DESIGN OF OPERATING SYSTEMS
LABORATORY MANUAL

BY
MANAL ELSIR ELHADI MUSTAFA
B.Sc. Hon. (Computer Engineering, 1999)

A Thesis Submitted For Partial Fulfillment of Degree of Master
Of Computer Architecture and Networking

Department of Electrical and Electronics Engineering
Faculty of Engineering, University of Khartoum
Sudan

(July 2003)

Dedication

I dedicate this work to my loving family,
And to Mr. Elfatih S. Idris
And to all whom I love

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my thesis advisor, Dr ELRASHED OSMAN ELKHDER, for his support, guidance and advice during the course of this research.

I am indebted to DR SAMI SHAREIF for much support and encouragement over the years.

I am grateful to Ustaz MOHAMMED ELKARORRY & Ustaz ABD ALLAH TAG ELDEEN for the inspiring work, which provided the starting point for this work.

I am deeply grateful to my parents, HIKMAT MOHAMMED ABD ALAZEEZ and ELSIR ELHADI MUSTAFA, for their love and support in good times and bad. And to Mr. ELFATIH S. IDRIS, for his advice and for all the time we share.

صلختسم

رتويبمكلا ةسدنه ةبعشب سويرولاكبلا ةجرد ..... رجح ةباث .ررقم ليغشتلا مظن ةسدنه ررقم .ة.خ.ا ةيسارديلا لوصفلا ..... ىلع ... تاراهم ةدع ررق.ا بلطتي .

ةمدختس.ا بيلاس.ل ةمدقم ةنم دوصقم ليغشتلا مظن لمعم باتكل ميمصتلا اذه implement ليغشتلا مظنة..لا مظنل ةعباتلا عاون.او .

و رثكا ليصافتب انداد .ليغشتلا مظن ةسدنه ررق .ةحلم اهنم دصقلا براجتلا نم ةعوم .ىلع ىوت . باتكلا hands-onليغشتلا مظن ميمصتو ليل . .ة.خ .

Abstract

Engineering Operating Systems course is the capstone course for student Bachelor degree within the Department of Computer Engineering. The course requires the use of many skills that student have developed and refined over the last several semesters.

This design of operating systems laboratory manual is intended as a general introduction to the techniques used to implement operating systems and related kinds of systems software.

The manual contains set of experiment aims to supplement the operating systems classroom instruction by providing a more detailed and hands-on experience on the analysis and design of operating systems.

TABLE OF CONTENT

CHAPTER ONE
INTRODUCTION

CHAPTER TWO
OPERATING SYSTEM STRUCTURE

CHAPTER THREE
MANUAL DESIGN

CHAPTER FOURE
LABORATORY EXPERIMENTS

CHAPTER FIVE
CONCLUTION

CHAPTER ONE
INTRODUCTION

An operating system is a program, which acts as interface between a user of a computer and the computer hardware. It has tow main goals: -

– It provides a convenient environment for the user to execute a program.

– To use the computer hardware in efficient way.

A computer system can be divided into four components refer to figure Figure 1-1: -

Hardware, operating system , application programs and user.

Figure 1- 1 computer system components

The hardware (CPU, Memory, I/O) provides the basic compiling resource. The application programs define the utilization of these resources to solve the computing problems of the user.

The operating system controls and coordinates the use of the hardware among the various application programs for the users. It is a resource allocator. A computer system has many resources, which may be required to solve a problem: -

1. CPU time 2. Memory space 3. File storage 4. I/O devices.

The operating system acts as a manager of these resources and allocates them to specific programs of users as necessary for their tasks.

The operating system is a control program controlling the execution of user programs to prevent errors and improper use of computer, especially I/O devices. It easier to define operating system by what they do, rather than what they are. [1]

1.1 Objective and function of the operating system:

The operating system must instruct the keyboard to read the statement of the program. If the program or job is written in FORTRAN (e.g.), then the computer must be instructed by an operating system program to load and execute the FORTRAN compiler (a different compiler is loaded for a different language). After the high level FORTRAN instructions have been converted to machine instructions, the computer must be directed by operating system program to load these machine instructions into memory and begin execution of these instructions.

When the program is to print answers or results on the printer an operating system program must direct this output to the proper printer.

When this job is terminated the computer must be directed by an operating system program to continue execution of other jobs. The collection of routines that read data from keyboard, load the compiler, load and initiates

execution of the program, controls the output and monitors the job completion is called the operating system .see figure 1-2.

2

An operating system can be very simple for micros and mini-system to very sophisticated or complex for large general-purpose systems. The size and complexity of the operating system is a function of the type of functions the computer system to perform. [1]

Figure: - (1-2) multi step processing of user program

3

1.2 A Brief History of the UNIX operating system

The first version of UNIX was created in 1969 by Kenneth Thompson and Dennis Ritchie, system engineers at AT&T's Bell Labs. It went through many revisions and gained in popularity until 1977, when it was first made commercially available by Interactive Systems Corporation.

At the same time a students from the University of California at Berkeley was working to improve UNIX. In 1977 it released the first Berkeley Software Distribution, which became known as BSD.

By 1983 commercial interest was growing and Sun Microsystems produced a UNIX workstation. System V appeared directly descended from the original AT&T UNIX and the prototype of the more widely used variant today.[2]

Linux:

Back in August of 1991, a student from Finland began a post to the comp.os.minix newsgroup with the words:

> Hello everybody out there using minix -
>
> I'm doing a (free) operating system (just a hobby, won't be big and professional like gnu) for 386(486) AT clones.

The student was Linus Torvalds, and the "hobby" he spoke of eventually became what peoples know today as Linux.

A full-featured POSIX-like operating system, Linux has been developed not just by Linus, but by hundreds of programmers around the world.

4

Red Hat Linux:

Go in a group of programmers based in North Carolina. Their goal was to make it easier for people to give Linux a try. Like many other such groups, their approach was to bundle all the necessary bits and

pieces into a cohesive distribution, relieving "newbies" from some of the more esoteric aspects of bootstrapping a new operating system on their PCs.

Since Red Hat Linux's introduction in the summer of 1994, Linux and Red Hat have grown by leaps and bounds. Much has changed; support for more esoteric hardware, huge increases in reliability, and the growing use of Linux by companies around the world.

