copies) || (software_info->total_copies==0))
               software_list[software_location].total_copies=
                              software_info->total_copies;
           else library_reply=1;
       }
    return (library_reply);
}


/*************************************************************************/
/*                                                                       */
/* Function "remove_node"                                                */
/*                                                                       */
/* In the event that a machine is powered-down while still technically   */
/* possessing a copy of a given software package, the library administrator */
/* can issue a remove_node command to check out the software associated with */
/* a given node.  The software data associated with that node (aka, the  */
/* number of copies in use) is updated, and the node is completely removed */
/* from the client list.  If the node is successfully removed from the client*/
/* list, a zero is returned.  If the specified node does not exist, a one is */
/* returned.                                                             */
/*                                                                       */
/*************************************************************************/

int remove_node(node_address)
    char *node_address;
{
    int ip1,ip2,ip3,ip4;
    int node_location;
    int counter,library_reply;
    char audit_description[40];

    sscanf( node_address,"%d.%d.%d.%d",&ip1,&ip2,&ip3,&ip4);
    sprintf(node_address,"%03d.%03d.%03d.%03d",ip1,ip2,ip3,ip4);
    node_location=find_node_address(node_address,FALSE);
    if (node_location!= -1)
    {
       if (strcmp(client_list[node_location].software_package, "") != 0)
       {
          (client_list[node_location].software_in_use->copies_in_use)--;
           sprintf(audit_description,"%-15s %-15s",
                   node_address,client_list[node_location].software_package);
           generate_audit("CHECK_IN",audit_description);
       }
       for (counter=node_location;(counter<number_of_clients-1); counter++)
             client_list[counter+0]=client_list[counter+1];
       number_of_clients--;
       library_reply=0;
    }
    else library_reply=1;
    return (library_reply);
}


/*************************************************************************/
/*                                                                       */
/* Function "restart_system"                                             */
```

```
/*                                                                       */
/* Sometimes the library administrator needs to completely restart the   */
/* server (as in the case of a power outage taking down a bulk of client */
/* machines.  This routine initializes the software list, the client list, */
/* and returns the result of reading the software configuration (a zero  */
/* indicates a success, and a one indicates a read error.                */
/*                                                                       */
/*************************************************************************/

int restart_system()

{
   initialize_software_list();
   initialize_client_list();
   return(read_software_configuration());
}


/*************************************************************************/
/*                                                                       */
/* function "read_library_image"                                         */
/*                                                                       */
/* In the event of a system crash, we can try to re-read our own audit file */
/* to rebuild out internal database.  If the audit file is there, and we are */
/* able to read it (and execute its contents), a FALSE is returned by this */
/* function (indicating that we do not need to restart the system from   */
/* scratch.  If the audit file isn't there, or it is corrupted, or the last */
/* entry in the audit file is a SHUTDOWN, we need to return a TRUE (to force */
/* system re-initialization).                                            */
/*                                                                       */
/*************************************************************************/

int read_library_image()

{
   char audit_entry[80];
   char audit_time[9],client_command[9],client_address[16],
        package_name[16],node_address[16];
   int  total_copies, copies_in_use,library_reply;
   struct SOFTWARE_BLOCK software_info;
   int  system_restart;

   initialize_software_list();
   initialize_client_list();
   system_restart=FALSE;
   while (!feof(audit_file))
   {
     if (fscanf(audit_file,"%s %s %s",audit_time,client_command,
                client_address)==3)
        {
           if (strcmp(client_command,"CHECK_OUT")==0)
             {
                fscanf(audit_file,"%s%d",package_name,&copies_in_use);
                library_reply=regstr(client_address,package_name,READ,
                                     &copies_in_use);
             }
           if (strcmp(client_command,"SET_MAX")==0)
             {
                fscanf(audit_file,"%s%d",package_name,&total_copies);
                strcpy(software_info.software_package,package_name);
                software_info.total_copies=total_copies;
                library_reply=set_max(&software_info,TRUE);
             }
           if (strcmp(client_command,"REMOVE")==0)
             {
                fscanf(audit_file,"%s",node_address);
```

```
                library_reply=remove_node(node_address);
              }
            if (strcmp(client_command,"SHUTDOWN")==0)
              {
                initialize_software_list();
                initialize_client_list();
              }
            if (strcmp(client_command,"CHECK_IN")==0)
                fscanf(audit_file,"%s",package_name);
            if (strcmp(client_command,"QPACKAGE")==0)
                fscanf(audit_file,"%s",package_name);
            if (strcmp(client_command,"QNODE")==0)
                fscanf(audit_file,"%s",node_address);
        }
    }
    if (strcmp(client_command,"SHUTDOWN")==0) system_restart=TRUE;
    return(system_restart);
}


/*****************************************************************************/
/*                                                                           */
/* Procedure "save_library_image"                                            */
/*                                                                           */
/* When a day change is detected, and a new audit file has been opened, we   */
/* need to save the current status of the librarian.  The current state is   */
/* defined by the current site requirements and the nodes/packages in use.   */
/* These are all output to the audit file; in the event of a librarian       */
/* failure, its current state can be regenerated for the day by re-reading   */
/* its own audit trail - even if "LICENSE_DB" was destroyed.                 */
/*                                                                           */
/*****************************************************************************/

save_library_image()
{
    int counter;
    time_t library_image_time;
    char current_time[9];

    time(&library_image_time);
    strncpy(current_time,(ctime(&library_image_time)+11),8);
    current_time[8]='\0';

    for (counter=0;counter<number_of_packages;counter++)
         fprintf(audit_file,"%-8s %-10s %-15s %-15s %3d\n",
                 current_time,"SET_MAX","000.000.000.000",
                 software_list[counter].software_package,
                 software_list[counter].total_copies);

    for (counter=0;counter<number_of_clients;counter++)
         fprintf(audit_file,"%-8s %-10s %-15s %-15s %3d\n",
                 current_time,"CHECK_OUT",
                 client_list[counter].node_address,
                 client_list[counter].software_in_use->software_package,
                 client_list[counter].software_in_use->copies_in_use);

}


/*****************************************************************************/
/*                                                                           */
/* Procedure "open_audit_file"                                               */
/*                                                                           */
/* This procedure opens the audit file.  The audit file name is in the       */
/* format day_mon_dd.audit.  Likewise, starting date is set to the format    */
/* day mon dd.  If an error is detected while attempting to open the audit   */
```

```
/* file, an error message is generated, and the server dies.                 */
/*                                                                           */
/*****************************************************************************/

open_audit_file()
{
    time_t starting_audit_time;
    int audit_file_present;
    int system_restart;

    number_of_system_errors=0;
    audit_file_present=TRUE;
    system_restart=FALSE;
    time(&starting_audit_time);
    strncpy(starting_date  ,ctime(&starting_audit_time)+0,10);
    strncpy(audit_file_name,ctime(&starting_audit_time)+4, 7);
    audit_file_name[7]='\0';
    starting_date [10]='\0';

    strncat(audit_file_name,ctime(&starting_audit_time),3);
    strcat(audit_file_name,".audit");
    audit_file_name[3]='_';
    audit_file_name[6]='_';
    if (audit_file_name[4]==' ') audit_file_name[4]='0';

    if ((audit_file=fopen(audit_file_name,"r"))!=NULL)
        {
            if (days_in_service==0)
                {
                    system_restart=read_library_image();
                    fclose(audit_file);
                }
            else system_error("Existing audit file detected.  Ignored");
        }
    else audit_file_present=FALSE;
    if ((audit_file=fopen(audit_file_name,"a"))==NULL)
        system_error("Unable to open audit file.");
    if ((audit_file_present==FALSE) && (days_in_service==0)) system_restart=TRUE;
    if (system_restart==TRUE) restart_system();
    days_in_service++;
}

void get_nextday(date)
struct DATE *date;
{
  int dd, mm, wday;

  dd = date->mday;
  mm = date->mon;
  wday = date->wday;

  dd ++;
  if (dd > no_of_days[mm-1])
  {
    dd = 1;
    mm ++;
  }

  wday ++;
  if (wday == 7)
    wday = 0;

  date->mday = dd;
  date->mon = mm;
  date->wday = wday;
}
```

```
void get_filename(filename, date)
char *filename;
struct DATE *date;
{
  int dd, mm, wday;

  dd = date->mday;
  mm = date->mon;
  wday = date->wday;

  strcpy(filename, month[mm-1]);
  filename[3] = '_';
  filename[4] = '\0';
  strcat(filename, day[dd-1]);
  filename[6] = '_';
  filename[7] = '\0';
  strcat(filename, weekday[wday]);
  filename[10] = '\0';
  strcat(filename, ".report");

}

int oneday_report(filename)
char *filename;
{
  FILE *report_file;
  char package_name[16];
  int elements_read, total_copies, concurrent, checked, time_used;
  int software_location;

  if ((report_file=fopen(filename,"r+"))==NULL) return;

  package_name[0]='\0';

  while (!feof(report_file)) {
       elements_read=fscanf(report_file,"%s %d %d %d %d\n", package_name,
                            &total_copies, &concurrent, &checked, &time_used);
       if (elements_read==5) {
                 package_name[15]='\0';
                 software_location=
                       find_package_in_usagelist(package_name,TRUE);
                 if (software_location != -1) {
                    usage_list[software_location].copies = total_copies;
                    if (concurrent > usage_list[software_location].concurrent)
                       usage_list[software_location].concurrent = concurrent;
                    if (checked > usage_list[software_location].checked)
                       usage_list[software_location].checked = checked;
                    if (time_used > usage_list[software_location].time_used)
                       usage_list[software_location].time_used = time_used;
                 }
       } else if (elements_read!=EOF) {
                   system_error("Unable to read LICENSE_DB configuration.");
                   return(1);
       }
   }
   fclose(report_file);
   return(0);
}

get_report(type, date)
int  type;
struct DATE  *date;
{
  char filename[18];
  int i;
```

```
  in_usage_number_of_packages = 0;

  switch (type)
  {
    case DAILY:
      get_filename(filename, date);
      oneday_report(filename);
      break;
    case WEEKLY:
      for (i = 0; i < 7; i++)
      {
        get_filename(filename, date);
        oneday_report(filename);
        get_nextday(date);
      }
      break;
    case MONTHLY:
      for (i = 0; i < no_of_days[date->mon -1]; i++)
      {
        get_filename(filename, date);
        oneday_report(filename);
        get_nextday(date);
      }
      break;
    case SEMESTER+1:
    case SEMESTER+3:
      for (i = 0; i < 120; i++)
      {
        get_filename(filename, date);
        oneday_report(filename);
        get_nextday(date);
      }
      break;
    case SEMESTER+2:
      for (i = 0; i < 100; i++)
      {
        get_filename(filename, date);
        oneday_report(filename);
        get_nextday(date);
      }
      break;
  }
}

