UNIT 5 INTRODUCTION TO OPERATING SYSTEM

Structure

5.0 OBJECTIVES

After going through this unit, you should be able to:

- list and interpret the basic functions of operating systems;
- to make comparison among different types of operating systems;
- compare different structures (models) of operating system; and
- list future trends in operating system design.

5.1 INTRODUCTION

An Operating System is a system software which may be viewed as an organised collection of software consisting of procedures for operating a computer and providing an environment for execution of programs. It acts as an interface between users and the hardware of a computer system.

There are many important reasons for studying operating systems. Some of them are:

1) User interacts with the computer through operating system in order to accomplish his task since it is his primary interface with a computer.

2) It helps users to understand the inner functions of a computer very closely.

3) Many concepts and techniques found in operating system have general applicability in other applications.

Compilers, Assemblers, Software utilities like Lex, Yacc and GUIs (MS-WINDOW etc.). Software have been developed under a particular operating system environment. The introductory concepts and principles of an operating system will be the main issues for the discussion in this unit. Evolution of operating system, types and models of operating systems will also be broadly covered here.

5.2 WHAT IS AN OPERATING SYSTEM?

An operating system is an essential component of a computer system. The primary objectives of an operating system is to make computer system convenient to use and utilise computer hardware in an efficient manner.

An operating system is a large collection of software which manages resources of the computer system, such as memory, processor, file system and input/output devices. It keeps track of the status of each resource and decides who will have a control over computer resources, for how long and when. The positioning of operating system in overall computer system is shown in figure 1.

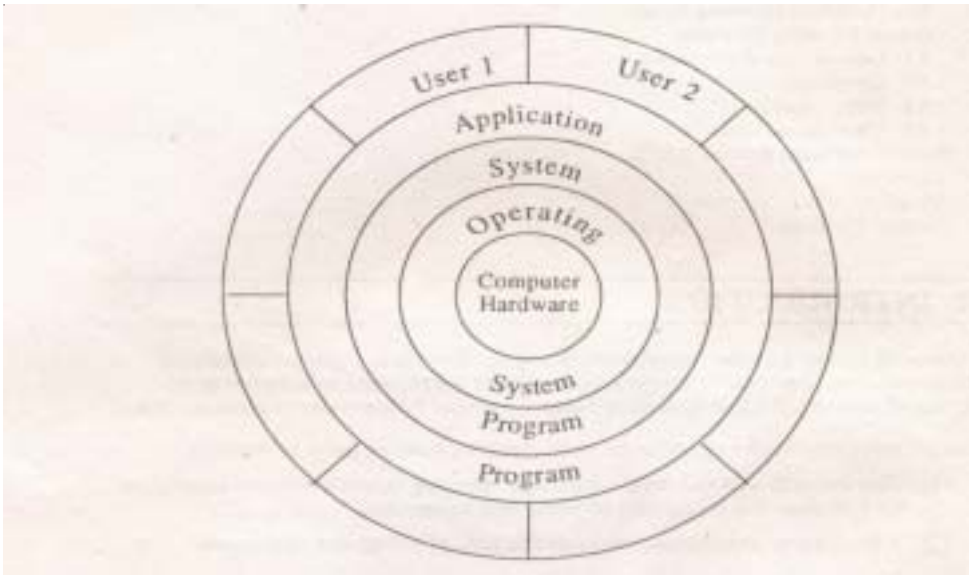


Fig. 1: Component of Computer System

From the diagram, it is clear that operating system directly controls computer hardware resources. Other programs rely on facilities provided by the operating system to gain access to computer system resources. There are two ways one can interact with operating system:

i) By means of Operating system call in a program

ii) Directly by means of Operating System Commands.

System Call: System calls provide the interface to a running program and the operating system. User program receives operating system services through the set of system calls. Earlier these calls were available in assembly language instructions but nowadays these features are supported through high-level languages like C, Pascal etc., which replaces assembly language for system programming. The use of system calls in C or pascal programs very much resemble pre-defined functions or subroutine calls.

As an example of how system calls are used, let us consider a simple program to copy data from one file to another. In an interactive system, the following system calls will be generated by the operating system:

- Prompt messages for inputting two file names and reading it from terminal.
- Open source and destination file.
- Prompt error messages in case the source file cannot be open because it is protected against access or destination file cannot be created because there is already a file with this name.
- Read the source file.
- Write into the destination file.
- Display status information regarding various Read/Write error conditions. For example, the program may find that the end of the file has been reached or the there was a hardware failure. The write operation may encounter various errors, depending upon the output device (no more disk space, physical end of tape, printer output of paper and so on).
- Close both files after the entire file is copied.

As we can observe, a user program makes heavy use of the operating system. All interaction between the program and its environment must occur as the result of requests from the program to the operating system.

Operating System Commands: Apart from system calls, users may interact with operating system directly by means of operating system commands.

For example, if you want to list files or sub-directories in MS-DOS, you invoke dir command. In either case the operating system acts as an interface between users and the hardware of a computer system. The fundamental goal of computer systems is to solve user problems. Towards this goal computer hardware is designed. Since the bare hardware alone is not very easy to use, programs (software) are developed. These programs require certain common operations, such as controlling peripheral devices. The command function of controlling and allocating resources are then brought together into one piece of software; the operating system.

To see what operating systems are and what operating systems do, let us consider how they have evolved over the years. By tracing that evolution, we can identify the common elements of operating systems and examine how and why they have developed as they have.

5.3 EVOLUTION OF OPERATING SYSTEMS

An operating system may process its task serially (sequentially) or concurrently (several tasks simultaneously). It means that the resources of the computer system may be dedicated to a single program until its completion or they may be allocated among several programs in different stages of execution. The feature of operating system to execute multiple programs in interleaved fashion or different time cycles is called as multiprogramming systems. In this section, we will try to trace the evolution of operating system. In particular, we will describe serial processing, batch processing and multiprogramming.

5.3.1 Serial Processing

Programming in 1's and 0's (machine language) was quite common for early computer systems. Instruction and data used to be fed into the computer by means of console switches or perhaps through a hexadecimal keyboard. Programs used to be started by loading the program computer register with the address of the first instruction of a program and its result (program) used to be examined by the contents of various registers and memory locations of the machine. Therefore, programming in this style caused a low utilisation of both users and machine.

Advent of Input/Output devices, such as punched cards paper tape and language translators (Compiler/Assemblers) brought a significant step in computer system utilization. Program started being coded into programming language that are first changed into object code (binary code) by translator and then automatically gets loaded into memory by a program called loader. After transferring a control to the loaded program, the execution of a program begins and its result gets displayed or printed. Once in memory, the program may be re-run with a different set of input data.

The process of development and preparation of a program in such environment is slow and cumbersome due to serial processing and numerous manual processing. In a typical sequence first the editor is called to create a source code of user program written in programming language, translator is called to convert a source code into binary code and then finally loader is called to load executable program into main memory for execution. If syntax errors are detected, the whole process must be restarted from the beginning.