. Linux is still developed by people worldwide; Linus is still involved. Red Hat is still headquartered in North Carolina still trying to make Linux easier for people to use.

On most systems, Red Hat Linux is easy to install; the installation program can walk user through the process in as little as 15 minutes. The system itself is very flexible. With RPM, user can install and uninstall individual software packages with minimal effort. [3]

1.3 A Brief History of the Windows NT Operating System

With the fifth major version of the Microsoft Windows NT operating system, to be called Windows 2000, Windows NT technology from Microsoft Corporation. will go mainstream to millions of users, and Microsoft will cease distributing the technology under the separate

Windows NT product name. Here's a brief look back at the history of this phenomenally successful product:

• **1988:** Microsoft formed what would become the development team for Windows NT, with the goal of developing a thoroughly modern, fully 32-bit, robust, multipurpose operating system.

• **August 1991:** This version, Windows NT, offered the advanced operating system features needed for mission-critical applications, high-performance servers, advanced workstations and client/server computing.

• **July 1993:** Microsoft released Windows NT 3.1 and Windows NT Advanced Server 3.1. They broke new ground in operating system power, performance and reliability with a range of features: micro-kernel architecture, pre-emptive multitasking scheduler, x86/MIPS support (and, shortly after introduction, Alpha processor support), **September 1994:** Windows NT Workstation 3.5 and Windows NT Server 3.5, code-named "Daytona," were introduced. These versions built on the ruggedness and stability of version 3.1 to greatly enhance speed, reduce size and provide greater connectivity to other systems, particularly Novell NetWare and UNIX environments. Both versions included accessibility features for users with limited dexterity or hearing impairments.

• **June 1995:** Less than a year later, Microsoft announced Windows NT 3.51. Windows NT Server now included a tool to help customers manage Client Access Licenses for Microsoft BackOffice family products and a utility that enabled over-the-network installation of Windows 95. Windows NT Workstation provided support for Windows 95-compatible applications, popular fax software, a replaceable WinLogon screen.
6
• **July 1996:** Microsoft released Windows NT Workstation 4.0 and Windows NT Server 4.0. Windows NT Server 4.0 brought ease of use and management, higher network throughput, and a complete set of tools for developing and managing Intranets. Windows NT Server 4.0 included Microsoft Internet Information Server version 2.0 and the Microsoft FrontPage Web site creation and management tool version 1.1. Also included the popular Windows 95 user interface and built-in networking support, providing secure, easy access to the Internet and corporate Intranets.
• **Oct. 21, 1998:** Microsoft releases Service Pack 4.0 for Windows NT Workstation 4.0 and Windows NT Server 4.0. Since the original introduction of Windows NT 4.0, the product has evolved through four service packs and one option pack, adding and integrating public-key and certificate authority functionality, smart card support, improved Symmetric multiprocessing (SMP) scalability, clustering, reliable synchronous and queued transaction support, streaming media features, and many Web-relevant browser and server technologies.
• **Oct. 27, 1998:** Microsoft announces that Windows NT will be known as "Windows" with the next major version being called "Windows 2000." The move reflects the marketplace momentum driving Windows NT-based technology to become the mainstream Windows-based technology for millions of customers worldwide. The new naming system is also designed to make it easier for consumers and businesses to choose the right Windows operating system products for their needs.

Microsoft, Windows NT, Windows, Win32, BackOffice and FrontPage are either registered trademarks or trademarks of Microsoft Corporation. in the United States and/or other countries.
7
Other product and company names herein may be trademarks of their respective owners. [4]

1.4 What is the difference between Linux and Windows?

There are a lot of differences between Linux and Windows. The first noticed right off is Linux is highly configurable. Unlike Windows, Linux

comes with the complete source code for the kernel (kernel32.exe under windows). The kernel is the main operating component of an operating system. It handles input/output, running of programs, and memory. Since the source code is released with Linux, user can configure the kernel to be very small and support only the devices that user has in its system. This makes for a very stable system. Something else that noticed about Linux that sets it apart from Windows is the fact that it is a stable operating system. It hardly (if ever) crashes or has some kind of weird errors that pop up for no reason. Windows is acceptable to many viruses, on the other hand, Linux is only acceptable to one virus. The author of this virus also freely released a cleaning program to clean the virus since it was for demonstration purposes only. What is this X Windows thing?

X Windows is a Graphic User Interface (GUI) for Unix systems. It provides a way to produce windows/graphics just like MS Windows or Mac OS. The main difference between MS Windows and X Windows is that when user start X Windows, it's just a blank screen until user configure a Window Manager (WM) for it.

1.5 Thesis objective and layout:

The aim of this thesis is to design and prepare a manual laboratory for the operating systems (Windows/ Linux) Laboratory of department of computer engineering, College of Technological Sciences, Omderman, Sudan. The manual should provide the practical curricula for the operating system courses taught by the department as part of the B.Sc. Hon. degree program

To realize this goal, the department course syllabus was reviewed to determine the topics to be considered when designing the laboratory experiments in order to be consistent with the course. Moreover, the Word Wide Web (WWW) was intensively used in order to obtain a broader view about the modern laboratory curricula adopted by the universities word wide.

Chapter 2 is a representation of the operating system course topics. Chapters 3 discuss the way of designing a manual including manual objectives and represent the topics of the laboratory course to achieve the goal of the course, The manual organization and usage at the end of the chapter.

Chapter 4 contains section about experiment design and another section contains detailed descriptions to each experiment and laboratory work. There are some points about Laboratory report writing guidelines.

The thesis conclusion and recommendations are the subject for chapter 5, finally part 1 and 2 of the manual are included in appendix A and B respectively.

CHAPTER 2
Operating System Structures

2.1 What is an Operating System (OS)?

A computer's software were divided into two categories:

User software: Applications such as email clients, word processors, games, etc.

System software: e.g., disk managers, network daemons, In this scheme of things, the operating system is the most fundamental piece of system software.

The OS is dedicated to make the computer efficient and usable.