/*****************************************************************************/
/*                                                                           */
/* Procedure "generate_audit"                                                */
/*                                                                           */
/* This procedure, given a node number and an action string, places an audit */
/* item in the audit file.  If the current date is no longer valid (we have  */
/* been running for more than a day), the current date is updated, the       */
/* output file is closed, and a new output file is generated.                */
/*                                                                           */
/* The format of the audit file typically is as follows:                     */
/*  (time)  (action)    (node)        (action parameters)                    */
/* -------- -------- --------------- ---------------------------------      */
/* 00:00:00 ACTION   000.000.000.000                                         */
/*                                                                           */
/* Variations do exist, however.                                             */
/*                                                                           */
/*****************************************************************************/

generate_audit(audit_action,audit_description)
   char *audit_action,*audit_description;
{
```

```
    time_t present_audit_time;
    char current_time[9],present_date[11];

    time(&present_audit_time);
    strncpy(present_date, ctime(&present_audit_time),10);
    strncpy(current_time,(ctime(&present_audit_time)+11),8);
    present_date[10]='\0';
    current_time[ 8]='\0';


    if (strncmp(starting_date,present_date,10)!=0)
       {
          fclose(audit_file);
          if (child_pid != 0) wait();

          if ((child_pid = fork() ) == 0)
             execl("gen_report", audit_file_name, 0);
          open_audit_file();
          save_library_image();
       }

    fprintf(audit_file,"%-8s %-10s %s\n",current_time,audit_action,
                                         audit_description);
    fflush(audit_file);
}


/*************************************************************************/
/*                                                                       */
/* Function "get_client_address"                                         */
/*                                                                       */
/* This function returns a sixteen character internet node_address given a */
/* client SVCXPRT transp pointer.  The node_address is in the format of  */
/* "000.000.000.000\0".                                                  */
/*                                                                       */
/*************************************************************************/

get_client_address(transp,node_address)
    register SVCXPRT *transp;
    char *node_address;
{
    char internet_address[4];
    bcopy(&transp->xp_raddr.sin_addr,internet_address,4);
    sprintf(node_address,"%03u.%03u.%03u.%03u",
                         (internet_address[0] & 0xff),
                         (internet_address[1] & 0xff),
                         (internet_address[2] & 0xff),
                         (internet_address[3] & 0xff));
}


/*************************************************************************/
/*                                                                       */
/* Function "get_client_arguments"                                       */
/*                                                                       */
/* This function, given a SVCXPRT transp pointer, an xdrproc_t pointer to */
/* the xdr handling routine, and a pointer to the return arguments, attempts */
/* an svc_getargs.  If the get fails, an error is generated and logged.  */
/*                                                                       */
/*************************************************************************/

int get_client_arguments(transp,xdr_routine,return_parameters)
    register SVCXPRT *transp;
    xdrproc_t xdr_routine;
    char *return_parameters;
```

```
{
    if (svc_getargs(transp,xdr_routine,return_parameters)==0)
       {
            system_error("Unable to get RPC arguments.");
            return (RPC_ERROR);
       }
    return (0);
}


/*************************************************************************/
/*                                                                       */
/* Function "send_client_reply"                                          */
/*                                                                       */
/* This function, given an SVCXPRT transp pointer, an xdrproc_t pointer to */
/* the xdr handling routine, and a pointer to the return arguments, attempts */
/* a svc_sendreply.  In the event that the sendreply can not occur, an error */
/* message is generated and logged.                                      */
/*                                                                       */
/*************************************************************************/

send_client_reply(transp,xdr_routine,library_reply)
     register SVCXPRT *transp;
     xdrproc_t xdr_routine;
     int *library_reply;

{
     if (svc_sendreply(transp,xdr_routine,library_reply)==0)
        {
           system_error("Unable to send RPC reply.");
           return;
        }
}


/*************************************************************************/
/*                                                                       */
/* Procedure "rpc_service"                                               */
/*                                                                       */
/* This is the heart of the APS librarian.  Once svc_run receives an RPC */
/* request, all essential information is passed to this routine for further */
/* processing.  We first get the client address of the rpc-originator, and */
/* then one huge case statement to handle the various RPC procedure numbers. */
/* There are three ways to exit this routine: via an error while getting RPC */
/* arguments, via the SHUTDOWN command (which exits the program completely), */
/* and via normal termination of an RPC procedure.  Errors during the sending */
/* of client replies are not handled specially; since they are the last  */
/* routine called before returning to svc_run, we merely record their result */
/* and continue.                                                         */
/*                                                                       */
/*************************************************************************/

rpc_service(rqstp,transp)
    register struct svc_req *rqstp;
    register SVCXPRT *transp;
{
    struct sockaddr_in client_addr;
    int  copies_in_use,library_reply,counter;
    int  node_location=0,software_location=0;
    int  ip1,ip2,ip3,ip4;

    char client_node[16];
    char node_address[16],package_name[16];
    char audit_description[40];
    char *node_data;
    time_t current_time;
```

```
struct tm *server_time;
struct DATE report_date;
struct SOFTWARE_BLOCK software_info;

get_client_address(transp,client_node);
switch (rqstp->rq_proc)
  {
     case SHUTDOWN:
        {
          if (get_client_arguments(transp,xdr_void,NULL)==RPC_ERROR) return;
          generate_audit("SHUTDOWN",client_node);
          fclose(audit_file);
          fclose(error_log);
          library_reply=0;
          send_client_reply(transp,xdr_int,&library_reply);
          svc_destroy(transp);
          exit(0);
        }

     case RESTART:
        {
          if (get_client_arguments(transp,xdr_void,NULL)==RPC_ERROR) return;
          generate_audit("RESTART",client_node);
          library_reply=restart_system();
          send_client_reply(transp,xdr_int,&library_reply);
          return;
        }

     case CHECK_IN:
        {
          if (get_client_arguments(transp,xdr_void,NULL)==RPC_ERROR) return;
          library_reply=regstr(client_node,IDLE_MACHINE,CHECK_IN,
                               &copies_in_use);
          if (library_reply==0)
          {
              sprintf(audit_description,"%-15s %-15s %3d",
                  client_node,IDLE_MACHINE,copies_in_use);
              generate_audit("CHECK_OUT",audit_description);
          }
          send_client_reply(transp,xdr_int,&library_reply);
          return;
        }

     case CHECK_OUT:
        {
          if (get_client_arguments(transp,xdr_string16,package_name)==
              RPC_ERROR) return;
          library_reply=regstr(client_node,package_name,CHECK_OUT,
                               &copies_in_use);
          sprintf(audit_description,"%-15s %-15s %3d",
                  client_node,package_name,copies_in_use);
          if (library_reply==0)
              generate_audit("CHECK_OUT",audit_description);
          send_client_reply(transp,xdr_int,&library_reply);
          return;
        }


     case REMOVE:
        {
          if (get_client_arguments(transp,xdr_string16,node_address)==
              RPC_ERROR) return;
          library_reply=remove_node(node_address);
          sprintf(audit_description,"%-15s %-15s",client_node,node_address);
          if (library_reply==0)
              generate_audit("REMOVE",audit_description);
```

```
          send_client_reply(transp,xdr_int,&library_reply);
          return;
        }

     case SET_MAX:
        {
          if (get_client_arguments(transp,xdr_software_block,&software_info)
                == RPC_ERROR) return;
          sprintf(audit_description,"%-15s %-15s %3d",client_node,
                software_info.software_package,software_info.total_copies);
          library_reply=set_max(&software_info,TRUE);
          if (library_reply==0)
             {
                library_reply=rewrite_software_configuration();
                generate_audit("SET_MAX",audit_description);
             }
          send_client_reply(transp,xdr_int,&library_reply);
          return;
        }

     case QNODE:
        {
          if (get_client_arguments(transp,xdr_string16,node_address)==
              RPC_ERROR) return;
          sscanf(node_address,"%d,%d,%d,%d",&ip1,&ip2,&ip3,&ip4);
          sprintf(node_address,"%03d.%03d.%03d.%03d",ip1,ip2,ip3,ip4);
          sprintf(audit_description,"%-15s %-15s",client_node,node_address);
          generate_audit("QNODE",audit_description);
          node_location=find_node_address(node_address,FALSE);
          if (node_location!=-1) strcpy(package_name,
              client_list[node_location].software_in_use->software_package);
          else strcpy(package_name,"Node not in use.");
          send_client_reply(transp,xdr_string16,package_name);
          return;
        }

     case QPACKAGE:
        {
          if (get_client_arguments(transp,xdr_string16,package_name)==
              RPC_ERROR) return;
          sprintf(audit_description,"%-15s %-15s",client_node,
                  package_name);
          generate_audit("QPACKAGE",audit_description);
          software_location=find_software_package(package_name,FALSE);
          if (software_location!=-1)
             {
                strncpy(software_info.software_package,
                        software_list[software_location].software_package,15);
                software_info.total_copies=
                        software_list[software_location].total_copies;
                software_info.copies_in_use=
                        software_list[software_location].copies_in_use;
             }
          else
             {
                strcpy(software_info.software_package,"Not in use.");
                software_info.total_copies=0;
                software_info.copies_in_use=0;
             }
          send_client_reply(transp,xdr_software_block,&software_info);
          return;
        }

     case QUSERS:
        {
          if (get_client_arguments(transp,xdr_void,NULL)==RPC_ERROR) return;
```