The next development was the replacement of card-decks with standard input/ output and some useful library programs, which were further linked with user program through a system software called linker. While there was a definite improvement over machine language approach, the serial mode of operation is obviously not very efficient. This results in low utilization of resources.

5.3.2 Batch Processing

Utilisation of computer resources and improvement in programmer's productivity was still a major prohibition. During the time that tapes were being mounted or programmer was operating the console, the CPU was sitting idle.

The next logical step in the evolution of operating system was to automate the sequencing of operations involved in program execution and in the mechanical aspects of program development. Jobs with similar requirements were batched together and run through the computer as a group. For example, suppose the operator received one FORTRAN program, one COBOL program and another FORTRAN program. If she runs them in that order, she would have to set up for FORTRAN program environment (loading the FORTRAN compiler tapes) then setup COBOL program and finally FORTRAN program again. If he runs the two FORTRAN program as a batch, however he could set up only once for FORTRAN thus saving operator's time.

Batching similar jobs brought utilisation of system resources quite a bit. But there were still problems. For example, when a job is stopped, the operator would have to notice that fact by observing the console, determine why the program stopped and then load the card reader or paper tape reader with the next job and restart the computer. During this transition from one job to the next, the CPU sat idle.

To overcome this idle time, a small program called a resident monitor was

created which is always resident in the memory. It automatically sequenced one job to another job. Resident monitor acts according to the directives given by a programmer through control cards which contain informations like marking of job's beginnings and endings, commands for loading and executing programs, etc. These commands belong to job control language. These job control language commands are included with user program and data. Here is an example of job control language commands.

$COB	—	Execute the COBOL compiler
$JOB	—	First card of a job
$END	—	Last card of a job
$LOAD	—	Load program into memory
$RUN	—	Execute the user program

Figure 2 shows a sample card deck set up for a simple batch system.

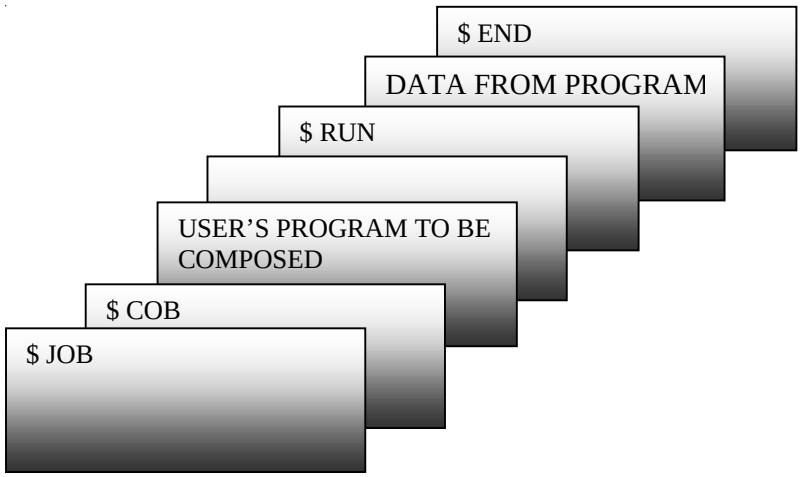


Fig. 2: Card Deck for Cobol Program for a Simple Batch System

With sequencing of program execution mostly automated by batch operating system, the speed discrepancy between fast CPU and comparatively slow input/output devices such as card readers, printers emerged as a major performance bottleneck. Even a slow CPU works in the microsecond range, with millions of instructions per second. But, fast card reader, on the other hand, might read 1200 cards per minute. Thus, the difference in speed between the CPU and its input/output devices may be three orders of magnitude or more.

The relative slowness of input/output devices can mean that CPU is often waiting for input/output. As an example, an Assembler or compiler may be able to process 300 or more cards per second. A fast card reader, on the other hand, may be able to read only 1200 cards per minute. This means that assembling or compiling a 1200 card program would require only 4 seconds of CPU time but 60 seconds to read. Thus, the CPU is idle for 56 out of 60 seconds or 93.3 per cent of the time. The resulting CPU utilization is only 6.7 per cent. The process is similar for output operations. The problem is that while an input/output is occurring, the CPU is idle, waiting for the input/output to complete; while the CPU is executing, input/output devices are idle.

Over the years, of course, improvements in technology resulted in faster input/output devices. But CPU speed increased even faster. Therefore, the need was to increase the throughput and resource utilization by overlapping input/output and processing operations. Channels, peripheral controllers and later dedicated input/output processors

brought a major improvement in this direction. DMA (Direct Memory Access) chip which directly transfers the entire block of data from its own buffer to main memory. CPU requires to be interrupted per block only by DMA. Apart from DMA, there are two other approaches to improving system performance by overlapping input, output and processing. These are buffering and spooling.

Buffering is a method of overlapping input, output and processing of a single job. The idea is quite simple. After data has been read and the CPU is about to start operating on it, the input device is instructed to begin the next input immediately. The CPU and input device are then both busy. With luck by the time that the CPU is ready for the next data item, the input device will have finished reading it. The CPU can then begin processing the newly read data, while the input device starts to read the following data. Similarly, this can be done for output. In this case, the CPU creates data that is put into a buffer until an output device can accept it.

If the CPU is, on the average much faster than an input device, buffering will be of little use. If the CPU is always faster, then it always finds an empty buffer and have to wait for the input device. For output, the CPU can proceed at full speed until, eventually all system buffers are full. Then the CPU must wait for the output device. This situation occurs with input/output bound jobs where the amount of input/output relation to computation is very high. Since the CPU is faster than the input/output device, the speed of execution is controlled by the input/output device, not by the speed of the CPU.

More sophisticated form of input/output buffering called SPOOLING (simultaneous peripheral operation on line) essentially use the disk as a very large buffer (fig. 3) for reading and for storing output files.

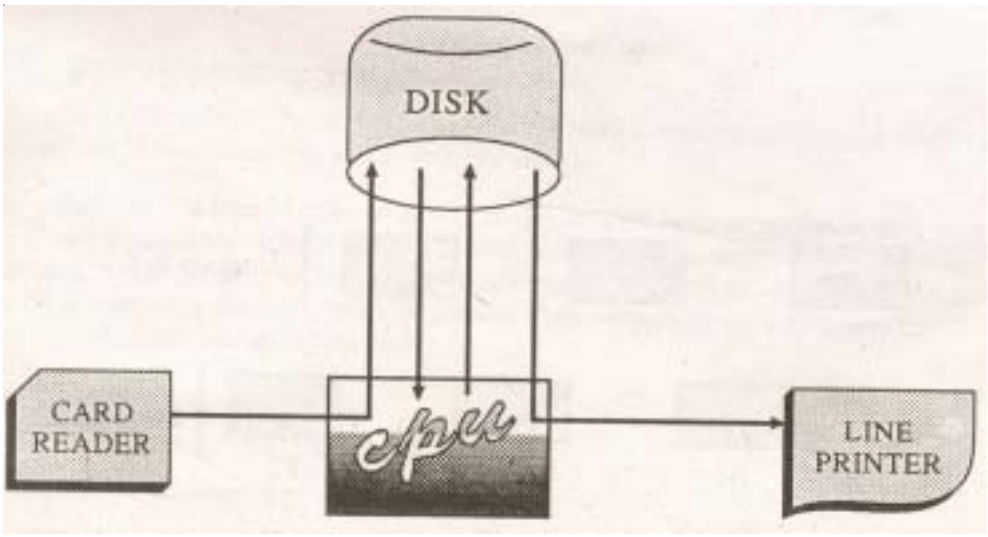


Fig. 3: Spooling

Buffering overlaps input, output and processing of a single job whereas Spooling allows CPU to overlap the input of one job with the computation and output of other jobs. Therefore this approach is better than buffering. Even in a simple system, the spooler may be reading the input of one job while printing the output of a different job.