2. 2 Types of Computer Systems

1. Batch System
2. Multiprogramming Systems
3. Multitasking Systems
4. Personal Computers
5. Multiprocessor Systems
6. Distributed Systems

• **Batch Systems (1950's)**

These had processor speeds measured in kHz.

Programs were stored on cards - a fast card reader could handle up to 20 cards per second.

Programs were submitted by hand to an operator who put complementary programs into separate “batches" and loaded them. The OS took the form of the resident monitor - consisting of:

- Card Interpreter - reading and carrying out instructions on cards.

- Loader - placing system programs and application programs into memory.
- Device Drivers - software to manage to Input and Output (I/O) devices.

The biggest problem with Batch Systems was slow performance (I/O) and CPU operations could not overlap; the CPU was much faster than the card reader but had to wait for instructions to be read in.

The solution is to over-lap job execution with I/O. The OS would simultaneously:

- Read the next jobs from card reader onto a disk
- Execute the current job

- Output the results of the previous job to the printer.

With the advancement of disk technology, job scheduling became possible. In particular, the concept of Multiprogramming was introduced.

• **Multiprogramming Systems (1960s)**

BASIC IDEA: Processors run faster than Input and Output devices.
BASIC GOAL: Keep the processor in use at all times.
Several jobs are selected from a job pool and are loaded into memory.
The OS selects one and starts to execute it.
If that job is delayed (waiting for I/O) the CPU selects another and runs that instead. When that one is delayed, a 3rd is selected and that is executed.
Eventually, by the time the OS returns to the first job it should have completed the I/O by them.

• **Multitasking (1970s):**

BASIC IDEA: for interactive computing the CPU must be able to switch between jobs quickly.
BASIC GOAL: to give the impression to users/programs that they have exclusive use of the computer.

The CPU gives a small amount of attention to each available job. This is called:
Time-Sharing. This gives the impression that all the jobs are running at the same time.
Instead of submitting programs on cards to the operator, several users (2, 10, 100...) interact with the computer at the same time, using key-board/mouse/whatever.
Typing is slow: about 1 key-press every 0.1s. Between key-presses, the CPU has time to perform other tasks such as reading another user's keystrokes.

• **Personal Computers (1980's)**

Also known as microcomputers, PCs were cheap compared to mainframes/minicomputers. One could afford to have a ratio of One user to one Computer; the operating systems were designed primarily to be easy to use.
Early PC makers included Apple and IBM.
Because only one user was using a computer at any given time, the OS could be much simpler than on the big systems:

-No multiprogramming
-No security
- No file/memory protection.

This was to change because

-Small computers became faster - comparable to speed on mini computers.

- Use of networking requires security.
- Multitasking Operating Systems were ported to microcomputers.

• **Multiprocessor Systems (1980s):**

BASIC IDEA: to increase a computer's speed, increase the number of CPUs that it has access to.

12

As limits of processor speeds are reached, high performance computing (where emphasis is on performing as many calculations as possible in a short amount of time) looked to increasing the number of CPUs in a computer.

There are two basic types of multiprocessing:

Symmetric multiprocessing (SMP): common on smaller systems, must operating systems have a facility for SMP. All processors are equal (peers) and run a copy of the OS.

Asymmetric multiprocessing: master-slave model.

Real-time Systems:

Are employed when there are very rigid time constraints - when the system must complete its designated task in a given time frame. Examples include:

-Robotics Control
- Medical Equipment
- Domestic Appliances
- Fuel injection systems

There are two basic types of real-time system:

1. Hard real-time: no secondary storage (e.g., hard-drive), no time-sharing or multiprogramming, data is usually stored in ROM.

2. Soft real-time: Used in real-time systems that require some advanced OS features, e.g., multimedia systems, space exploration, etc.

When a time-critical task must be completed, it gets priority over other operations until it completes.

• **Distributed Systems:**

Distributed Information Systems are a familiar concept: the World Wide Web is a good example. Instead of all the information being stored on one computer, small amounts are on different computers connected by a network. [5][6]

13

2. 3 Operating system services:

The operating system provides certain services to programs and to users of those programs. These services are provided for the convenience of the programmer, to make the programming task easier.

Some classes of services are:

1. Program creation

The operating system provides a variety of facilities and services to assist the programmer in creating programs. These are lumped under the generic name utilities

2. Program Execution

A number of tasks need to be performed to execute a program. Instructions and data must be loaded into main memory, I/O devices and files must be initialized, and other resources must be prepared. The operating system handles all this for the user.

The program must be able to end its execution, either normally or abnormally (indicating error)

3. Access to I/O device requires its own peculiar set of instructions or control signals for operation, the operating system takes care of the details so that the programmer can think in terms of simple reads and writes.

A running program may require input and output this I/O may involve a file or an I/O device, the operating system provides some means to do so.

4. file system manipulation and control

Programs need to read and write files. Peoples want to create and delete files by name.

In the case of files, control must include an understanding of not only the nature of the I/O device (disk drive, tape drive) but also the file format on the storage medium, again, the operating system worries a bout the details.

5. Communications

There are many circumstances in which one process needs to exchange information with another process. There are two major ways in which such communication can occur. The first takes place between two process executing on the same computer; the second takes place between processes executing on different computer systems that are tied together by a computer network. Communications may be implemented via shared memory or by message passing in which packets of information are moved between processes by the operating system.

6. Error detection

The operating system constantly needs to be aware of possible error. Error may occur in the CPU and memory hardware (such as a memory error or a power failure), in I/O devices (such as a parity error on tape, a connection failure on a network or lack of paper in the printer), or in the user program (such as an arithmetic overflow, an attempt to access an illegal memory location, or a too great use of CPU time. For each type of error, the operating system should take appropriate action to ensure correct and consistent computing.

In addition, there are a set of operating system functions which exist not for the user but for the efficient operation of the system itself those systems with multiple users can gain efficiency for sharing the computer resources among users.

7. Resource allocation

When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. Many different types of resources are managed by the operating system e.g. resources – CPU cycles, main memory, file storage, I/O devices others are plotters, modems and other peripheral devices.

8. Accounting

Keep track of which users use how much and what kind of computer resources i.e. users can be billed – also usage statistics.

9. Protection and security

The owners of information stored in multi-user computer system may want to control its use. Security of the system from outsides is also important. Use passwords to be allowed access to the resources. Security extends to defending external adapters, from invalid access attempts.