```
            generate_audit("QUSERS",client_node);
            software_info.software_package[0]='\0';
            software_info.total_copies=number_of_clients;
            software_info.copies_in_use=software_list[0].copies_in_use;
            send_client_reply(transp,xdr_software_block,&software_info);
            return;
          }

        case QTIME:
          {
            if (get_client_arguments(transp,xdr_void,NULL)==RPC_ERROR) return;
            time(&current_time);
            generate_audit("QTIME",client_node);
            server_time=localtime(&current_time);
            send_client_reply(transp,xdr_date,server_time);
            return;
          }

        case QNODES:
          {
            if (get_client_arguments(transp,xdr_void,NULL)==RPC_ERROR) return;
            generate_audit("QNODES",client_node);
            node_data=client_list[0].node_address;
            send_client_reply(transp,xdr_node_array,&node_data);
            return;
          }

        case QLICENSE:
          {
            if (get_client_arguments(transp,xdr_void,NULL)==RPC_ERROR) return;
            generate_audit("QLICENSE",client_node);
            node_data=software_list[0].software_package;
            send_client_reply(transp,xdr_package_array,&node_data);
            return;
          }

        case DAILY:
          {
           int i;
           if (get_client_arguments(transp,xdr_mydate,&report_date)==RPC_ERROR)
              return;
           get_report(DAILY, &report_date);

             node_data=usage_list[0].software_package;
             send_client_reply(transp,xdr_usage_array,&node_data);
             return;
          }

        case WEEKLY:
          {
           int i;
           if (get_client_arguments(transp,xdr_mydate,&report_date)==RPC_ERROR)
              return;
           get_report(WEEKLY, &report_date);

             node_data=usage_list[0].software_package;
             send_client_reply(transp,xdr_usage_array,&node_data);
             return;
          }

        case MONTHLY:
          {
           int i;
           if (get_client_arguments(transp,xdr_mydate,&report_date)==RPC_ERROR)
              return;
           get_report(MONTHLY, &report_date);
```

```
             node_data=usage_list[0].software_package;
             send_client_reply(transp,xdr_usage_array,&node_data);
             return;
          }

        case SEMESTER+1:
        case SEMESTER+2:
        case SEMESTER+3:
          {
           int i;
           if (get_client_arguments(transp,xdr_mydate,&report_date)==RPC_ERROR)
              return;
           get_report(rqstp->rq_proc, &report_date);

             node_data=usage_list[0].software_package;
             send_client_reply(transp,xdr_usage_array,&node_data);
             return;
          }
        default:
          {
             svcerr_noproc(transp);
             return;
          }
      }
}


/*************************************************************************/
/*                                                                       */
/* procedure "main"                                                      */
/*                                                                       */
/* This routine opens the error log file, opens the audit file, and then */
/* attempts to create a tcp connection.  If it fails in creating the tcp */
/* connection, the program terminates.  It then registers itself as an RPC */
/* server, and goes into an infinite loop, waiting for RPC responses to  */
/* arrive.                                                               */
/*                                                                       */
/*                                                                       */
/*************************************************************************/

main()
{
   register SVCXPRT *transp;

   open_error_log();
   open_audit_file();
   if ((transp=svctcp_create(RPC_ANYSOCK,BUFSIZ,BUFSIZ))==NULL)
       {
          system_error("Unable to create TCP server.");
          exit(1);
       }
   pmap_unset(SOFTLIB,SOFTVERS);
   if (!svc_register(transp, (u_long) SOFTLIB,
                             (u_long) SOFTVERS,
                              rpc_service,IPPROTO_TCP))
      {
         system_error("Unable to register service.");
         exit(1);
      }

    if (fork()) exit(0);  /* run as daemon */
    close(0);
    close(1);
    close(2);
    setpgrp();  /* detach from process group */
```

```
        svc_run();
    }
```

```
static char sccsid[] = "@(#)license.c   July, 92";
/*
 * COMPONENT_NAME: X11
 *
 * Program:        XFACE.C
 *
 * X Windows/Motif interface for NLMS management portion.
 *
*/
```

```
/*---------------------------------------------------------------
**      Include Files
*/
#define fd_set

#include "libraria.h"
#include <stdio.h>
#include <string.h>
#include <fcntl.h>

#include <errno.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <X11/X.h>
#include <X11/Xlib.h>
#include <X11/Xatom.h>
#include <X11/Intrinsic.h>
#include <X11/Shell.h>
#include <X11/Core.h>

#include <Xm/Xm.h>
#include <Xm/List.h>
#include <Xm/CascadeB.h>
#include <Xm/DialogS.h>
#include <Xm/BulletinB.h>
#include <Xm/MainW.h>
#include <Xm/MessageB.h>
#include <Xm/PushB.h>
#include <Xm/RowColumn.h>
#include <Xm/SelectioB.h>
```

```
/*-------------------------------------------------------------
**      Global Variables
*/
#define MAX_COPIES      200

#define ABOUT           232
#define HELP            230

#define QUIT                    999
#define DIALOG_HELP             305
#define DIALOG_CANCEL           306
#define DIALOG_OK               307
#define NOMATCH                 310
#define DESTROY1                321
#define DESTROY2                322

#define SUCCESS         0
#define WRONGFORMAT     1

#define NODEFILE                "nodeslist.dat"

XmStringCharSet charset = XmSTRING_DEFAULT_CHARSET;
                                /* used to set up XmStrings */

static int no_nodes;
static char Nodes[MAX_CLIENTS][15];
static char CharNodesList[MAX_CLIENTS][30];
XmString NodesList[MAX_CLIENTS];

static int no_packages;
static char s_m_package[16];
static char CharPackagesName[MAX_PACKAGES][16];
static char CharPackagesList[MAX_PACKAGES][20];
XmString PackagesList[MAX_PACKAGES];
XmString PackagesName[MAX_PACKAGES];

static char CharLicenseList[MAX_PACKAGES][24];
XmString LicenseList[MAX_PACKAGES];

static char CharUsageList[MAX_PACKAGES][75];
XmString UsageList[MAX_PACKAGES];

static char CharQNodesList[MAX_CLIENTS][33];
XmString QNodesList[MAX_CLIENTS];

char Error[128];
char server[] = "apsrisc";

static void DialogAcceptCB (Widget, caddr_t, caddr_t);
```

```
int date_disect(str, ddp, mmp, yyp, weekdayp)
char  *str;
int   *ddp, *mmp, *yyp, *weekdayp;

{
  char str1[3];
  int temp;

  strncpy(str1, str, 2);
  str1[3] = '\0';
  *mmp = atoi(str1);

  str += 3;
  strncpy(str1, str, 2);
  str1[3] = '\0';
  *ddp = atoi(str1);

  str += 3;
  strncpy(str1, str, 2);
  str1[3] = '\0';
  *yyp = atoi(str1);

  if (*mmp <= 0 || *ddp <= 0 || *yyp <= 0)
    return WRONGFORMAT;

  temp = *ddp - 1 + first_day_of_month[*mmp-1];
  *weekdayp = temp - temp / 7 * 7;
  return SUCCESS;
}


int month_disect(str, ddp, mmp, yyp, weekdayp)
char  *str;
int   *ddp, *mmp, *yyp, *weekdayp;

{
  char str1[3];
  int temp;

  *ddp = 1;

  strncpy(str1, str, 2);
  str1[3] = '\0';
  *mmp = atoi(str1);

  str += 3;
  strncpy(str1, str, 2);
  str1[3] = '\0';
  *yyp = atoi(str1);

  if (*mmp <= 0 || *ddp <= 0 || *yyp <= 0)
    return WRONGFORMAT;

  *weekdayp = first_day_of_month[*mmp-1];
  return SUCCESS;
}
```

```
/*-------------------------------------------------------------
**      ReadLocalNodes
**              Read in nodes list and their names.
*/
int ReadLocalNodes()
{
        FILE    *nodefile;
        int     i, j;
        char    addlist[15], namelist[10];
        if ((nodefile = fopen(NODEFILE, "r")) == NULL) {
                printf("can't open nodeslist.dat\n");
                exit(1);
        }
        i = 0;
        while (fscanf(nodefile, "%s %s", addlist, namelist) == 2) {
                sprintf(Nodes[i], "%-14s\0", addlist);
                sprintf(CharNodesList[i], "%-14s %-14s\0", addlist, namelist);
                i++;
        }
        fclose(nodefile);

        for (j=0; j<i; j++) {
                NodesList[j] = (XmString) XmStringCreateLtoR (
                        CharNodesList[j], charset);
        }
        NodesList[i] = NULL;

        return i;
}
```

```
/*-------------------------------------------------------------
**      GetPackages
**              Make a RPC call to get the list of packages for later display.
*/
int GetPackages()
{
     int counter, library_function, librarian_result;
     char package_name[16], total_copies[3];
     char *node_data;

     library_function = QLICENSE;
     node_data=software_list[0].software_package;
     call_rpc(server,library_function,xdr_void,NULL,
              xdr_package_array,&node_data);

     for (counter=0;counter<number_of_packages;counter++)
        {
          sprintf(CharPackagesName[counter], "%s\0",
               software_list[counter].software_package);
          sprintf(CharPackagesList[counter], "%-15s %3d\0",
               software_list[counter].software_package,
               software_list[counter].total_copies);
        }
     for (counter=0;counter<number_of_packages;counter++) {
          PackagesName[counter] = (XmString) XmStringCreateLtoR (
               CharPackagesName[counter], charset);
          PackagesList[counter] = (XmString) XmStringCreateLtoR (
               CharPackagesList[counter], charset);
        }
     PackagesList[number_of_packages] = NULL;

     return(number_of_packages);
}
```

```
/*-------------------------------------------------------------
**      DestoryCB
*/
void DestroyCB (w, client_data, call_data)
Widget          w;
caddr_t         client_data;
caddr_t         call_data;
{
Widget pp;
        switch ( (int) client_data)
        {
          case DESTROY1:
                XtDestroyWidget (w);
                break;
          case DESTROY2:
                pp = XtParent(w);
                XtDestroyWidget (w);
                XtDestroyWidget (pp);
                break;
        }
}
```

```
/*-------------------------------------------------------------
**      CallRPC
*/
void CallRPC (w, client_data, call_data)
Widget          w;              /*  widget id           */
caddr_t         client_data;    /*  data from application   */
caddr_t         call_data;      /*  data from widget class  */
{
        register int ac;                /* arg count                */
        Arg al[10];                     /* arg list                 */
        char msg[150];
        Widget          msgD, kid;
        Widget          Q_Nodes_Result_dialog;
        Widget          Q_License_Result_dialog;
        Widget          Q_Usage_Result_dialog;


     XmSelectionBoxCallbackStruct  *cb;

     int i, counter, library_function, librarian_result;
     char package_name[16];
     char *node_data;

     struct tm server_time;
     struct SOFTWARE_BLOCK package_info;

        library_function = (int)client_data;

        switch (library_function)
        {
        case QNODE:
          cb = (XmSelectionBoxCallbackStruct *)call_data;
          i = 0;
          while (! (XmStringCompare(cb->value, NodesList[i])) )
                i++;
          call_rpc(server,library_function,xdr_string16,Nodes[i],
                xdr_string16,package_name);
          sprintf(msg,"Node Number    : %s\nSoftware Package: %s\n\0",
                Nodes[i], package_name);
          ac = 0;
          XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
          XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
          ac++;
          msgD = XmCreateMessageDialog(w,
                "      Query    Node      Result      ", al, ac);
          XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
          XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
          kid = XmSelectionBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
          XtUnmanageChild(kid);
          XtManageChild (msgD);

          break;

        case QPACKAGE:
          cb = (XmSelectionBoxCallbackStruct *)call_data;
          i = 0;
          while (! (XmStringCompare(cb->value, PackagesName[i])) )
                i++;
          call_rpc(server,library_function,xdr_string16,CharPackagesName[i],
                xdr_software_block,&package_info);
          sprintf(msg,
        "Software Package: %s\nCopies Available: %2d\nCopies in Use    : %2d\n\0",
                        package_info.software_package, package_info.total_copies,
                        package_info.copies_in_use);
          ac = 0;
```