5.3.3 Multiprogramming

buffering and Spooling improve system performance by overlapping the input, output and computation of a single job, but both of them have their limitations. A single user cannot always keep CPU or I/O devices busy at all times. Multiprogramming offers a more efficient approach to increase system

performance. In order to increase the resource utilization, systems supporting multiprogramming approach allow more than one job (program) to utilize CPU time at any moment. More number of programs competing for system resources, will result in better resource utilization.

The idea is implemented as follows. The main memory of a system contains more than one program (fig. 4).

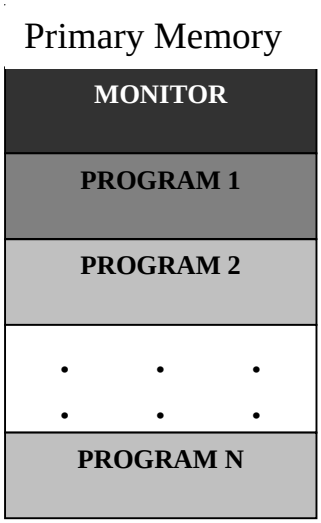
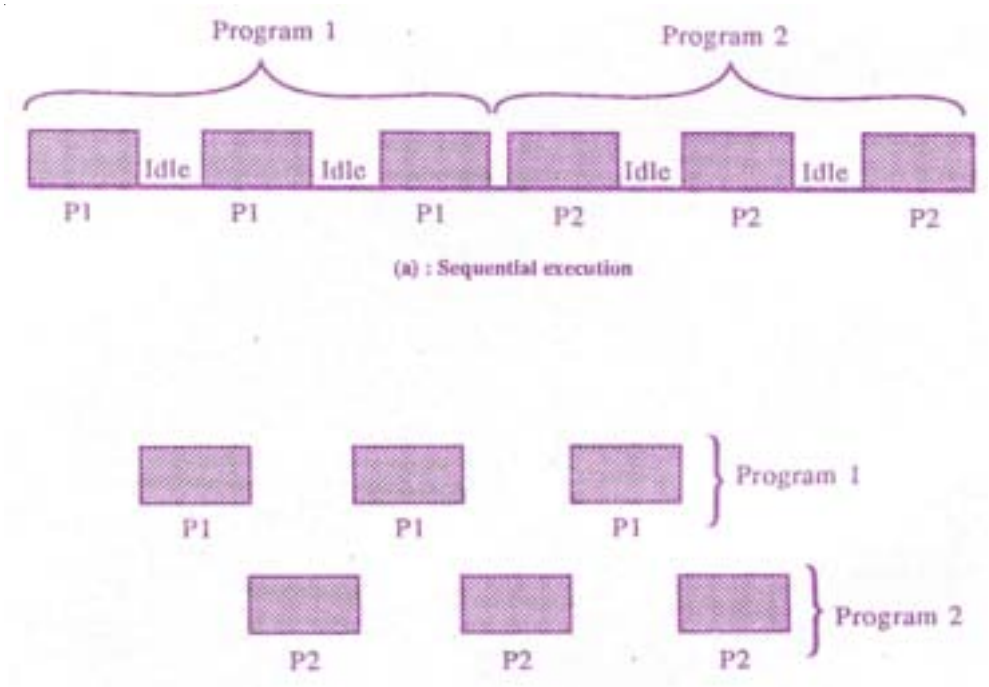


Fig. 4: Memory Layout in Multiprogramming Environment

The operating system picks one of the programs and start executing. During execution process Program 1 may need some I/O operation to complete. In a sequential execution environment (Fig. 5a) the CPU would sit idle. In a multiprogramming system, (Fig. 5b) operating system will simply switch over to the next program (Program 2).



(b): Execution in Multiprogramming Environment
Fig. 5: Multiprogramming

When that program needs to wait for some I/O operation, it switches over to Program 3 and so on. If there is no other new program left in the main memory, the CPU will pass its control back to the previous programs.

Multiprogramming has traditionally been employed to increase the resource utilization of a computer system and to support multiple simultaneously interactive users (terminals). Compared to operating system which supports only sequential execution, multiprogramming system requires some form of CPU and memory management strategies which will be discussed in the next section.

Self-Check Exercise

1) What is a system call? Give several examples of system calls.

2) Define the essential difference between
   a) Spooling
   b) Buffering

3) What are the main advantages of multiprogramming?

Note: i) Write your answer in the space given below.

ii) Check your answer with the answers given at the end of this Unit.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

5.4 TYPES OF OPERATING SYSTEM

In this section, we will discuss general properties, types of applications advantages, disadvantages and basic requirements of different types of operating systems.

Each type of operating system is discussed briefly with regard to the following features :

— Processor scheduling

— Memory management

— I/O management

— File management

5.4.1 Batch Operating System

As discussed earlier during batch processing environment it requires grouping of similar jobs which consist of programs, data and system commands.

The suitability of this type of processing is in programs with large computation time with no need of user interaction/involvement. Some examples of such programs include payroll, forecasting, statistical analysis and large scientific number crunching programs. Users are not required to wait while the job is being processed. They can submit their programs to operators and return later to collect them.

But it has two major disadvantages:

i) Non-interactive environment

ii) Off-line debugging

Non-interactive environment: There are some difficulties with a batch system from the point of view of programmer or user. Batch operating systems allow little or no interaction between users and executing programs. The turn around time taken between job submission and job completion in batch operating system is very high. Users have no control over intermediate results of a program. This type of arrangement does not create flexibility in software development.

The second disadvantage with this approach is that programs must be debugged which means a programmer cannot correct bugs the moment it occurs.

Progress scheduling (i.e. allocation strategy for a process to a processor), memory management file management and I/O management in batch processing are quite simple.

Jobs are typically processed in the order of submission, that is, in the first come, first served basis.

Memory is usually divided into two areas. One of them is permanently fixed for containing operating system routines and the other part contains only user programs to be executed; When one program is over, the new program is loaded into the same area.

Since there is only one program in the execution at a time, there is no competition for I/O devices, therefore, allocation and de-allocation for I/O devices is very trivial.

Access to files is also serial and there is hardly a need of protection and file access control mechanism.