How the operating system services are provided?

They are provided by two methods: system calls and system programs.

1. System Calls :

System calls provide the interface between a process and operating system.

C – language allow system calls to be made directly (also Pascal and Fortran)

How system calls are used?

As an example consider writing a simple program to read data from one file and to copy them to another file.

The first input the program will need is the names of the two files: the input file and the out put file.

On. Mouse – based and icon- based systems, a menu of file names is usually displayed in a window. The user can then use the mouse to select the same name, and a window can be opened for destination name to be specified.

Once the two file names are obtained, the program must open the input file and create the output file (system call).

If the file does not exist an error message is given (system call) and the program terminates abnormally (system call).

Now both files are set up, then enter a loop that read from input file (a system call) and writes to the output file (another system call).
After the entire file is copied, the program may close both files (another system call) and finally terminates normally (the last system call).
Most user never see this level of detail, however e.g. a write statement in Pascal or Fortran, is compiled into a support routine that issues the necessary system calls, checks for errors and finally return to the user program thus , most of the details of the operating system interface are hidden from the programmer by the compiler and by the support routine.
There are five types of system calls: -
1- process and job control

A running program needs to be able to halt its execution either normally (end) or an abnormally (abort).
A process or job executing one program may want to load and execute another program.
2- File manipulation

This means how to be able to create and delete files by names (create file, delete file).
Once the file is created this means need to (open) it and use it. Then it may need to (read), (write), and (reposition) [rewriting to end of file].
Finally (close) the file was indicating that no longer using it. Also these are valid for directories.
In addition, for either files or directories, it needs to be able to determine the values of various attributes, and perhaps rest them if necessary (get file attributes) and (set file attributes). File attributes indicate the file name, file type, protection codes, accounting information and so on.
3- Device management

A program as it running may need additional resources to proceed (e.g. more memory, access to file and so on). If the resources are

available they can be granted and control return to user program, otherwise, the program must wait until resource is available. Files can be thought of as virtual devices. Thus many of the system calls for files are also needed for devices. If there are multiple user of the system, however, we must first (request device) after user were finished with the device then user must (release device) (similar to open and close).
Once the device has been requested (and allow to use), the users can (read), (write) the device with tapes
4- Information maintenance

Much system calls exits simply for the purpose of transferring information between the user program and the operating system. Most systems has a system call to return the current time and date. Other system call may return information about the system, such as the number of current users, version number of operating system. The amounts of free memory or disc space, and so on.

5- Communication

There are two common models of communication refer to figure (2-1 (a))

• Message passing model

Here, information is exchanged through an inter-process- communication facility provided by the operating system. Before communication can take place, a connection must be opened. The name of the other communicator must be known to it, being a process on the same computer, or a process on another computer in the network. Each computer in the network has a host name by which it is identified. Also each process has a process name or identifier, by which the operating system can refer to it. The gets host id and gets process id system calls do this translation. These identifiers are then passed to the general purpose open and close system calls provided by the file system, or to

specific open connection and close connection system calls depending on the model of communications. The receiver process usually must give its permission for communication to take [lace with an a accept connection system call. The client and server, then exchange messages by read message and write message system calls. The close connection system call terminates the communication.

• Shared –memory model

Process use map memory system calls to gets access to regions of shared memory refer to figure (2-1(b)). They may then exchange information by reading and writing data in these shared areas. Two Processes should not write to the same location simultaneously.

2- System Programs:

Another aspect of a modern system is the collection of system programs. System programs provide a more convenient environment for program development (creation, debugging) and execution. Some of them are user interfaces to system calls. They can be divided into several categories: -

• File manipulation :

These system programs create, delete, copy rename, print, dump, list, and generally manipulate files and directories.

Status information:
Some programs ask the system for the date, time, amount of available Memory or disc space, number of users, or similar Status information. That information is then formatted, and is printed to the terminal or other O/P device or file.

• File modification:

Several text editors may be available to create and modify the content of files stored on disc or tape.

19

figure 2-1 communication models

Programming- language support: Compilers, assemblers, and interpreters for common programming language (such as Fortran, Cobol, Pascal, Basic, and C) are often provided to the user with the operating system or individuality priced.

• Program loading and execution:

Once a program is assembled or compiled, it must be loaded into memory to be executed.

The system may provide absolute loaders, re-locatable loaders, and linkage editors and overlay loaders. Debugging systems are also required.

20

• Communications :

These programs provide the mechanism for creating virtual connections among processes users, and different computer systems.

They allow users to send messages to each others screens, to send electronic mail or to transfer files from one machine to another and even to use other computers remotes as though these machines were local (known as remote login).

• Application programs:

Most operating system are supplied with programs that are useful to solve common problems, or to perform common operating e.g. web browsers, word processors and text formatters, spreadsheets, data base systems, compiler compilers, plotting and statistical- analysis packages, and games.

Perhaps the most important system program for an operating system is the command interpreter. The main function of which is to get and execute he next user –specified command. How these commands can be implemented? UNIX implements most commands by special system programs. In this case, the command interpreter does not “understand” the command in any way; it merely uses the command to identify a file to be loaded into memory and executed, thus a command delete G, would search for a file called delete load the file into memory and execute it with the parameter G . The function

associated with the delete command would be defined completely by the code in the file delete. Advantages. Programmers can add new commands to the system easily by creating new files of the proper name. Here no modifications are required in the command interpreter.
21

2.4 System Structure

How the operating system components are interconnected and melded into a kernel?

UNIX layer structure:

As do most computer system Unix consists of two separable parts: kernel and the system program. Everything below the system call interfaces and above the physical hardware is the kernel refers to figure (2-2). The kernel provides the file system, CPU scheduling, memory management, and other operating system functions through system calls. System programs use the kernel-supported system calls to provide useful functions such as compilation and file manipulation.

System calls define the programmer interface to Unix; the set of system program commonly available defines the user interface. The programmer and user interface defines the context that the kernel must support.