```
            XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
            XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
            ac++;
            msgD = XmCreateMessageDialog(w,
                                "   Query  Package  Result   ", al, ac);
            XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
            XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
            kid = XmSelectionBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
            XtUnmanageChild(kid);
            XtManageChild (msgD);
            break;

          case CHECK_IN:
            call_rpc(server,library_function,xdr_void,NULL,
                        xdr_int,&librarian_result);
            sprintf(msg,"Librarian Result is %d\n\0",librarian_result);
            ac = 0;
            XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
            XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
            ac++;
            msgD = XmCreateMessageDialog(w, "  Check In Result  ", al, ac);
            XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
            XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
            kid = XmSelectionBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
            XtUnmanageChild(kid);
            XtManageChild (msgD);
            break;

          case CHECK_OUT:
            cb = (XmSelectionBoxCallbackStruct *)call_data;
            i = 0;
            while (! (XmStringCompare(cb->value, PackagesName[i])) )
                  i++;
            call_rpc(server,library_function,xdr_string16,CharPackagesName[i],
                        xdr_int,&librarian_result);
            sprintf(msg,"Librarian Result is %d\n\0",librarian_result);
            ac = 0;
            XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
            XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
            ac++;
            msgD = XmCreateMessageDialog(w, "  Check Out Result  ", al, ac);
            XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY2);
            XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
            kid = XmSelectionBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
            XtUnmanageChild(kid);
            XtManageChild (msgD);
            break;

          case QTIME:
            call_rpc(server,library_function,xdr_void,NULL,
                xdr_date,&server_time);
            sprintf(msg,
    "Current date is %3s %02d-%02d-%04d\nCurrent time is %02d:%02d:%02d.00\n\0",
                                    weekday[server_time.tm_wday],
                                    server_time.tm_mon+1,
                                    server_time.tm_mday,
                                    server_time.tm_year+1900,
                                    server_time.tm_hour,
                                    server_time.tm_min,
                                    server_time.tm_sec);
            ac = 0;
            XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
            XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
            ac++;
            msgD = XmCreateMessageDialog(XtParent(w), "  Query  Time  Result  ",
                                    al,ac);
```

```
            XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
            XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
            kid = XmSelectionBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
            XtUnmanageChild(kid);
            XtManageChild (msgD);
            break;

          case QUSERS:
            call_rpc(server,library_function,xdr_void,NULL,xdr_software_block,
                        &package_info);
            sprintf(msg,
"Machines Registered:%3d\nMachines     in Use:%3d\nMachines       Idle:%3d\n\0",
package_info.total_copies, package_info.total_copies-package_info.copies_in_use,
package_info.copies_in_use);

            ac = 0;
            XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
            XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
            ac++;
            msgD = XmCreateMessageDialog(XtParent(w),"  Query  Users  Result  ",
                                    al,ac);
            XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
            XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
            kid = XmSelectionBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
            XtUnmanageChild(kid);
            XtManageChild (msgD);
            break;

          case QNODES:
            node_data=client_list[0].node_address;
            call_rpc(server,library_function,xdr_void,NULL,xdr_node_array,
                  &node_data);
            librarian_result=0;

            for (counter=0;counter<number_of_clients;counter++)
                  sprintf(CharQNodesList[counter], "%-15s %-13s\0",
                        client_list[counter].node_address,
                        client_list[counter].software_package);
            for (counter=0;counter<number_of_clients;counter++)
                  QNodesList[counter] = XmStringCreateLtoR (
                                    CharQNodesList[counter], charset);
            QNodesList[number_of_clients] = NULL;

            ac = 0;
            XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
            XtSetArg(al[ac], XmNlistLabelString, XmStringCreateLtoR(
                              "Machine         Packages", charset));
            ac ++;
            XtSetArg(al[ac], XmNlistItems, QNodesList); ac++;
            XtSetArg(al[ac], XmNlistItemCount, number_of_clients); ac++;
            XtSetArg(al[ac], XmNvisibleItemCount,
                        (XtArgVal) number_of_clients); ac++;
            XtSetArg(al[ac], XmNlistVisibleItemCount, 16); ac++;

            XtSetArg(al[ac], XmNselectionLabelString, NULL); ac++;
            Q_Nodes_Result_dialog = XmCreateSelectionDialog(XtParent(w),
                        "  Query  Nodes  Result  ", al, ac);

            XtAddCallback(Q_Nodes_Result_dialog, XmNokCallback, DestroyCB,
                              DESTROY1);
            XtAddCallback(Q_Nodes_Result_dialog, XmNhelpCallback, DialogAcceptCB,
                              DIALOG_HELP);
            kid = XmSelectionBoxGetChild(Q_Nodes_Result_dialog,
                                    XmDIALOG_CANCEL_BUTTON);
            XtUnmanageChild(kid);
            kid = XmSelectionBoxGetChild(Q_Nodes_Result_dialog,
```

```
                                        XmDIALOG_SELECTION_LABEL);
        XtUnmanageChild(kid);
        kid = XmSelectionBoxGetChild(Q_Nodes_Result_dialog,
                                        XmDIALOG_TEXT);
        XtUnmanageChild(kid);
        XtManageChild(Q_Nodes_Result_dialog);

        break;

      case QLICENSE:
        node_data=software_list[0].software_package;
        call_rpc(server,library_function,xdr_void,NULL,
              xdr_package_array,&node_data);
        librarian_result=0;

        for (counter=0;counter<number_of_packages;counter++)
           sprintf(CharLicenseList[counter], "%-15s %3d %3d\0",
                     software_list[counter].software_package,
                     software_list[counter].total_copies,
                     software_list[counter].copies_in_use);
        for (counter=0;counter<number_of_packages;counter++)
              LicenseList[counter] = (XmString) XmStringCreateLtoR (
                     CharLicenseList[counter], charset);
        LicenseList[number_of_packages] = NULL;

        ac = 0;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        XtSetArg(al[ac], XmNlistLabelString, XmStringCreateLtoR(
                              "Packages      Total Use", charset));
        ac ++;
        XtSetArg(al[ac], XmNlistItems, LicenseList); ac++;
        XtSetArg(al[ac], XmNlistItemCount, number_of_packages); ac++;
        XtSetArg(al[ac], XmNvisibleItemCount,
                        (XtArgVal) number_of_packages); ac++;
        XtSetArg(al[ac], XmNlistVisibleItemCount, 16); ac++;

        XtSetArg(al[ac], XmNselectionLabelString, NULL); ac++;
        Q_License_Result_dialog = XmCreateSelectionDialog(XtParent(w),
              "  Query  License  Result ", al, ac);

        XtAddCallback(Q_License_Result_dialog, XmNokCallback, DestroyCB,
                              DESTROY1);
        XtAddCallback(Q_License_Result_dialog, XmNhelpCallback,DialogAcceptCB,
                              DIALOG_HELP);
        kid = XmSelectionBoxGetChild(Q_License_Result_dialog,
                                        XmDIALOG_CANCEL_BUTTON);
        XtUnmanageChild(kid);
        kid = XmSelectionBoxGetChild(Q_License_Result_dialog,
                                        XmDIALOG_SELECTION_LABEL);
        XtUnmanageChild(kid);
        kid = XmSelectionBoxGetChild(Q_License_Result_dialog,
                                        XmDIALOG_TEXT);
        XtUnmanageChild(kid);
        XtManageChild(Q_License_Result_dialog);
        break;

      case REMOVE:
        cb = (XmSelectionBoxCallbackStruct *)call_data;
        i = 0;
        while (! (XmStringCompare(cb->value, NodesList[i])) )
              i++;
        call_rpc(server,library_function,xdr_string16, Nodes[i],
              xdr_int,&librarian_result);
        sprintf(msg,"Librarian Result is %d\n",librarian_result);
        ac = 0;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
```

```
        XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
        ac++;
        msgD = XmCreateMessageDialog(w,
              "  Remove  Node  Result  ", al, ac);
        XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
        XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
        kid = XmSelectionBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
        XtUnmanageChild(kid);
        XtManageChild (msgD);
        break;

      case SET_MAX:
        {
        char *p;

        cb = (XmSelectionBoxCallbackStruct *)call_data;
        XmStringGetLtoR(cb->value, charset, &p);
        i = atoi(p);
        strcpy(package_info.software_package, s_m_package);
        package_info.total_copies=i;
        package_info.copies_in_use=0;
        call_rpc(server,library_function,xdr_software_block,&package_info,
              xdr_int,&librarian_result);
        sprintf(msg,"Librarian Result is %d\n",librarian_result);
        ac = 0;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
        ac++;
        msgD = XmCreateMessageDialog(XtParent(XtParent(w)),
              "    Set  MAX  Result   ", al, ac);
        XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY2);
        XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
        kid = XmSelectionBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
        XtUnmanageChild(kid);
        XtManageChild (msgD);
        break;
        }

      case DAILY:
        {
        char *p, title[50];
        int dd, mm, yy,  dateinputok,  weekday;
        struct DATE  report_date;

        cb = (XmSelectionBoxCallbackStruct *)call_data;
        XmStringGetLtoR(cb->value, charset, &p);
        dateinputok = date_disect(p, &dd, &mm, &yy, &weekday);
        if (dateinputok != SUCCESS)
        {
          sprintf(msg,
"Date has not been entered correctly\nPlease use the format of MM-DD-YY\0");
          ac = 0;
          XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
          XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
          ac++;
          msgD = XmCreateErrorDialog(XtParent(w),
                        "        Warning        ", al, ac);
          XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
          XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
          kid = XmMessageBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
          XtUnmanageChild(kid);
          XtManageChild (msgD);
          break;
        }
        report_date.mday = dd;
        report_date.mon = mm;
```