5.4.2 Multiprogramming Operating System

Multiprogramming operating systems compared to batch operating systems are fairly sophisticated. As illustrated in figure 5, multiprogramming has a significant potential for improving system throughput and resource utilization with very minor differences. Different forms of multiprogramming operating system are multitasking, multiprocess and multiuser operating systems. In this section, we will briefly discuss the main features and functions of these systems.

Multitasking Operating Systems: A running state of a program is called a process or a task. A multitasking operating system (also called multiprocessing

operating system) supports two or more active processes simultaneously. Multiprogramming operating system is processes in execution states simultaneously) allows the instruction and data from two or more separate processes to reside in primary memory simultaneously. Note that multiprogramming implies multiprocessing or multitasking operation, but multiprocessing operation (or multitasking) does not imply multiprogramming. Therefore, multitasking operation is one of the mechanism that multiprogramming operating system employs in managing the totality of computer related resources like CPU, memory and I/O devices.

The simplest form of multitasking is called serial multitasking or context switching. This is nothing more than stopping one temporarily to work on another. If you have used sidekick, then you have used serial multitasking. While a program is running, you decide that you want to use the calculator, so you pop it and use it. When you stop using the calculator, the program continues running.

Multiuser operating system allow simultaneous access to a computer system through two or more terminals. Although frequently associated with multiprogramming, multiuser operating system does not imply multiprogramming or multitasking. A dedicated transaction processing system such as railway reservation system that supports hundreds of terminals under control of a single program is an example of multiuser operating system. On the other hand, general purpose time sharing systems (discussed later in this section) incorporate features of both multiuser and multiprogramming operating system. Multiprocess operation without multiuser support can be found in the operating system of some advanced personnel computers and in real systems (discussed later).

Time Sharing System: It is a form of multiprogrammed operating system which operates in an interactive mode with a quick response time. The user types a request to the computer through a keyboard. The computer processes it and a response (if any) is displayed on the user's terminal. A time sharing system allows the many users to simultaneously share the computer resources. Since each action or command in a time-shared system take a very small fraction of time, only a little CPU time is needed for each user. As the CPU switches rapidly from one user to another user, each user is given impression that he has his own computer, while it is actually one computer shared among many users.

Most time sharing system use time-slice (round robin) scheduling of CPU. In this approach, programs are executed with rotating priority that increases during waiting and drops after the service is granted. In order to prevent a program from monopolizing the processor, a program executing longer than the system defined time-slice is interrupted by the operating system and placed at the end of the queue of waiting program.

Memory management in time sharing system provides for the protection and separation of user programs. Input/output management feature of time-sharing system must be able to handle multiple users (terminals). However, the processing of terminals interrupts are not time critical due to the relative slow speed of terminals and users. As required by most multiuser environment allocation and deallocation of devices must be performed in a manner that preserves system integrity and provides for good performance.

The words multiprogramming, multiprocessing and multitasking are often confused. There are, of course, some distinctions between these similar, but distinct terms.

The term multiprogramming refers to the situation in which a single CPU divides its time between more than one job. Time sharing is a special case of

multiprogramming, where a single CPU serves a number of users at interactive terminals.

In multiprocessing, multiple CPUs perform more than one job at one time. Multiprogramming and multiprocessing are not mutually exclusive. Some mainframes and supermini computers have multiple CPUs each of which can juggle several jobs.

The term multitasking is described as any system that runs or appears to run more than one application program one time. An effective multitasking environment must provide many services both to the user and to the application program it runs. The most important of these are resource management which divides the computers time, memory and peripheral devices among competing tasks and interprocess communication, which lets tasking coordinate their activities by exchanging information.

Real-time Systems: It is another form of operating system which are used in environment where a large number of events mostly external to computer systems, must be accepted and processed in a short time or within certain deadlines. Examples of such applications are flight control, real time simulations etc. Real time systems are also frequently used in military application.

A primary objective of real-time system is to provide quick response times. User convenience and resource utilization are of secondary concern to real-time system. In the real-time system each process is assigned a certain level of priority according to the relative importance of the event it processes. The processor is normally allocated to the highest priority process among those which are ready to execute. Higher priority process usually pre-empte execution of lower priority processes. This form of scheduling called, priority based pre-emptive scheduling, is used by a majority of real-time systems.

Memory Management: In real-time operating system there is a little swapping of program between primary and secondary memory. Most of the time, processes remain in primary memory in order to provide quick response, therefore, memory management in real-time system is less demanding compared to other types of multiprogramming system. On the other hand, processes in real-time system tend to cooperate closely thus providing feature for both protection and sharing of memory.

I/O Management: Time-critical device management is one of the main characteristics of real-time systems. It also provides sophisticated form of interrupt management and I/O buffering.

File Management: The primary objective of file management in real-time systems is usually the speed of access rather than efficient utilization of secondary storage. In fact, some embedded real-time systems does not have secondary memory. However, where provided file management of real-time system must satisfy the same requirement as those found in time sharing and other multiprogramming systems.

5.4.3 Network Operating System

A network operating system is a collection of software and associated protocols that allows a set of autonomous computers which are interconnected by a computer network to be used together in a convenient and cost-effective manner. In a network operating system, the users are aware of existence of multiple computers and can log in to remote machines and copy files from one machine to another machine.

Some of typical characteristics of network operating systems which make it different from distributed operating system (discussed in the next section) are the following:

- Each computer has its own private operating system instead of running part of a global system wide operating system.
- Each user normally works on his/her own system; using a different system requires some kind of remote login, instead of having the operating system dynamically allocate processes to CPUs.
- Users are typically aware of where each of their files are kept and must move file from one system to another with explicit file transfer commands instead of having file placement managed by the operating system.

The system has little or no fault tolerance; if 5% of the personnel computers crash, only 5% of the users are out of business.

Network operating system offers many capabilities including:

— Allowing users to access the various resources of the network hosts

— Controlling access so that only users in the proper authorization are allowed to access particular resources.

— Making the use of remote resources appear to be identical to the use of local resources

— Providing up-to-the minute network documentation on-line.

As we said earlier, the key issue that distinguishes a network operating system from a distributed one is how aware the users are of the fact that multiple machines are being used. This visibility occurs in three primary area; file system, protection and program execution.

File System: The important issue in file system is related to how a file is placed (accessed) on one system from another in a network. There are two important approaches to this problem.

1) Running a special file transfer program

2) specifying a path name

Running a special file transfer program: When connecting two or more systems together, the first issue that must be faced is how to access the file system available on some other system. To deal with this issue user runs a special file transfer program that copies the needed remote file to the local machine, where they can then be accessed normally. Sometimes remote printing and mail is also handled this way. One of the best known examples of network that primarily support file transfer and mail via special programs is the UNIX's uucp (user to user control program) program and its network USENET.

Path name specification: The second approach in this direction is that programs from one machine can open files on another machine by providing a path name telling were the file is located. For example, one could say.