Most system programs are written in C, and the Unix programmer's manual presents all system calls as C function. A system program written in C for one system can generally be moved to another system and simply recompiled, even though the two systems may be quite different. Only the compiler knows the details of such calls. This feature is a major reason for the portability of Unix programs. System calls for Unix can be roughly general into three categories: file manipulation, process control, and information manipulation instead of four. This is because devices in Unix are treated as (special) files, the same system calls support both files and devices.

System Generation:

It is possible to design, code, and implement an operating system specifically for one machine at one site. More commonly, however, operating systems are designed to run any of a class of machines at a variety of sites with a variety of peripheral configurations. The system
22

Figurer 2-2Unix system structure

must then be configured or generated for each specific computer site. This process is sometimes known as system generation (SYSGEN).

The operating system is normally distributed on disk. To generate a system it must use a special program. The SYSGEN Program (Reads from a given file

or asks the operator of the system for information concerning the specific configuration of the hardware system, or passed
The hardware directly to determine what components is there. The following kinds of information must be determined.
• What CPU is to be used? What options is installed? (Floats point …etc). For multiple CPU systems, each CPU must be described.
• How much memory is available? (To define legal addresses).
23
• What devices are available? The system will Read to know how to address each device (the device number), the device interrupt number. The device type and model, and any special device characteristics.
• What operating system functions are desired? These options might include how many buffers of which size should be used, what type of CPU scheduling algorithm is desired, what is maximum number of process to be supported and so on.
Once this information is determined, it can be used in several ways. The simplest way, it can be used to modify a copy of the same code of the operating system. The operating system would then be completely compiled. An output object version of the operating system that is tailored to the system described is now produced.
After an operating system is generated, it must be made available for use by hardware but how does the hardware know where the kernel is, or how to load it? The procedure of starting a computer by loading the kernel is known as booting the system.
On most computer systems, there is small piece of code, stored in read only memory (ROM), known as the bootstrap loader. This code is able to locate the kernel, load it into main memory, and start it execution.

Operating systems are now almost always written in high – level language (e.g. Unix in C) this feature improves their implementation, maintenance, and portability .To create an operating system for a particular machine configuration users must perform system generation.

2.5 Operating Systems Course Description

An Operating System is a complex software package that manages the resources of a computer system, and provides the base upon which applications can be written. In this course students will study the basic components of an operating system, their functions, mechanisms, policies and techniques used in their implementation and several
24
examples from popular operating systems. The components which will be discussed, include:

2.5.1. Processes

A process is a program in execution. The components of a process are: the program to be executed, the data on which the program will execute, the resources required by the program—such as memory and file(s)—and the status of the execution.

Interprocess communication:

Cooperating processes need to exchange information, as well as synchronize with each other, to perform their collective task(s).

Methods for effective sharing of information among cooperating processes are collectively known as inter-process communication (IPC). Two basic models are used:

– Shared memory: “shared data” are directly available to each process in their address spaces.

– Message passing: “shared data” are explicitly exchanged.

There are two basic operations on messages:

– Send (): transmission of a message.

– Receive (): receipt of a message.

The OS component which implements these operations (and more) is called a ‘‘message passing’ ’ system.

2.5.2 Scheduling

In multiprogramming systems, when there is more than one runable process (i. e., ready), the operating system must decide which one to activate. The decision is made by the part of the operating system called the scheduler, using a scheduling algorithm.

Types of scheduling:

In general, (job) scheduling is performed in three stages: short-, medium-, and long- term. The activity frequencies of these stages are implied by their names.

Medium- term scheduling involves suspending or resuming processes by swapping (rolling) them out of or into memory. Long- term (job) scheduling is done when a new process is created. It initiates processes and so controls the degree of multi- programming (number of processes in memory).

Short- term (process or CPU) scheduling occurs most frequently and decides which process to execute next.

Long- & medium- term scheduling:

Acting as the primary resource allocator, the long-term scheduler admits more jobs when the resource utilization is low, and blocks the incoming jobs from entering the ready queue when utilization is too high.

When the main memory becomes over- committed, the medium- term scheduler releases the memory of a suspended (blocked or stopped) process by swapping (rolling) it out.

Short- term scheduling:

Short- term scheduler, also known as the process or CPU scheduler, controls the CPU sharing among the ‘‘ ready’’ processes. The selection of a process to execute next is done by the short- term scheduler.

The following are some common scheduling algorithms:

• First- Come- First- Served (FCFS)
• Shortest Job First (SJF)
• Round- Robin (RR)
• Priority- based

2.5.3. Synchronization

Concurrent processes:

In a multiprogramming environment, processes executing concurrently are either competing for the CPU and other global resources, or cooperating with each other for sharing some resources.

An OS deals with competing processes by carefully allocating resources and properly isolating processes from each other. For cooperating processes, on the other hand, the OS provides mechanisms to share some resources in certain ways as well as allowing:

Processes to properly interact with each other. Cooperation is either by sharing or by communication.

Mutual exclusion:

Mutual exclusion is a mechanism to ensure that only one process (or person) is doing certain things at one time, thus avoid data inconsistency. All others should be prevented from modifying shared data until the current process finishes.

Critical section:

A section of code, or a collection of operations, in which only one process may be executing at a given time and which we want to make ‘ ‘sort of’’ atomic.

Atomic means either an operation happens in its entirety or NOT at all; i. e., it cannot be interrupted in the middle.

Atomic operations are used to ensure that cooperating processes execute correctly.

Mutual exclusion mechanisms are used to solve the critical section problem.

Semaphores:

A semaphore is a synchronization variable (guard) that takes on non-negative integer values with only two atomic operations.

27

Semaphores are simple, and allow the solution of many interesting problems. They are useful for more than just mutual exclusion.

Type of semaphores:

Semaphores are usually available in two flavors:

• Binary —is a semaphore with an integer value of 0 and 1.

• Counting —is a semaphore with an integer value ranging between 0 and an arbitrarily large number. Its initial value might represent the number of units of the critical resources that are available. This form is also known as a general semaphore.

2.5.4. Deadlocks

Deadlock is defined as the permanent blocking of a set of processes that compete for system resources, including database records and communication lines. Unlike some other problems in multiprogramming systems, there is no efficient solution to the deadlock problem in the general case.