```
        report_date.year = yy;
        report_date.wday = weekday;
        sprintf(title, "Daily Usage Report   (for the day of %s)     ", p);

        node_data=usage_list[0].software_package;

        call_rpc(server,library_function,xdr_mydate,&report_date,
              xdr_usage_array,&node_data);
        librarian_result=0;

        for (counter=0;counter<in_usage_number_of_packages;counter++)
         sprintf(CharUsageList[counter], "%-15s %5d %14d %13d %19d\0",
                      usage_list[counter].software_package,
                      usage_list[counter].copies,
                      usage_list[counter].concurrent,
                      usage_list[counter].checked,
                      usage_list[counter].time_used/60);
        for (counter=0;counter<in_usage_number_of_packages;counter++)
              UsageList[counter] = (XmString) XmStringCreateLtoR (
                      CharUsageList[counter], charset);
        UsageList[in_usage_number_of_packages] = NULL;

        ac = 0;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        XtSetArg(al[ac], XmNlistLabelString, XmStringCreateLtoR(
            "Package      licenses   concurrently   checked/day   used(in minutes)/day", charset));
        ac ++;
        XtSetArg(al[ac], XmNlistItems, UsageList); ac++;
        XtSetArg(al[ac], XmNlistItemCount, in_usage_number_of_packages); ac++;
        XtSetArg(al[ac], XmNvisibleItemCount,
                      (XtArgVal) in_usage_number_of_packages); ac++;
        XtSetArg(al[ac], XmNlistVisibleItemCount, 16); ac++;

        XtSetArg(al[ac], XmNselectionLabelString, NULL); ac++;
        Q_Usage_Result_dialog = XmCreateSelectionDialog(XtParent(w),
              title, al, ac);

        XtAddCallback(Q_Usage_Result_dialog, XmNokCallback, DestroyCB,
                      DESTROY1);
        XtAddCallback(Q_Usage_Result_dialog, XmNhelpCallback, DialogAcceptCB,
                      DIALOG_HELP);
        kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                     XmDIALOG_CANCEL_BUTTON);
        XtUnmanageChild(kid);
        kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                     XmDIALOG_SELECTION_LABEL);
        XtUnmanageChild(kid);
        kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                     XmDIALOG_TEXT);
        XtUnmanageChild(kid);
        XtManageChild(Q_Usage_Result_dialog);
        break;
        }

      case WEEKLY:
        {
        char *p, title[50];
        int dd, mm, yy,  dateinputok,  weekday;
        struct DATE  report_date;

        cb = (XmSelectionBoxCallbackStruct *)call_data;
        XmStringGetLtoR(cb->value, charset, &p);
        dateinputok = date_disect(p, &dd, &mm, &yy, &weekday);
        if (dateinputok != SUCCESS)
        {
          sprintf(msg,
```

```
"Date has not been entered correctly\nPlease use the format of MM-DD-YY\0");
          ac = 0;
          XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
          ac++;
          msgD = XmCreateErrorDialog(XtParent(w),
                        "        Warning        ", al, ac);
          XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
          XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
          kid = XmMessageBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
          XtUnmanageChild(kid);
          XtManageChild (msgD);
          break;
        }
        if (weekday != 1)
        {
          sprintf(msg,
"Weekly report requires to enter that Monday\nin the format of MM-DD-YY\0");
          ac = 0;
          XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
          XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
          ac++;
          msgD = XmCreateErrorDialog(XtParent(w),
                        "        Warning        ", al, ac);
          XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
          XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
          kid = XmMessageBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
          XtUnmanageChild(kid);
          XtManageChild (msgD);
          break;
        }
        report_date.mday = dd;
        report_date.mon = mm;
        report_date.year = yy;
        report_date.wday = weekday;
        sprintf(title, "Daily Usage Report   (over the week of %s)   ", p);

        node_data=usage_list[0].software_package;

        call_rpc(server,library_function,xdr_mydate,&report_date,
              xdr_usage_array,&node_data);
        librarian_result=0;

        for (counter=0;counter<in_usage_number_of_packages;counter++)
         sprintf(CharUsageList[counter], "%-15s %5d %14d %13d %19d\0",
                      usage_list[counter].software_package,
                      usage_list[counter].copies,
                      usage_list[counter].concurrent,
                      usage_list[counter].checked,
                      usage_list[counter].time_used/60);
        for (counter=0;counter<in_usage_number_of_packages;counter++)
              UsageList[counter] = (XmString) XmStringCreateLtoR (
                      CharUsageList[counter], charset);
        UsageList[in_usage_number_of_packages] = NULL;

        ac = 0;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        XtSetArg(al[ac], XmNlistLabelString, XmStringCreateLtoR(
            "Package      licenses   concurrently   checked/day   used(in minutes)/day", charset));
        ac ++;
        XtSetArg(al[ac], XmNlistItems, UsageList); ac++;
        XtSetArg(al[ac], XmNlistItemCount, in_usage_number_of_packages); ac++;
        XtSetArg(al[ac], XmNvisibleItemCount,
                      (XtArgVal) in_usage_number_of_packages); ac++;
        XtSetArg(al[ac], XmNlistVisibleItemCount, 16); ac++;

        XtSetArg(al[ac], XmNselectionLabelString, NULL); ac++;
```

```
            Q_Usage_Result_dialog = XmCreateSelectionDialog(XtParent(w),
                  title, al, ac);

            XtAddCallback(Q_Usage_Result_dialog, XmNokCallback, DestroyCB,
                              DESTROY1);
            XtAddCallback(Q_Usage_Result_dialog, XmNhelpCallback, DialogAcceptCB,
                              DIALOG_HELP);
            kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                    XmDIALOG_CANCEL_BUTTON);
            XtUnmanageChild(kid);
            kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                    XmDIALOG_SELECTION_LABEL);
            XtUnmanageChild(kid);
            kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                    XmDIALOG_TEXT);
            XtUnmanageChild(kid);
            XtManageChild(Q_Usage_Result_dialog);
            break;
            }

          case MONTHLY:
            {
            char *p, title[50];
            int dd, mm, yy,  dateinputok,  weekday;
            struct DATE  report_date;

            cb = (XmSelectionBoxCallbackStruct *)call_data;
            XmStringGetLtoR(cb->value, charset, &p);
            printf("input string is %s\n", p);
            dateinputok = month_disect(p, &dd, &mm, &yy, &weekday);
            if (dateinputok != SUCCESS)
            {
              sprintf(msg,
    "Date has not been entered correctly\nPlease use the format of MM-YY\0");
              ac = 0;
              XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
              XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
              ac++;
              msgD = XmCreateErrorDialog(XtParent(w),
                        "         Warning         ", al, ac);
              XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
              XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
              kid = XmMessageBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
              XtUnmanageChild(kid);
              XtManageChild (msgD);
              break;
            }
            report_date.mday = dd;
            report_date.mon = mm;
            report_date.year = yy;
            report_date.wday = weekday;
            sprintf(title, "Daily Usage Report   (over the month of %s)   ", p);

            node_data=usage_list[0].software_package;

            call_rpc(server,library_function,xdr_mydate,&report_date,
                  xdr_usage_array,&node_data);
            librarian_result=0;

            for (counter=0;counter<in_usage_number_of_packages;counter++)
             sprintf(CharUsageList[counter], "%-15s %5d %14d %13d %19d\0",
                        usage_list[counter].software_package,
                        usage_list[counter].copies,
                        usage_list[counter].concurrent,
                        usage_list[counter].checked,
                        usage_list[counter].time_used/60);
```

```
            for (counter=0;counter<in_usage_number_of_packages;counter++)
                  UsageList[counter] = (XmString) XmStringCreateLtoR (
                        CharUsageList[counter], charset);
            UsageList[in_usage_number_of_packages] = NULL;

            ac = 0;
            XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
            XtSetArg(al[ac], XmNlistLabelString, XmStringCreateLtoR(
                  "Package      licenses   concurrently   checked/day   used(in minutes)/day", charset));
            ac ++;
            XtSetArg(al[ac], XmNlistItems, UsageList); ac++;
            XtSetArg(al[ac], XmNlistItemCount, in_usage_number_of_packages); ac++;
            XtSetArg(al[ac], XmNvisibleItemCount,
                        (XtArgVal) in_usage_number_of_packages); ac++;
            XtSetArg(al[ac], XmNlistVisibleItemCount, 16); ac++;

            XtSetArg(al[ac], XmNselectionLabelString, NULL); ac++;
            Q_Usage_Result_dialog = XmCreateSelectionDialog(XtParent(w),
                  title, al, ac);

            XtAddCallback(Q_Usage_Result_dialog, XmNokCallback, DestroyCB,
                              DESTROY1);
            XtAddCallback(Q_Usage_Result_dialog, XmNhelpCallback, DialogAcceptCB,
                              DIALOG_HELP);
            kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                    XmDIALOG_CANCEL_BUTTON);
            XtUnmanageChild(kid);
            kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                    XmDIALOG_SELECTION_LABEL);
            XtUnmanageChild(kid);
            kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                    XmDIALOG_TEXT);
            XtUnmanageChild(kid);
            XtManageChild(Q_Usage_Result_dialog);
            break;
            }

          case SEMESTER:
            {
            char *p, title[50];
            int dd, mm, yy,  weekday, semester;
            struct DATE  report_date;

            cb = (XmSelectionBoxCallbackStruct *)call_data;
            XmStringGetLtoR(cb->value, charset, &p);
            semester = atoi(p);
            if (semester != 1 && semester != 2 && semester != 3)
            {
              sprintf(msg,
    "Date has not been entered correctly\nPlease enter 1 for Spring, or 2 for Summer, or 3 for Fall\0");
              ac = 0;
              XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
              XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
              ac++;
              msgD = XmCreateErrorDialog(XtParent(w),
                        "         Warning         ", al, ac);
              XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
              XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
              kid = XmMessageBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
              XtUnmanageChild(kid);
              XtManageChild (msgD);
              break;
            }

            switch (semester) {
              case 1:
```

```
            date_disect("01-15-92", &dd, &mm, &yy, &weekday);
            sprintf(title, "Daily Usage Report   (over Spring semester)   ");
            break;
          case 2:
            date_disect("05-15-92", &dd, &mm, &yy, &weekday);
            sprintf(title, "Daily Usage Report   (over Summer sections)   ");
            break;
          case 3:
            date_disect("09-01-92", &dd, &mm, &yy, &weekday);
            sprintf(title, "Daily Usage Report   (over Fall semester)   ");
        }

        report_date.mday = dd;
        report_date.mon = mm;
        report_date.year = yy;
        report_date.wday = weekday;

        node_data=usage_list[0].software_package;

        call_rpc(server,library_function+semester,xdr_mydate,&report_date,
            xdr_usage_array,&node_data);
        librarian_result=0;

        for (counter=0;counter<in_usage_number_of_packages;counter++)
         sprintf(CharUsageList[counter], "%-15s %5d %14d %13d %19d\0",
                    usage_list[counter].software_package,
                    usage_list[counter].copies,
                    usage_list[counter].concurrent,
                    usage_list[counter].checked,
                    usage_list[counter].time_used/60);
        for (counter=0;counter<in_usage_number_of_packages;counter++)
              UsageList[counter] = (XmString) XmStringCreateLtoR (
                    CharUsageList[counter], charset);
        UsageList[in_usage_number_of_packages] = NULL;

        ac = 0;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        XtSetArg(al[ac], XmNlistLabelString, XmStringCreateLtoR(
             "Package      licenses   concurrently   checked/day   used(in minutes)/day", charset));
        ac ++;
        XtSetArg(al[ac], XmNlistItems, UsageList); ac++;
        XtSetArg(al[ac], XmNlistItemCount, in_usage_number_of_packages); ac++;
        XtSetArg(al[ac], XmNvisibleItemCount,
                    (XtArgVal) in_usage_number_of_packages); ac++;
        XtSetArg(al[ac], XmNlistVisibleItemCount, 16); ac++;