Open ("system 1/pathname",READ);

Or

Open ("/../Machine 1/pathname",READ);

"/.." means start at the local root directory and go upward one level (to the subdirectory) and then down to the root directory of system. In fig. 6 the root directory of three systems S_1, S_2 and S_3 are shown with a subdirectory above them. To access a file from machine S_3, one could say

Open ("/../S3/X",READ)

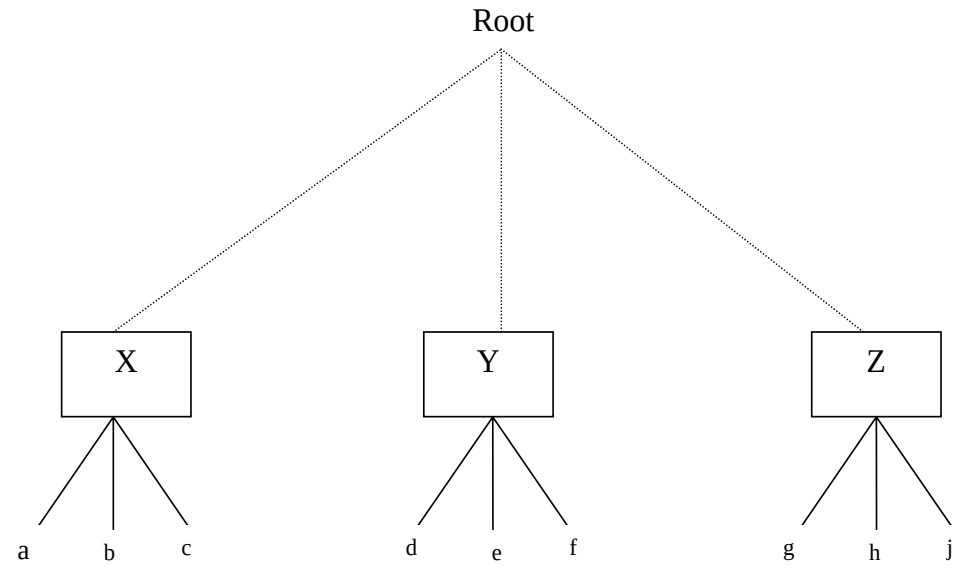


Fig. 6: A (virtual) Subdirectory above the Root Directory Provides Access to Remote Files

Protection:

Closely related to the file system is the issue of protection. UNIX and many other operating systems assign a unique internal identifier to each user. Each file in the file system has a little table associated with it (called an i-node in UNIX) telling who the owner is, where the disk blocks are located, etc. If two previously independent machines are now connected, it may turn out that same internal User Identifier (UID), for example, number 12, has been assigned to a different user on each machine. Consequently, when user 12 tries to access a remote file, the remote file system cannot see whether the access is permitted since two different users have the same UID.

One solution to this problem is to require all remote users wanting to access file on machine X to first log on to X using a user name that is, local to X. When used this way, the network just being used as a fancy to allow users at any terminal to log on to any computer, just as a telephone company switching centre allows any subscriber to call any other subscriber.

This solution is usually inconvenient for people and impractical for programs, so something better is needed. The next step up is to allow any user to access files on any machine without having to log in, but the UID corresponding to "GUEST" or "DEMO" or some other publicly known login name. Generally such names have little authority and can only access file that have been designed as readable or writable by all users.

A better approach is to have the operating system provide a mapping between UIDs, so that when a user with UID 12, on his or her home machine accesses a remote machine on which his or her UID is 15, the remote machine treats all

accesses as though they were done by user 15. This approach implies that sufficient tables are provided to map each user from his or her home (machine, UID) pair to the appropriate UID for any other machine (and that messages cannot be tampered with).

Execution Location

Program execution is the third area in which machine boundaries are visible in network operating systems. When a user or a running program wants to create a new process, where is the process created? At least four schemes have been used thus far. The first of these is that the user simply says “CREATE PROCESS” in one way or another, and specifies nothing about where. Depending on the implementation, this can be the best or worse way to do it.

The second approach to process location is to allow users to run job on any machine by first logging in there. In this model, processes on different machines cannot communicate or exchange data, but a simple manual load balancing is possible.

The third approach is a special command that the user types at a terminal to cause a program to be executed on a specific machine. A typical command might be :

Remote vax4 who

To run the **who program on machine vax4**. In this arrangement, the environment of the new process is the remote machine. In other words, if that process tries to read or write files from its current working directory, it will discover that its working directory is on the remote machine, and that files that were in the parent process’s directory are no longer present. Similarly, files written in the working directory will appear on the remote machine, not the local one.

The fourth approach is to provide the “CREATE PROCESS” system call with a parameter specifying where to run the new process, possibly with a new system call for specifying the default site. As with the previous method, the environment will generally be the remote machine. In many cases, signals and other forms of interprocess communication between processes do not work properly among processes on different machines.

Now let us see how file system protection and program execution are supported in distributed operating systems.

5.4.4 Distributed Operating System

A distributed operating system is one that looks to its users like an ordinary centralized operating system but runs on multiple independent CPUs. The key concept here is transparency. In other words, the use of multiple processors should be invisible to the user. Another way of expressing the same idea is to say that user views the system as virtual uniprocessor but not as a collection of distinct machines. In a true distributed system, users are not aware of where their programs are being run or where their files are residing; they should all be handled automatically and efficiently by the operating system.

Distributed operating systems have many aspects in common with centralized ones but they also differ in certain ways. Distributed operating system, for example, often allow programs to run on several processors at the same time, thus requiring more complex processor scheduling (scheduling refers to a set of policies and mechanisms built into the operating systems that controls the order

in which the work to be done is completed) algorithms in order to achieve maximum utilisatin of CPU's time.

Fault-tolerance is another area in which distributed operating systems are different. Distributed systems are considered to be more reliable than uniprocessor based system. They perform even if certain part of the hardware is malfunctioning. This additional feature supported by distributed operating system has enormous implications for the operating system.

Advantages of Distributed Operating Systems

There are three important advantages in the design of distributed operating system:

1) **Major break thorough in microprocessor technology:** Micro-processors have become very much powerful and cheap, compared with mainframes and minicomputers, so it has become attractive to think about designing large systems consisting of small processors. These distributed systems clearly have a price/performance advantages over more traditional systems.

2) **Incremental Growth:** The second advantage is that if there is a need of 10 per cent more computing power, one should just add 10 per cent more processors. System architecture is crucial to the type of system growth, however, since it is hard to give each user of a personal computer another 10 per cent.

3) **Reliability:** Reliability and availability can also be a big advantage; a few parts of the system can be down without disturbing people using the other parts; On the minus side, unless one is very careful, it is easy for the communication protocol overhead to become a major source of inefficiency.

Now let us see how file system, protection and program execution are supported in distributed operating system.

File System: Distributed operating system supports a single global file system visible from all machines. When this method is used, there is one directory for executable programs (in UNIX, it is bin directory), one password file and so on. When a program wants to read the password file it does something like

Open (*/etc/password", READ-ONLY)

Without reference to where the file is. It is upto the operating system to locate the file and arrange for transport of data as they are needed.