Deadlock occurs when sets of processes are in a wait state, because each process is waiting for a resource that is held by some other waiting process. herefore, all deadlocks involve conflicting resource needs by two or more processes.

Figure :- (2-3) Resource allocation graph

28

2. 5. 5 Memory Management

The CPU fetches instructions and data of a program from memory; therefore, both the program and its data must reside in the main (RAM and ROM) memory.

Simple management schemes:

An important task of a memory management system is to bring (load) programs into main memory for execution.

Fragmentation:

Fragmentation refers to the unused memory that the management system cannot allocate.

• Internal fragmentation

Waste of memory within a partition, caused by the difference between the size of a partition and the process loaded.

Severe in static (fixed) partitioning schemes.

• External fragmentation

Waste of memory between partitions, caused by scattered noncontiguous free space. Severe in dynamic (variable size) partitioning schemes.
Compaction is a technique that is used to overcome external fragmentation.
Swapping:

The basic idea of swapping is to treat main memory as a ‘ ‘pre-emptable’’ resource. A high- speed-swapping device is used as the backing storage of the preempted processes.
SWAP- IN.
SWAP- OUT
Swapping is medium- term scheduling methods refer to figure (2-4).
Swapping brings flexibility even to systems with fixed partitions, because:

“ If needed, the operating system can always make room for high- priority jobs ’ ’

Figure:- (2-4) Swapping in a medium – term scheduling method

Paging:

Physical memory is divided into a number of fixed size blocks, called frames. The logical memory is also divided into chunks of the same size, called pages.
Memory caching:

Frequently used data in main memory; can also be stored in fast buffers, called cache memory, or simply cache.
Cache terminology:
Cache hit : item is in the cache.
Cache miss : item is not in the cache; must do a full operation.
Categories of cache miss :
• Compulsory: the first reference will always miss.
• Capacity: non- compulsory misses because of limited cache size.

2.5.6. Virtual Memory

Virtual memory; a program can run with only some of its virtual address spaces in main memory.
Principles of operation:

The basic idea with virtual memory is to create an illusion of memory that is as large as a disk (in gigabytes) and as fast as memory (in nanoseconds).
Virtual memory:

Virtual memory (sub-) system can be implemented as an extension of paged or segmented memory management or sometimes as a combination of both.

2.5.7 Secondary Storage

Secondary storage is the non- volatile repository for (both user and system) data and programs.

Uses of secondary storage include storing various forms of programs (source, object, executable) and temporary storage of virtual memory pages (paging device or swap space).

Information in secondary storage may be in a variety of forms, including readable text and raw data (e. g., binary).

2.5.8. Protection and security

Protection is a mechanism to control access to resources.

Protection mechanisms provide controlled access by limiting the type of access that various users can make.

Security is a measure of confidence that the integrity of a (computer) system relies on.

A security system prevents unauthorized access to a system.

Goals of protection:

• Prevent accidental and maliciously destructive behavior.

• Ensure fair and reliable resource usage.

• Provide a mechanism for the enforcement of the policies governing resources use.

Policy: what is to be done.

Mechanism: how something is to be done.

Security:

Protection is an internal (within a computer system) problem.

Security includes external environments as well.

Security measures:

– Physical: securing a site from intruders.

– Human: eliminating bribery.

Security violations: unauthorized:

– Reading of data

– Modification of data

– Destruction of data

Authentication:

Ability to identify a legitimate user from malicious ones. Based on some combination of three sets of items:

– User possession (a key or a card)

– User knowledge (a user ID or password)
– User attributes (fingerprint, retina pattern, and signature)
Encryption:
A common method of protecting information transmitted over unreliable links. Basic encryption mechanism is as follows:
• The information is encrypted from its initial form (clear text) to an internal form (cipher text).
• The cipher text can be stored or transmitted.
• The receiver decrypts the cipher text back to clear text.
There are two common techniques:
• Secret key
• Public key.

[1] [2] [3] [13]

CHAPTER 3

MANUAL DESIGN

This laboratory is intended as a general introduction to the techniques used to implement operating systems and related kinds of systems software. Among the topics covered will be process management (creation, synchronization, and communication); processor scheduling; deadlock prevention, avoidance, and recovery; main-memory management; virtual memory management (swapping, paging, segmentation and page-replacement algorithms); control of disks and other input/output devices; file-system structure and implementation; and protection and security. Students are going to look at and improve the code for the components that deal with scheduling, context switching, process management, message passing, memory management, interrupt processing.

This laboratory helps the students to gain well understanding of modern techniques and gives good link to the related theories. This provides students with power full tools for their practical life in the future.

3.1. Manual Objectives

This laboratory builds on the broad overview of operating system principles gained from completing operating systems course Laboratory.

COURSE GOAL:

To provide students with a deep understanding of modern operating system technology, implementation techniques and research issues.

OBJECTIVES:

Technical:

Provide in-depth coverage of modern operating system issues, such as:
• Micro-kernels and inter process communication (IPC).
• Performance.
• Kernel design and implementation.
• Scheduling for operating system.
• Alternative protection and security models.

Educational:

• Exposing students to current operating systems research and modern OS technology.

• Providing experience in low-level systems programming in a realistic development environment.

• Encouraging interest in further study and research in the area.

Professional:

• Working in an environment and on problems similar to a professional OS or embedded systems implementor in industry.

• Building a whole system almost from the hardware up.

Understanding lowest-level OS code and its interaction with hardware.

Pre-requisites

Operating System is the capstone course for student Bachelor degree within the Department of Computer Engineering. The course requires the use of many skills that students have developed and refined over the last several semesters. Due to the skill level required and inherent difficulty of this course, it is required that student have successfully completed the pre-requisite courses. Successful completion means that student have completed the course in a semester prior to the

current one, All students should have successfully completed a course in Computer Architecture and should have Completed C++ course. Computer Architecture will help with the hardware concepts, and C++ will provide broader exposure to programming languages.

NOTE: It is NOT the intent of this course to teach student how to program in a new programming language. Since student are all seniors in Computer Engineering this means student should be able to pick up the essentials of any programming language within a few weeks.

3.2 Manual Organization:

This laboratory manual briefly describes each experiment's aims and some of the theory. A more in-depth description is given with the actual experiment.