        XtSetArg(al[ac], XmNselectionLabelString, NULL); ac++;
        Q_Usage_Result_dialog = XmCreateSelectionDialog(XtParent(w),
             title, al, ac);

        XtAddCallback(Q_Usage_Result_dialog, XmNokCallback, DestroyCB,
                                    DESTROY1);
        XtAddCallback(Q_Usage_Result_dialog, XmNhelpCallback, DialogAcceptCB,
                                    DIALOG_HELP);
        kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                    XmDIALOG_CANCEL_BUTTON);
        XtUnmanageChild(kid);
        kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                    XmDIALOG_SELECTION_LABEL);
        XtUnmanageChild(kid);
        kid = XmSelectionBoxGetChild(Q_Usage_Result_dialog,
                                    XmDIALOG_TEXT);
        XtUnmanageChild(kid);
        XtManageChild(Q_Usage_Result_dialog);
        break;
        }
```

```
      case RESTART:
        call_rpc(server,library_function,xdr_void,NULL,
              xdr_int,&librarian_result);
        sprintf(msg,"Librarian Result is %d\n",librarian_result);
        ac = 0;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
        ac++;       /*XtParent(XtParent(w)),*/
        msgD = XmCreateMessageDialog(w,
             "    Restart    Result    ", al, ac);
        XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
        XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
        kid = XmSelectionBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
        XtUnmanageChild(kid);
        XtManageChild (msgD);
        break;

      case SHUTDOWN:
        call_rpc(server,library_function,xdr_void,NULL,xdr_void,NULL);
        sprintf(msg,"OK. The server program has been shutdown.\n");
        ac = 0;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
        ac++;       /*XtParent(XtParent(w)),*/
        msgD = XmCreateMessageDialog(w,
             "    Shutdown    Result    ", al, ac);
        XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY2);
        XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
        kid = XmSelectionBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
        XtUnmanageChild(kid);
        XtManageChild (msgD);
        break;

      }
}
```

```
/*-------------------------------------------------------------
**      DialogCancelCB
**              Process callback from Dialog cancel actions.
*/
static void DialogCancelCB (w, client_data, call_data)
Widget          w;              /*  widget id           */
caddr_t         client_data;    /*  data from application   */
caddr_t         call_data;      /*  data from widget class  */
{
        switch ((int)client_data)
        {
          case DIALOG_CANCEL:
                XtUnmanageChild(w);
          case DIALOG_HELP:
                /* no action is necessage */
                break;
          default:
                /* a unknown client_data was recieved and
                   there is no setup to handle this */
                fprintf (stderr, "Warning: in cancel callback\n");
                break;
        }
}
```

```
/*-------------------------------------------------------------
**      NoMatchCB
**              Process callback from dialog when no match.
*/
void NoMatchCB (w, client_data, call_data)
Widget          w;
caddr_t         client_data;
caddr_t         call_data;
{
        Widget          msgD, kid;
        char            msg[60];
        register int    ac;
        Arg             al[10];

          sprintf(msg,"Check your typing or\nSelect one item\nfrom the list\0");

          ac = 0;
          XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
          ac++;
          msgD = XmCreateErrorDialog(XtParent(w),
                                     Warning       ", al, ac);
          kid = XmMessageBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
          XtUnmanageChild(kid);
          XtManageChild (msgD);
}
```

```
static void GetCopies (w, client_data, call_data)
Widget          w;              /*  widget id           */
caddr_t         client_data;    /*  data from application   */
caddr_t         call_data;      /*  data from widget class  */
{
     XmSelectionBoxCallbackStruct  *cb;
     register int ac;           /* arg count                */
     Arg al[10];                /* arg list                 */
     Widget     Copies_select;
     int i;

          cb = (XmSelectionBoxCallbackStruct *)call_data;
          i = 0;
          while (! (XmStringCompare(cb->value, PackagesList[i])) )
               i++;
          strcpy(s_m_package, CharPackagesName[i]);

          ac = 0;
          XtSetArg(al[ac], XmNmessageString, PackagesList[i]);
          ac++;
          XtSetArg(al[ac], XmNselectionLabelString,
                    XmStringCreateLtoR("Set up New Total Copies:", charset) );
          ac++;
          XtSetArg(al[ac], XmNokLabelString,
                         XmStringCreateLtoR("Confirm", charset) );
          ac++;
          XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
          Copies_select = XmCreatePromptDialog(w, "   Total   Copies?      ",
                                               al, ac);
          XtAddCallback(Copies_select, XmNokCallback, CallRPC, SET_MAX);
          XtAddCallback(Copies_select, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
          XtAddCallback(Copies_select, XmNcancelCallback, DestroyCB, DESTROY1);
          XtManageChild(Copies_select);

}
```

```
/*-------------------------------------------------------------
**      MenuCB
**              Process callback from PushButtons in PulldownMenus.
*/
void MenuCB (w, client_data, call_data)
Widget          w;              /*  widget id           */
caddr_t         client_data;    /*  data from application   */
caddr_t         call_data;      /*  data from widget class  */
{
        register int ac;                /* arg count                */
        Arg al[10];                     /* arg list                 */
        char msg[80];
        Widget          msgD;
        Widget          Q_Node_select;
        Widget          R_Node_select;
        Widget          Q_Package_select;
        Widget          C_O_Package_select;
        Widget          S_M_Package_select;
        Widget          Date_select;
        Widget          Restart_select;
        Widget          Shutdown_select;
        Widget          kid;

        switch ((int)client_data)
        {
                case QUIT:
          {
        exit(0);
            break;
          }
                case QNODE:
          {
        no_nodes = ReadLocalNodes();
        ac = 0;
        XtSetArg(al[ac], XmNlistLabelString, XmStringCreateLtoR(
                "Address        Name",charset));
        ac ++;
        XtSetArg(al[ac], XmNokLabelString, XmStringCreateLtoR("Apply",charset));
        ac ++;
        XtSetArg(al[ac], XmNlistItems, NodesList); ac++;
        XtSetArg(al[ac], XmNlistItemCount, no_nodes); ac++;
        XtSetArg(al[ac], XmNvisibleItemCount, (XtArgVal) no_nodes); ac++;
        XtSetArg(al[ac], XmNlistVisibleItemCount, 16); ac++;
        XtSetArg(al[ac], XmNmustMatch, True); ac++;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        Q_Node_select = XmCreateSelectionDialog(XtParent(w),
                "      Node    Selection    Box      ", al, ac);

        XtAddCallback(Q_Node_select, XmNokCallback, CallRPC, QNODE);
        XtAddCallback(Q_Node_select, XmNcancelCallback, DestroyCB, DESTROY1);
        XtAddCallback(Q_Node_select, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
        XtAddCallback(Q_Node_select, XmNnoMatchCallback, NoMatchCB, NOMATCH);

        XtManageChild(Q_Node_select);
            break;
          }

                case QPACKAGE:
          {
        no_packages = GetPackages();
        ac = 0;
        XtSetArg(al[ac], XmNlistLabelString, XmStringCreateLtoR(
                        "Packages",charset));
        ac ++;
        XtSetArg(al[ac], XmNokLabelString, XmStringCreateLtoR("Apply",charset));
```

```
ac ++;
XtSetArg(al[ac], XmNlistItems, PackagesName); ac++;
XtSetArg(al[ac], XmNlistItemCount, no_packages); ac++;
XtSetArg(al[ac], XmNvisibleItemCount, (XtArgVal) no_packages); ac++;
XtSetArg(al[ac], XmNlistVisibleItemCount, 16); ac++;
XtSetArg(al[ac], XmNmustMatch, True); ac++;
XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
Q_Package_select = XmCreateSelectionDialog(XtParent(w),
        "  Package  Selection  Box", al, ac);
XtAddCallback(Q_Package_select, XmNokCallback, CallRPC, QPACKAGE);
XtAddCallback(Q_Package_select, XmNcancelCallback, DestroyCB, DESTROY1);
XtAddCallback(Q_Package_select, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
XtAddCallback(Q_Package_select, XmNnoMatchCallback, NoMatchCB, NOMATCH);

XtManageChild(Q_Package_select);
    break;
  }

        case CHECK_OUT:
  {
no_packages = GetPackages();
ac = 0;
XtSetArg(al[ac], XmNlistLabelString, XmStringCreateLtoR(
                "Packages",charset));
ac ++;
XtSetArg(al[ac], XmNokLabelString, XmStringCreateLtoR("Apply",charset));
ac ++;
XtSetArg(al[ac], XmNlistItems, PackagesName); ac++;
XtSetArg(al[ac], XmNlistItemCount, no_packages); ac++;
XtSetArg(al[ac], XmNvisibleItemCount, (XtArgVal) no_packages); ac++;
XtSetArg(al[ac], XmNlistVisibleItemCount, 16); ac++;
XtSetArg(al[ac], XmNmustMatch, True); ac++;
XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
C_O_Package_select = XmCreateSelectionDialog(XtParent(w),
        "  Package  Selection  Box  ", al, ac);
XtAddCallback(C_O_Package_select, XmNokCallback, CallRPC, CHECK_OUT);
XtAddCallback(C_O_Package_select, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
XtAddCallback(C_O_Package_select, XmNcancelCallback, DestroyCB,
                DESTROY1);
XtAddCallback(C_O_Package_select, XmNnoMatchCallback,NoMatchCB,NOMATCH);