The convenience of having a single global name space is obvious. In addition, this approach means that operating system is free to move files around among machines to keep all the disks generally full and busy and that the system can maintain replicated copies of files if it chooses. When the user or program must specify the machine name, the system cannot decide on its own to move a file to a new machine because that would change the (user visible) name used to access the file. Thus in a network operating system, control over file placement must be done manually by the users, whereas in a distributed operating system it can be done automatically by the system itself.

Protection: In a true distributed system there is a unique UID for every user, and that UID should be valid on all machines without any mapping. In this way no protection problems arise on remote access to files; a remote access can be treated like a local access with the same UID. There is a difference between network operating system and distributed operating system in implementing

protection issue. In networking operating system, there are various machines, each with its own user to UID mapping but in distributed operating system there is a single systemwide mapping that is valid everywhere.

Program Execution: In the most distributed case the system chooses a CPU by looking at the processing load of the machine, location of file to be used etc. In the least distributed case, the system always run the process on one specific machine (usually the machine on which the user is logged in).

An important difference between network and distributed operating system is how they are implemented. A common way to realize a networking operating system is to put a layer of software on top of the native operating system of the individual machines. For example one could write a special library package that could intercept all the system calls and decide whether each one was local or remote. Although most system calls can be handled this way without modifying kernel (kernel is that part of operating system that manages all resources of computer).

Self Check Exercise

4) What are the main difficulties in writing an operating system for a real time environment?

Note: i) Write your answer in the space given below.

ii) Check your answer with the answers given at the end of this Unit.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

5.5 OPERATING SYSTEM STRUCTURE

Since operating system is a very large and complex software, supports large number of functions. It should be developed as a collection of several smaller modules with carefully defined inputs, outputs and functions rather than a single piece of software. In this section, we will examine different operating system structure.

5.5.1 Layered Structure Approach

The operating system architecture based on layered approach consists of number of layers (levels), each built on top of lower layers. The bottom layer is the hardware; the highest layer is the user interface. The first system constructed in this way was the THE system built by E.W. Dijkestra (1968) and his students. The THE system was a simple batch operating system which had 32k of 27 bit words.

The system supported 6 layers (Fig. 7).

5	User Programs
4	Buffering for I/O devices
3	Device Driver
2	CPU scheduling
1	Hardware
0	Hardware

Fig. 7: The Layered Structure of THE Operating System

As shown in fig. 7, layer 0 dealt with hardware; the higher layer layer1 handled allocation of jobs to processor. The next layer implemented memory management. The memory management scheme was virtual memory. Level 3 contained the device driver for the operator's console. By placing it, as well as I/O buffering at level 4, above memory management, the device buffers could be placed in virtual memory. The I/O buffering was also above the operator's console, so that I/O error conditions could be output to the operator's console.

The main advantage of the layered approach is modularity which helps in debugging and verification of the system easily. The layers are designed in such a way that it uses operation and services only if a layer below it. A higher layer need not know how these operations are implemented, only what these operation do. Hence each layer hides implementation details from higher level layers. Any layer can be debugged without any concern about the rest of the layer.

The major difficulty with the layered approach is definition of a new level i.e. how to differentiate one level from another. Since a layer can use services of a layer below it, it should be designed carefully. For example, the device driver for secondary memory must be at a lower level than the memory management routines since memory management requires the ability to use the backing store.

5.5.2 Kernel Approach

Kernel is that part of operating system which directly makes interface with hardware system. Its main functions are:

- To provide a mechanism for creation and deletion of processes
- To provide processor scheduling, memory management and I/O management.
- To provide mechanism for synchronization of processes so that processes synchronize their actions.
- To provide mechanism for interprocess communication.

The UNIX operating system is based on kernel approach (fig. 8). It consists of two separatable parts: (i) Kernel (ii) System Programs

As shown in the fig. 8, kernel is between system programs and hardware. The kernel supports the file system, processor scheduling, memory management and other operating system functions through system calls. UNIX operating system supports a large number of system calls for process management and other

operating system functions. Through these system calls program utilizes the services of operating system (kernel).

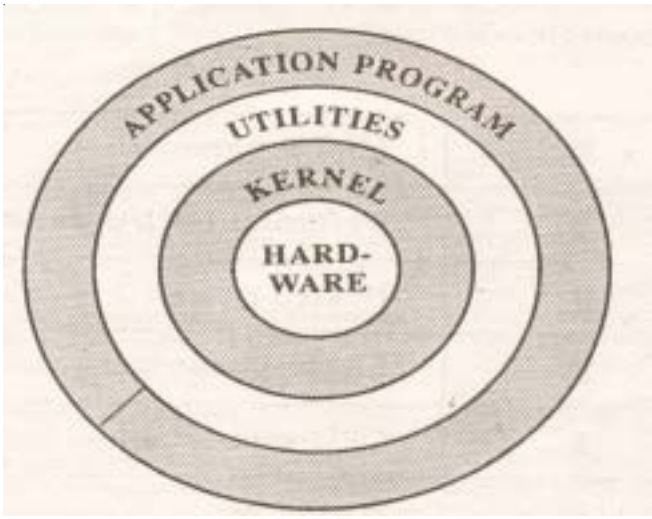


Fig. 8: UNIX Operating System Structure

5.5.3 Virtual Machine

It is a concept which creates an illusion of a real machine. It is created by a virtual machine operating system that makes a single real machine appear to be several real machines. This type of situation is analogous to communication line of telephone company which enables separate and isolated conversations over the same wire(s).

The following figure illustrates this concept.

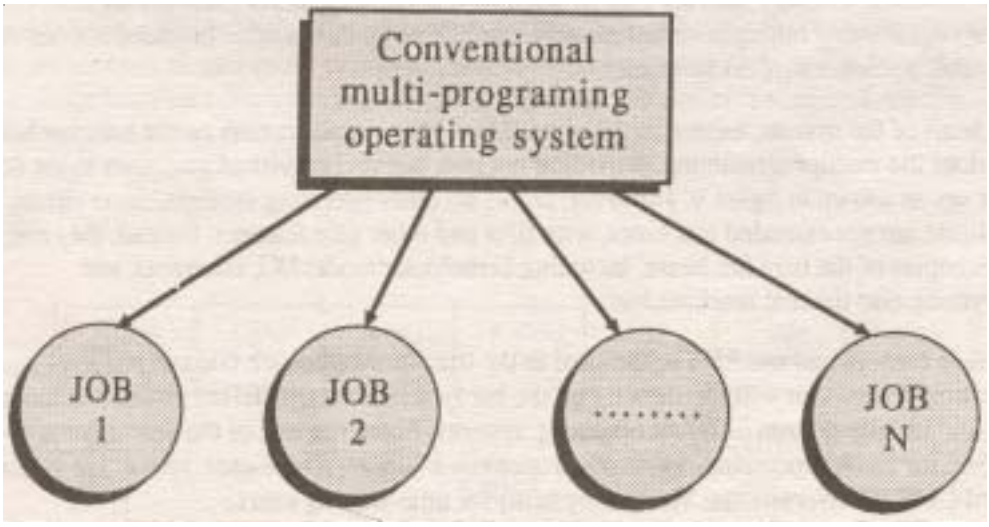


Fig. 9: Creation of Several Virtual Machines by a Single Physical Machine

From the user's view point, virtual machine can be made to appear to be very similar to existing real machine or they can be entirely different. An important aspect of this technique is that each user can run operating system of his own choice. This fact is depicted by OS_1 (Operating System 1), OS_2, OS_3 etc. in fig. 9.