• The first part of the documentation for each individual experiment is simply a list of the aims.

• This is followed by a more in-depth discussion of the theory associated with that particular experiment. Note that in some cases there are links back to the main text of the coursework modules in the operating systems course.

• The next stage is the procedure. This outlines what input parameters the program requires and then what results are generated.

• Finally the student can run the program

3.3 Manual Usage:

This manual consists of two parts; part 1 contains laboratory experiments the instructor select to handed to the student as laboratory work for the course. Part two contains a comprehensive list of operating system resources; these resources include a glossary of operating system terms, information on the resources in the operating system laboratory,

and a description of the commands used throughout the experiments. At the end of this manual some solutions of experiments were presented.

3.3.1 Students work

Students are advised to refer to their course notes and literature for more information about the topics introduced by the laboratory experiment.

Students will submit working source code with test programs and a brief lab assignment write-up. Students will schedule a time to demonstrate their working lab assignments to the teacher assistants (TA) in the lab. All students members should be prepared to answer detailed questions about the implementation of the All-technical questions, regarding the labs should be directed to the teacher assistants first. The teacher assistants will be responsible for managing the lab assignments. Students should enjoy their guidance as much as possible.

CHAPTER FOUR
LABORATORY EXPERIMENTS
Experiments Design

The manual contains a set of experiment aims to supplement the operating systems classroom instruction by providing a more detailed and hands-on experience on the analysis and design of operating systems. To

demonstrate the concepts of operating system course, case studies of operating system will be presented and a programming lab assignment will be an integral part of the course.
Laboratory experiments

4.1 Operating systems Experiments

This part of thesis contains a list of the experiment classes. The 14 experiments listed here are to be performed throughout the semester to aid in the learning of operating system concepts. These experiments were designed to teach core operating system concepts and provide useful, real-world coding examples. These coding examples attempt to show the student how the operating system works and how to interact with it.

4.2 Experiments Table of Contents

Experiment #1: Time

Purpose: The primary focus of this experiment will be "time." The experiment will show student several methods by which to measure time in user applications. It will introduce the concept of time and present several important features that need to be understood about measuring time in an application. Finally, it will provide pointers to other methods that can be used to measure the time and performance of applications.

Experiment #2: Multi-Tasking

Purpose: The concept of multi-tasking is not a new one, the basic concepts of multi-tasking will be introduced along with calls necessary to manage multiple processes.
This experiment deals with the management of multiple processes in the UNIX environment. Multiple processes can be used to perform multiple tasks simultaneously There are several UNIX system calls that are standard across all UNIX platforms that allow processes to be created, managed, and destroyed.

Experiment #3: Shared Resources

This experiment will teach the student how to use shared resources to communicate between the processes. This experiment will focus on two primary methods of interprocess communications, shared memory and message queues. However, before learning how to create and remove System V IPC constructs in C, two commands will prove very useful throughout this and future experiments. First, ipcs commands/ipcscan be used to list the allocated System V IPC resources. Second, ipcrm commands/ipcrmcan be used to remove allocated System V IPC resources.

Experiment #4: Signals

The purpose of this program is to measure the time it takes for a signal to be passed from one process to another.

Process to execute code in an asynchronous manner. In another words, the execution order within the program is no longer determinant.

Experiment #5: Semaphores

Purpose: The purpose of this program is to demonstrate how semaphores

can be used to protect a critical region. Its sole purpose is to print a character string (namely the alphabet) to the screen. Any number of processes can be used to cooperatively (or non-cooperatively) print the string to the screen. An index is stored in shared memory, this index is the index into the array that identifies which character within the string should be printed next. Without semaphores, all the processes access this index simultaneously and conflicts occur. With semaphores, the character string is displayed neatly to the screen.

Experiment #6: Scheduling, Priorities,

Purpose: in this experiment the program demonstrates the priority inversion of two processes. To demonstrate this, three processes are created along with one shared memory segment and one semaphore. The shared memory segment is used to allow the processes to communicate who has the lock. The semaphore is the critical resource that both two processes need.

Three processes will be started, a high priority process, a middle priority process, and a low priority process.

Experiment #7: Interfacing with an External Device:

There are many "external-devices" that a system must communicate with. There are hundreds of devices that a system must communicate with. Image for a minute if each of these devices took a different set of function calls to access. User can see this would lead to thousands of different function calls that would be needed to communicate with devices. Therefore, the developers of UNIX came up with a standard way to access devices. First, they are treated like files. User can open (), close (), write (), and read () to and from UNIX devices.

Purpose: The source file contains the calls used to access the serial library that is used for Experiments #7 and #8. Two calls are used to access the

serial library: int ttyOpen(char *device, int flags) , int ttyClose(int fp) ttyOpen(). Takes a character string that points to the Device to open. The flags are the flags that are passed to the open call .The file pointer to the file is returned on success.

ttyClose() takes a file pointer to an open file. That open file is closed. 0 is returned on success.

Both ttyOpen() and ttyClose() perform additional operations such as saving the current state of the terminal on open and restoring on close. ttyOpen() also sets the terminal in raw mode so that data can be sent without worrying about it being processed.

Experiment #8: A Simple operating system (Data Acquisition):

Purpose: Experiment #8 will create a simple, but complete, data acquisition system. It will use the concepts talked about in the previous experiments in a real-life setting.

The 'simple data system' will be a multi-process Operating System application which collects data via a serial port, decomutates the data, processes the data, and stores it to a file on disk. A device simulator is also created to send simulated data across the serial port. The system uses shared memory to communicate status between processes. Message queues are used to pass data between processes. The serial concepts talked about in Experiment #7 are also addressed. Signals are sent by the device driver to the serial processes and are also used as a communication method.

The data rates for this system are low, and there are no real performance

requirements. Any system should be able to run this system while maintaining the Operating System requirements (i.e. responding to the serial port). A serial port is a low data rate device, so this is expected.

As a final note, this experiment is a little different than the previous ones. The actual source code of the system will not be described in detail. Rather, the general concepts and design of the system will be addressed. For more information on the source code, see the code itself. The code is well commented and the comments should adequately explain the code.