XtManageChild(C_O_Package_select);
    break;
  }

        case REMOVE:
  {
no_nodes = ReadLocalNodes();
ac = 0;
XtSetArg(al[ac], XmNlistLabelString, XmStringCreateLtoR(
        "Address        Name",charset));
ac ++;
XtSetArg(al[ac], XmNlistItems, NodesList); ac++;
XtSetArg(al[ac], XmNlistItemCount, no_nodes); ac++;
XtSetArg(al[ac], XmNvisibleItemCount, (XtArgVal) no_nodes); ac++;
XtSetArg(al[ac], XmNlistVisibleItemCount, 16); ac++;
XtSetArg(al[ac], XmNokLabelString,
                        XmStringCreateLtoR("Confirm", charset)); ac++;
XtSetArg(al[ac], XmNmustMatch, True); ac++;
XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
R_Node_select = XmCreateSelectionDialog(XtParent(w),
        "  Node  Selection  Box  ", al, ac);
XtAddCallback(R_Node_select, XmNokCallback, CallRPC, REMOVE);
XtAddCallback(R_Node_select, XmNcancelCallback, DestroyCB, DESTROY1);
XtAddCallback(R_Node_select, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
XtAddCallback(R_Node_select, XmNnoMatchCallback, NoMatchCB, NOMATCH);
```

```
XtManageChild(R_Node_select);
    break;
  }

        case SET_MAX:
  {
no_packages = GetPackages();
ac = 0;
XtSetArg(al[ac], XmNlistLabelString, XmStringCreateLtoR(
                "Packages     Total",charset));
ac ++;
XtSetArg(al[ac], XmNlistItems, PackagesList); ac++;
XtSetArg(al[ac], XmNlistItemCount, no_packages); ac++;
XtSetArg(al[ac], XmNvisibleItemCount, (XtArgVal) no_packages); ac++;
XtSetArg(al[ac], XmNlistVisibleItemCount, 16); ac++;
XtSetArg(al[ac], XmNmustMatch, True); ac++;
XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
S_M_Package_select = XmCreateSelectionDialog(XtParent(w),
        "  Package  Selection  Box  ", al, ac);
XtAddCallback(S_M_Package_select, XmNokCallback, GetCopies, REMOVE);
XtAddCallback(S_M_Package_select, XmNcancelCallback, DestroyCB,
                DESTROY1);
XtAddCallback(S_M_Package_select, XmNhelpCallback, DialogAcceptCB,
                DIALOG_HELP);
XtAddCallback(S_M_Package_select, XmNnoMatchCallback,NoMatchCB,NOMATCH);

XtManageChild(S_M_Package_select);
    break;
  }

        case DAILY:
  {
ac = 0;
XtSetArg(al[ac], XmNselectionLabelString,
 XmStringCreateLtoR("Enter the date correctly (MM-DD-YY)", charset));
ac++;
XtSetArg(al[ac], XmNokLabelString,
   XmStringCreateLtoR("Confirm", charset) );
ac++;
XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
Date_select = XmCreatePromptDialog(w,
        "   Specify Which Day?        ", al, ac);
XtAddCallback(Date_select, XmNokCallback, CallRPC, DAILY);
XtAddCallback(Date_select, XmNcancelCallback, DestroyCB, DESTROY1);
XtAddCallback(Date_select, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
XtManageChild(Date_select);
    break;
  }

        case WEEKLY:
  {
ac = 0;
XtSetArg(al[ac], XmNselectionLabelString,
 XmStringCreateLtoR("Enter that MONDAY correctly (MM-DD-YY)", charset));
ac++;
XtSetArg(al[ac], XmNokLabelString,
   XmStringCreateLtoR("Confirm", charset) );
ac++;
XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
Date_select = XmCreatePromptDialog(w,
        "   Specify Which Week?        ", al, ac);
XtAddCallback(Date_select, XmNokCallback, CallRPC, WEEKLY);
XtAddCallback(Date_select, XmNcancelCallback, DestroyCB, DESTROY1);
XtAddCallback(Date_select, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
XtManageChild(Date_select);
```

```
            break;
          }
                case MONTHLY:
          {
        ac = 0;
        XtSetArg(al[ac], XmNselectionLabelString,
         XmStringCreateLtoR("Enter the month correctly (MM-YY)", charset));
        ac++;
        XtSetArg(al[ac], XmNokLabelString,
           XmStringCreateLtoR("Confirm", charset) );
        ac++;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        Date_select = XmCreatePromptDialog(w,
                "   Specify Which Month?        ", al, ac);
        XtAddCallback(Date_select, XmNokCallback, CallRPC, MONTHLY);
        XtAddCallback(Date_select, XmNcancelCallback, DestroyCB, DESTROY1);
        XtAddCallback(Date_select, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
        XtManageChild(Date_select);
            break;
          }
                case SEMESTER:
          {
        ac = 0;
        XtSetArg(al[ac], XmNselectionLabelString,
        XmStringCreateLtoR("Which semester?\nEnter 1 for Spring, 2 for Summer, 3 for Fall", charset));
        ac++;
        XtSetArg(al[ac], XmNokLabelString,
           XmStringCreateLtoR("Confirm", charset) );
        ac++;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        Date_select = XmCreatePromptDialog(w,
                "   Specify Which Semester?        ", al, ac);
        XtAddCallback(Date_select, XmNokCallback, CallRPC, SEMESTER);
        XtAddCallback(Date_select, XmNcancelCallback, DestroyCB, DESTROY1);
        XtAddCallback(Date_select, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
        XtManageChild(Date_select);
            break;
          }

                case RESTART:
          {
        strcpy(msg, "Are you sure you want to\nrestart the server program?");
        ac = 0;
        XtSetArg(al[ac],XmNmessageString,XmStringCreateLtoR(msg, charset));
        ac ++;
        XtSetArg(al[ac], XmNokLabelString,
                                XmStringCreateLtoR("Confirm",charset));
        ac ++;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        Restart_select = XmCreateQuestionDialog(XtParent(w),
                                "      Attention      ", al, ac);
        XtAddCallback(Restart_select, XmNokCallback, CallRPC, RESTART);
        XtAddCallback(Restart_select, XmNcancelCallback, DestroyCB, DESTROY1);
        XtAddCallback(Restart_select, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
        XtManageChild(Restart_select);
            break;
          }

                case SHUTDOWN:
          {
        strcpy(msg, "Are you sure you want to\nterminate the server program?\nif so,\nyou have to restart the server\nmanually!");
        ac = 0;
        XtSetArg(al[ac],XmNmessageString,XmStringCreateLtoR(msg, charset));
        ac ++;
        XtSetArg(al[ac], XmNokLabelString,
                                XmStringCreateLtoR("Confirm",charset));
```

```
        ac ++;
        XtSetArg(al[ac], XmNautoUnmanage, FALSE); ac++;
        Shutdown_select = XmCreateQuestionDialog(XtParent(w),
                                "      Attention      ", al, ac);
        XtAddCallback(Shutdown_select, XmNokCallback, CallRPC, SHUTDOWN);
        XtAddCallback(Shutdown_select, XmNcancelCallback, DestroyCB, DESTROY1);
        XtAddCallback(Shutdown_select, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
        XtManageChild(Shutdown_select);
            break;
          }

                case HELP:
          {
            sprintf(msg,"This is a X Window interface for Libmaint\n\0");
            ac = 0;
            XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
            ac++;
            msgD = XmCreateMessageDialog(XtParent(XtParent(w)),
                "   Help  Box          ", al, ac);
            XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
            XtAddCallback(msgD, XmNcancelCallback, DestroyCB, DESTROY1);
            XtAddCallback(msgD, XmNhelpCallback, DialogAcceptCB, DIALOG_HELP);
            XtManageChild (msgD);
            break;
          }

                default:
                        /* unknown client_data was recieved and
                           there is no setup to handle this */
                        fprintf(stderr, "Warning: in menu callback\n");
                        break;
        }

}
```

```
/*-------------------------------------------------------------
**      DialogAcceptCB
**              Process callback from Dialog actions.
*/
static void DialogAcceptCB (w, client_data, call_data)
Widget          w;              /*  widget id           */
caddr_t         client_data;    /*  data from application   */
caddr_t         call_data;      /*  data from widget class  */
{
        register int ac;                /* arg count                */
        Arg al[10];                     /* arg list                 */
        char msg[80];
        Widget msgD, kid;

        switch ((int)client_data)
        {
          case DIALOG_OK:
                {
                        XtUnmanageChild(w);
                }
                break;

          case DIALOG_HELP:
          {
            sprintf(msg,"Sorry! Help is not available at this time\n\0");
            ac = 0;
            XtSetArg(al[ac],XmNmessageString, XmStringCreateLtoR(msg, charset));
            ac++;
            msgD = XmCreateMessageDialog(XtParent(XtParent(w)),
                "   Help  Box          ", al, ac);
            XtAddCallback(msgD, XmNokCallback, DestroyCB, DESTROY1);
            XtAddCallback(msgD, XmNhelpCallback, DestroyCB, DESTROY1);
            kid = XmSelectionBoxGetChild(msgD, XmDIALOG_CANCEL_BUTTON);
            XtUnmanageChild(kid);
            XtManageChild (msgD);
            break;
          }

        default:
                /* unknown callback type */
                fprintf (stderr, "Warning: in accept callback\n");
                break;
        }
}
```

```
/*-------------------------------------------------------------
**      CreateMenuBar
**              Create MenuBar in MainWindow
*/
static Widget CreateMenuBar (parent)
Widget          parent;
{
        Widget          menu_bar;       /*  RowColumn                   */
        Widget          menu_pane;      /*  RowColumn                   */
        Widget          cascade;        /*  CascadeButton               */
        Widget          button;         /*  PushButton                  */

        Arg             al[10];         /*  arg list                    */
        register int    ac;             /*  arg count                   */

        /*      Create MenuArea.
        */
        ac = 0;
        menu_bar = XmCreateMenuBar (parent, "menu_bar", al, ac);