To understand this concept, let us try to understand the difference between conventional multiprogramming system (fig. 10) and virtual machine multi-programming (fig. 11). In conventional multiprogramming processes are allocated a portion of the real machines resources. The same machine resources are distributed among several processes.

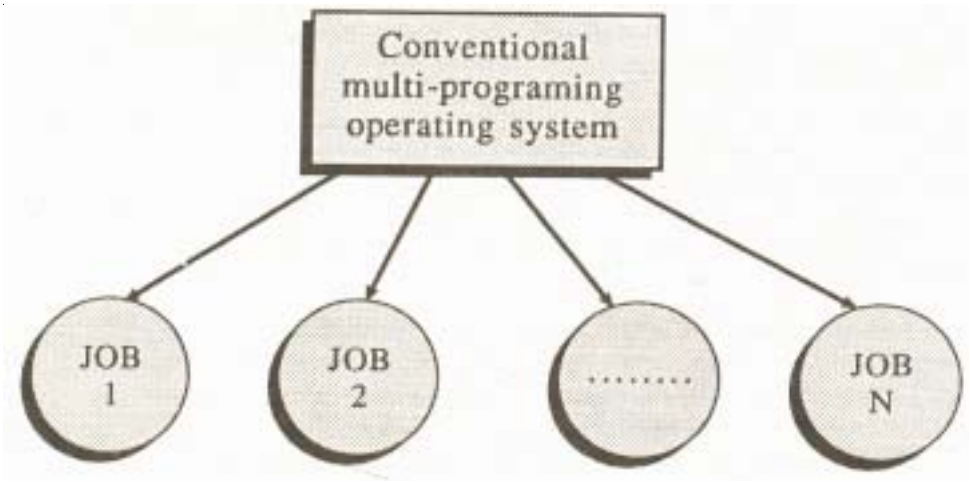


Fig. 10: Conventional Multiprogramming

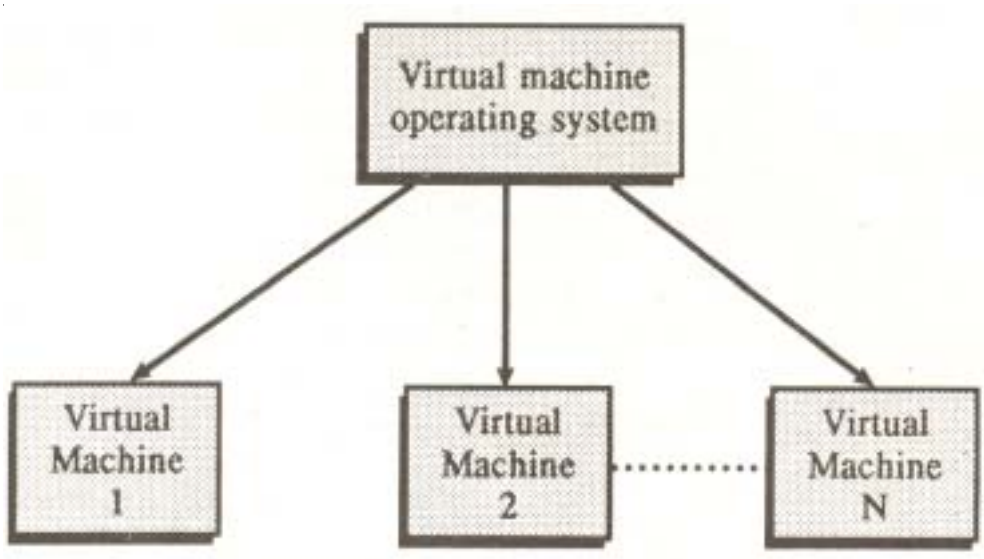


Fig. 10: Virtual Machine Multiprogramming

In virtual multiprogramming system, a single real machine gives an illusion of several virtual machines, each having its own virtual processor, storage and I/O devices possibly with much larger capacities. This is possible through process scheduling and virtual memory organization technique. Process scheduling can be used to share the CPU and make it appear that users have their own processor. Virtual memory organization technique can create illusion of very large memory for program execution.

The virtual machine has many uses and advantages:

1) Concurrent running of dissimilar operating system by different users.

2) **Elimination of certain** conversion problem.

3) **Software development** – programs can be developed and debugged for machine configuration that is different from those of host (for example

virtual operating system VM/370 can produce virtual 370 that are different from the real 370 such as larger main memory).

4) **Security and Privacy:** The high degree of separation between independent virtual machines aids in ensuring privacy and security. The most widely used operating system in this category is VM/370. It manages IBM/370 computer and creates illusion that each of several users has a complete system 370 (including wide range of I/O devices) which can run different operating systems at once, each of them on its own virtual machine.

It is also possible through software to share files existing on physical disk and several information through virtual communication software.

The virtual machines are created by sharing the resources of physical computer. CPU scheduling can be used to share the CPU and make it appear that users have their virtual machines, any software desired. The virtual machine software is concerned with multiprogramming multiple virtual machines onto a physical machine but need not consider any other software support from user.

The heart of the system, known as the virtual machine monitor, runs on the bare hardware and does the multiprogramming, providing not one, but several virtual machines to the next layer up, as shown in fig. 9. However, unlike all other operating systems, these virtual machines are not extended machines, with files and other nice features. Instead, they are exact copies of the bare hardware, including kernel/user mode, I/O, interrupts, and everything else the real machine has.

Because each virtual machine is identical to the true hardware, each one can run any operating system that will run directly on the hardware. In fact, different virtual machines can, and usually do, run different operating systems. Some run one of the descendants of OS/360 for batch processing, while other ones run a simple, single-user, interactive system called CMS (Conversational Monitor System) for time-sharing users.

When a CMS program executes a system call, the call is trapped to the operating system in its own virtual machine, not to VM/370, just as it would if it running on a real machine instead of virtual one. CMS then issues the normal hardware I/O instructions for reading its virtual disk or whatever is needed to carry out the call. These I/O instructions are trapped by VM/370, which then performs them as part of its simulation of the real hardware. By making a complete separation of the functions of multiprogramming and providing an extended machine, each of the pieces can be much simpler and more flexible. The virtual machine concept has several advantages. Notice that there is complete protection. Each machine is completely isolated from all other virtual machines, so there is not problem with protection. On the other hand, there is no sharing. To provide sharing, two approaches have been implemented. First, it is possible to share a minidisk. This scheme is modeled after a physical shared disk, but implemented by software. With this technique, files can be shared. Second, it is possible to define a network of virtual machines, each of which can send information over the virtual communications network. Again, the network is modelled after physical communication networks, but implemented in software.