Experiment #9 Interrupt Service Routines

Purpose: The user may write an interrupt service routine (ISR) and attach it to a particular interrupt using the intConnect routine provided. The time span is generally knonw as interrupt latency. Because many interrupts may occur within a short time of each other and a higher interrupt will block

lower priority interrupts, it is necessary to keep the ISR processing to a minimum. This is the responsibility of the application writer.

The next five experiments is lab exercises refer to a reference book:

Operating Systems: A Modern Perspective written by Gary J.Nuntt. About this part of Lab Exercises:

There are several laboratory exercises sprinkled throughout the book (at the end of chapters). These exercises have considerable hands-on information about how parts of the OS are built. Even if students don't actually solve the exercises, student can learn about specific implementation by browsing the background information in the lab manuals. Some of the exercises are specific to Windows operating

systems, and some are specific to UNIX; a few apply to both classes of operating systems.

Experiment #10 A Shell Program, Chapter 2:

This is a UNIX exercise that uses the fork() and exec() system calls. If student have never done any concurrent programming on UNIX, this is a great place to learn about it.

Experiment #11 Observing OS Behavior, Chapter 6:

This is a Linux or Solaris (or any other UNIX family member that supports the /proc file system) exercise. It is not difficult to solve, but provides interesting insight into kernel behavior. If student solved only one exercise in the entire book, this would be a candidate.

Experiment #12 Refining the Shell, Chapter 9:

This is a UNIX exercise to refine the shell developed as an exercise in Chapter 2. The Chapter 2 version focuses on fork () and exec (), and this one extends into file redirection and more concurrency.

Experiment #13 A Simple File Manager, Chapter 13:

This exercise can be solved in Windows or UNIX operating systems. The file system the student implement is simple, but the exercise gives student plenty of opportunities to get the flavor of writing a file manager. A full solution will have quite a bit of code.

Experiment #14 Using TCP/IP, Chapter 13:

This exercise can actually be solved on any OS that supports BSD sockets --, as do most versions of UNIX and the Windows family. Besides the experience with network programming, this exercise has an interesting sidelight into using non-blocking read operations.

4.3 Laboratory report writing guidelines

Description of the Report:

For each of the operating systems experiments Table of Contents Presented in these pages, the student is required to prepare and submit a laboratory report detailing the experiment.

Each laboratory report submitted by the student will have, at least, the following sections:

Title Page:

This page should be fairly self-explanatory. Include, at a minimum, the experiment number and name, student name, the date submitted, the college's name, the person it was submitted to, the class number, and the class name.

Table of Contents:

Again, this should be fairly self-explanatory. All pages should be numbered, include immediately after the title page a list of the contents, by section, along with the page number where each section starts.

Introduction:

The introduction should be no more than two to three paragraphs long. In general the introduction should "introduce" the reader to the topic and tell them what they should expect to find in the rest of the paper.

Results of Running Sample Program:

This section may not be appropriate for all experiments. However, a majority of the experiments require the student to take measurements

using the sample program provided in the experiment. This section should contain the results of running the sample program. The data that should be collected for each experiment is described in the assignment summary at the end of the experiment.

Analysis of Results:

In this section the results that were presented above are analyzed, discussed, and explained. This section should attempt to "make sense" out of the results that presented above.

Conclusion:

The conclusion is restating what student has already told the reader. The conclusion should reinforce the information already presented to the user. It should briefly summarize the experiment.

Appendix (i.e. Source Code):

The appendix should contain the source code that was produced as part of the experiment.

Remember that these are only general guidelines and the student should exercise common sense. Student reports may not and need not be perfect immediately. However, by the end of the semester student should be used to writing in a brief, precise, and informative technical style.

CHAPTER FIVE

CONCLUSION

This thesis covers concepts in concurrent programming and operating systems. The thesis consists of laboratory lectures ،to have assistance with programming assignments .

Prerequisites :Introductory courses in computer programming and data structures .

Goals of the course :To teach the fundamental concepts of concurrent programming and operating systems .

Recommended Textbook :It is recommended that user purchase a text on C/C++ to help learn the language and complete the laboratory assignments.

Laboratory Material: The laboratory manual is available as textbook. To solve the programming assignments this required using several different operating systems and machines. Additional reference material is available in the laboratory manual.

The thesis covered:

1. Overview, Introduction to operating system.
2. System Components, Processes. Process synchronization.
3. Deadlocks and solutions.
4. Memory Management. Virtual Memory.
5. File systems, Input/Output. Distributed file systems.
6. Privacy and Security.

Its recommended to evaluate the Student's performance after using this manual for one academic year to assess if the manual satisfies its academic goals, and introduce any necessary improvements as needed.

References

1. **Operating System**
A Design- Oriented Approach
University of new mexico
By: Charles Crowley

2. Operating System Concepts
Fifth edition
By: Abraham Silberschatz
Bell lab/Beter Baer Galvin
3**. Operating Systems A modern Perspective**
Second Edition
University of Colorado
By: Gary J.Nutt

Sites

1. http://hpdrc.cs.fiu.edu/Sem-DDS/documentation.html
2. http://www.linuxfocus.org/English/March2000/article142.shtml
3. http://www.redhat.com/docs/manuals/linux/RHL-6.2-Manual/getting-started-guide/ch-history.html
4. http://www.microsoft.com/windows/WinHistoryDesktop.mspx
5. http://www.cs.wpi.edu/~claypool/courses/3013-C01/
6. http://www.rt.db.erau.edu/

46

7. http://www.clarkson.edu/~akersmc/linux7.html
8. http://www.cis.ohio-state.edu/~agrawal/662/
9. http://www.rt.db.erau.edu/
10. http://www.math.grin.edu/~walker/courses/213.fa00/lab-semaphores.html

Operating Systems Topics & Lecture Notes

1.Refer To **Dr: ELRASHED OSMAN ELKHIDER** lectures note of engineering operating systems course
University of Khartoum
Department of Electronics & Electrical Engineering

47

APPENDIX A

LABOOROTRY MANUAL, PART 1. LABOROTORIES

APPENDIX B

LABOOROTRY MANUAL, PART 2. LOCAL RESOURCE AND SOME EXPERIMENTS SOLUTIONS

[Refer to the site: www. Erau.edu]

4849