        /*      Create "Options" PulldownMenu.
        */
        ac = 0;
        menu_pane = XmCreatePulldownMenu (menu_bar, "menu_pane", al, ac);

        ac = 0;
        XtSetArg(al[ac], XmNlabelString,
                XmStringCreateLtoR("Node...", charset)); ac++;
        XtSetArg(al[ac], XmNmnemonic, 'N'); ac++;
        button = XmCreatePushButton (menu_pane, "QNODE", al, ac);
        XtAddCallback (button, XmNactivateCallback, MenuCB, QNODE);
        XtManageChild (button);

        ac = 0;
        XtSetArg(al[ac], XmNlabelString,
                XmStringCreateLtoR("Package...", charset)); ac++;
        XtSetArg(al[ac], XmNmnemonic, 'P'); ac++;
        button = XmCreatePushButton (menu_pane, "QPACKAGE", al, ac);
        XtAddCallback (button, XmNactivateCallback, MenuCB, QPACKAGE);
        XtManageChild (button);

        ac = 0;
        XtSetArg(al[ac], XmNlabelString,
                XmStringCreateLtoR("Time", charset)); ac++;
        XtSetArg(al[ac], XmNmnemonic, 'T'); ac++;
        button = XmCreatePushButton (menu_pane, "QTIME", al, ac);
        XtAddCallback (button, XmNactivateCallback, CallRPC, QTIME);
        XtManageChild (button);

        ac = 0;
        XtSetArg(al[ac], XmNlabelString,
                XmStringCreateLtoR("Users", charset)); ac++;
        XtSetArg(al[ac], XmNmnemonic, 'U'); ac++;
        button = XmCreatePushButton (menu_pane, "QUSERS", al, ac);
        XtAddCallback (button, XmNactivateCallback, CallRPC, QUSERS);
        XtManageChild (button);

        ac = 0;
        XtSetArg(al[ac], XmNlabelString,
                XmStringCreateLtoR("NodeS", charset)); ac++;
        XtSetArg(al[ac], XmNmnemonic, 'S'); ac++;
        button = XmCreatePushButton (menu_pane, "QNODES", al, ac);
        XtAddCallback (button, XmNactivateCallback, CallRPC, QNODES);
        XtManageChild (button);
```

```
ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("License", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'L'); ac++;
button = XmCreatePushButton (menu_pane, "QLICENSE", al, ac);
XtAddCallback (button, XmNactivateCallback, CallRPC, QLICENSE);
XtManageChild (button);

ac = 0;
XtSetArg (al[ac], XmNsubMenuId, menu_pane);  ac++;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Query     ", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'Q'); ac++;
cascade = XmCreateCascadeButton (menu_bar, "Query", al, ac);
XtManageChild (cascade);

/*      Create "Options" Pulldown Menu.
*/
ac = 0;
menu_pane = XmCreatePulldownMenu (menu_bar, "menu_pane", al, ac);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Check In", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'I'); ac++;
button = XmCreatePushButton (menu_pane, "Check In", al, ac);
XtAddCallback (button, XmNactivateCallback, CallRPC, CHECK_IN);
XtManageChild (button);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Check Out...", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'O'); ac++;
button = XmCreatePushButton (menu_pane, "Check Out", al, ac);
XtAddCallback (button, XmNactivateCallback, MenuCB, CHECK_OUT);
XtManageChild (button);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Remove Node...", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'R'); ac++;
button = XmCreatePushButton (menu_pane, "Remove Node", al, ac);
XtAddCallback (button, XmNactivateCallback, MenuCB, REMOVE);
XtManageChild (button);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Set Max...", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'S'); ac++;
button = XmCreatePushButton (menu_pane, "Set Max", al, ac);
XtAddCallback (button, XmNactivateCallback, MenuCB, SET_MAX);
XtManageChild (button);

ac = 0;
XtSetArg (al[ac], XmNsubMenuId, menu_pane);  ac++;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Maintain      ", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'M'); ac++;
cascade= XmCreateCascadeButton (menu_bar, "Maintain", al, ac);
XtManageChild (cascade);

/*      Create "Options" Pulldown Menu.
*/
ac = 0;
menu_pane = XmCreatePulldownMenu (menu_bar, "menu_pane", al, ac);
```

```
ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Daily...", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'D'); ac++;
button = XmCreatePushButton (menu_pane, "Daily", al, ac);
XtAddCallback (button, XmNactivateCallback, MenuCB, DAILY);
XtManageChild (button);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Weekly...", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'W'); ac++;
button = XmCreatePushButton (menu_pane, "Weekly", al, ac);
XtAddCallback (button, XmNactivateCallback, MenuCB, WEEKLY);
XtManageChild (button);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Monthly...", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'M'); ac++;
button = XmCreatePushButton (menu_pane, "Monthly", al, ac);
XtAddCallback (button, XmNactivateCallback, MenuCB, MONTHLY);
XtManageChild (button);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Semester...", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'S'); ac++;
button = XmCreatePushButton (menu_pane, "Semester", al, ac);
XtAddCallback (button, XmNactivateCallback, MenuCB, SEMESTER);
XtManageChild (button);

ac = 0;
XtSetArg (al[ac], XmNsubMenuId, menu_pane);  ac++;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Report     ", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'R'); ac++;
cascade= XmCreateCascadeButton (menu_bar, "Report", al, ac);
XtManageChild (cascade);


/*      Create "Options" Pulldown Menu.
*/
ac = 0;
menu_pane = XmCreatePulldownMenu (menu_bar, "menu_pane", al, ac);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Restart", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'R'); ac++;
button = XmCreatePushButton (menu_pane, "Restart", al, ac);
XtAddCallback (button, XmNactivateCallback, MenuCB, RESTART);
XtManageChild (button);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Shutdown", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'S'); ac++;
button = XmCreatePushButton (menu_pane, "Shutdown", al, ac);
XtAddCallback (button, XmNactivateCallback, MenuCB, SHUTDOWN);
XtManageChild (button);

ac = 0;
XtSetArg(al[ac], XmNlabelString,
        XmStringCreateLtoR("Quit", charset)); ac++;
XtSetArg(al[ac], XmNmnemonic, 'Q'); ac++;
```

```
        button = XmCreatePushButton (menu_pane, "Quit", al, ac);
        XtAddCallback (button, XmNactivateCallback, MenuCB, QUIT);
        XtManageChild (button);

        ac = 0;
        XtSetArg (al[ac], XmNsubMenuId, menu_pane);  ac++;
        XtSetArg(al[ac], XmNlabelString,
                 XmStringCreateLtoR("Special ", charset)); ac++;
        XtSetArg(al[ac], XmNmnemonic, 'S'); ac++;
        cascade= XmCreateCascadeButton (menu_bar, "Special", al, ac);
        XtManageChild (cascade);


        /*      Create "Help" button.
        */

        ac = 0;
        cascade = XmCreateCascadeButton (menu_bar, "Help", al, ac);
        XtAddCallback (cascade, XmNactivateCallback, MenuCB, HELP);
        XtManageChild (cascade);

        ac = 0;
        XtSetArg (al[ac], XmNmenuHelpWidget, cascade);  ac++;
        XtSetValues (menu_bar, al, ac);

        return (menu_bar);
}
```

```
/*-------------------------------------------------------------
**      main
**              Initialize toolkit.
**              Create MainWindow and subwidgets.
**              Realize toplevel widgets.
**              Process events.
*/
void main (argc,argv)
unsigned int    argc;
char            **argv;
{
        Display         *display;       /*  Display             */
        Widget          app_shell;      /*  ApplicationShell    */
        Widget          mainwin;        /*  MainWindow          */
        Widget          menu_bar;       /*  RowColumn           */

        Arg             al[10];         /*  arg list            */
        register int    ac;             /*  arg count           */
        register int    i;              /*  counter             */


        /*      Initialize toolkit and open display.
        */

        XtToolkitInitialize ();
        display = XtOpenDisplay (NULL, NULL, argv[0], "XMdemos",
                        NULL, 0, &argc, argv);
        if (!display)
        {
                XtWarning ("license: can't open display, exiting...");
                exit (0);
        }


        /*      Create ApplicationShell.
        */
        app_shell = XtAppCreateShell (argv[0], "XMdemos",
                        applicationShellWidgetClass, display, NULL, 0);


        /*      Create MainWindow.
        */
        ac = 0;
        XtSetArg (al[ac], XmNshadowThickness, 0);  ac++;
        mainwin = XmCreateMainWindow (app_shell, "main", al, ac);
        XtManageChild (mainwin);


        /*      Create MenuBar in MainWindow.
        */
        menu_bar = CreateMenuBar (mainwin);
        XtManageChild (menu_bar);


        /*      Realize toplevel widgets.
        */
        XtRealizeWidget (app_shell);


        /*      Process events.
        */
        XtMainLoop ();
}
```

```
/*****************************************************************************/
/*                                                                           */
/* Procedure "call_rpc"                                                      */
/*                                                                           */
/* Given an RPC number, the addresses of the XDR input and output routines,  */
/* and the addresses of the XDR return parameters, this procedure calls the  */
/* standard RPC interface and returns the client status.  If an error        */
/* occurs, the RPC error is displayed and the program terminates.            */
/*                                                                           */
/*****************************************************************************/

call_rpc(server,library_function,inproc,in,outproc,out)
         char *server;
         int library_function;
         xdrproc_t inproc, outproc;
         char *in, *out;
{
   int client_status;
   client_status=callrpctcp(server,(u_long) SOFTLIB,
                            (u_long) library_function,
                            (u_long) SOFTVERS,
                            inproc , in,
                            outproc, out);
   if (client_status)
      {
         clnt_perrno(client_status);
         printf("\n");
         exit(-1);
      }
}


callrpctcp(host, prognum, procnum, versnum, inproc, in, outproc, out)
    long prognum, procnum, versnum;
    char *host, *in, *out;
    xdrproc_t inproc, outproc;
{
    struct sockaddr_in server_addr;
    int socket = RPC_ANYSOCK;
    enum clnt_stat clnt_stat;
    struct hostent *hp;
    register CLIENT *client;
    struct timeval total_timeout;

    if ((hp = gethostbyname(host)) == NULL) {
        fprintf(stderr, "can't get addr for '%s'\n", host);
        exit(-1);
    }
    bcopy(hp->h_addr, (caddr_t)&server_addr.sin_addr, hp->h_length);
    server_addr.sin_family = AF_INET;
    server_addr.sin_port = 0;
    if ((client = clnttcp_create(&server_addr, prognum, versnum,
      &socket, BUFSIZ, BUFSIZ)) == NULL) {
        perror("rpctcp_create");
        exit(-1);
    }
    total_timeout.tv_sec = 20;
    total_timeout.tv_usec = 0;
    clnt_stat = clnt_call(client, procnum,
        inproc, in, outproc, out, total_timeout);
    clnt_destroy(client);
    return (int)clnt_stat;
}
```