Such a virtual machine system is a perfect vehicle for operating systems research and development. Normally changing on operating system is a difficult process, since operating systems are large and complex programs, it is difficult to be sure that a change in one point does not cause obscure bugs in some other part. This situation can be particularly dangerous because of the power of the operating

system. Since the operating system executes in monitor mode, a wrong change in a pointer could cause an error that would destroy the entire file system. Thus it is necessary to test all changes to the operating system carefully.

But the operating system runs or and controls the entire machine. Therefore, the current system must be stopped and taken out of use, while changes are made and tested. This is commonly called system development time. Since it makes the system unavailable to users, system development time is often schedule late at night or on weekends.

A virtual machine system can eliminate much of this problem. System programmers are given their own virtual machine and system development is done on the virtual machine, instead of on a physical machine. Normal system operation seldom need be disrupted for system development.

5.5.4 Client-Server Model

VM/370 gains much in simplicity by moving a large part of the traditional operating system code (implementing the extended machine) into a higher layer itself. VM/370 it is still a complex program because simulating a number of virtual 370s is not that simple (especially if you want to do it efficiently).

A trend in modern operating systems is to take this idea of moving code up into higher layers even further, and remove as much as possible from the operating system, leaving a minimal kernel. The usual approach is to implement most of the operating system functions in user processes. To request a service, such as reading a block of a file, a user process (now known as the client process) sends the request to a server process, which then does the work and sends back the answer.

In this model, shown in fig. 12, all the kernel does is to handle the communication between clients and servers. By splitting the operating system up into parts, each of which only handles one facet of the system, such as file service, process service, terminal service or memory service. This way, each part becomes small and manageable. Furthermore, because all the servers run as user-mode processes, and not in kernel mode, they do not have direct access to the hardware. As a consequence, if a bug in the file server is triggered, the file service may crash, but this will not usually bring the whole machine down.

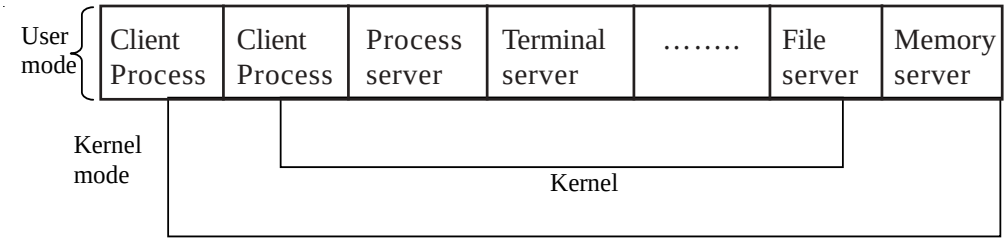


Fig. 12: The Client-Server Model

Another advantage of the client-server model is its adaptability to use in distributed systems (fig. 13). If a client communicates with a server by sending it messages, the client need not know whether the message is handled locally in its own machine, or whether it was sent across a network to a server on a remote machine. As far as the client is concerned, the same thing happens in both cases: a request was sent and a reply came back.

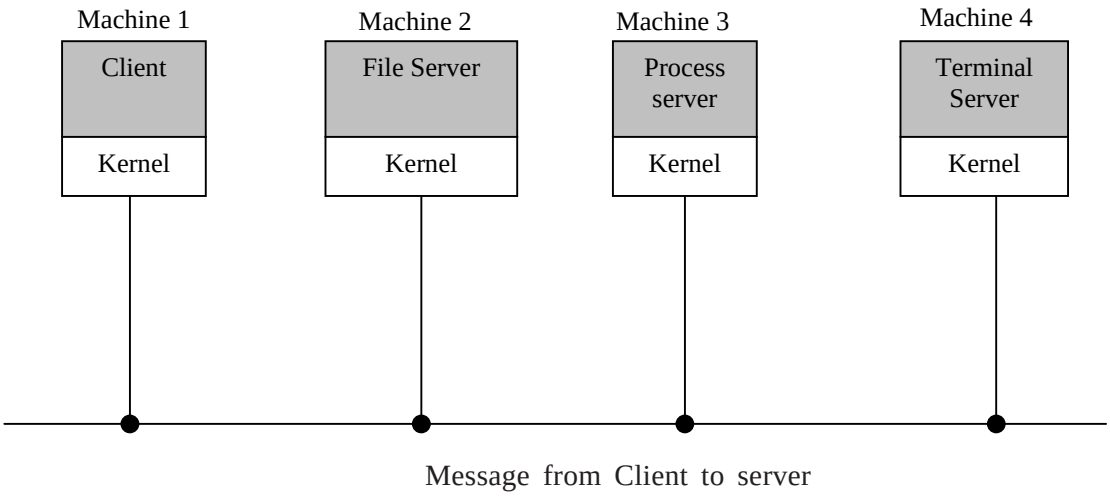


Fig. 13: The Client-Server Model in a Distributed System

The picture painted above of a kernel that handles only the transport of messages from clients to servers and back is not completely realistic. Some operating system functions (such as loading commands into the physical I/O device registers) are difficult, if not impossible, to do from user-space programs. There are two ways of dealing with this problem. One way is to have some critical server processes (e.g. I/O device drivers) actually run in kernel mode, with complete access to all the hardware, but still communicate with other processes using the normal message mechanism.

The other way is to build a minimal amount of mechanism into the kernel, but leave the policy decisions up to servers in user space. For example, the kernel might recognize that a message sent to a certain special address means to take the contents of that message and load it into the I/O device registers for some disk, to start a disk read. In this example, the kernel would not even inspect the bytes in the message to see if they were valid or meaningful; it would just blindly copy them into the disk's device registers. (Obviously some scheme for limiting such messages to authorized processes only must be used). The split between mechanism and policy is an important concept; it occurs again and again in operating system in various contexts.

Self Check Exercise

5) How is client-server model different from the other two operating system structures? List various advantages of client-server models.

Note: i) Write your answer in the space given below.

ii) Check your answer with the answers given at the end of this Unit.

.....

.....

.....

.....

.....

.....

5.6 FUTURE OPERATING SYSTEM TRENDS

A number of clear trends are emerging which will point the way the future

operating system will be designed. These are:

- Multiprocessing feature will be much more common because of development of VLSI chips and the decline in the cost of hardware.
- Microcode will support all the functions of operating system integrated into it, currently performed by software, therefore, the execution time will be faster.
- The trend is towards distributed control among the localized processors in place of centralized system. Therefore the use of distributed operating system will increase.
- The concurrency is becoming an important feature of programming language. Hardware and operating systems are being designed to execute concurrent programs more efficiently.
- Developments in software engineering will result in operating systems that will be easily maintainable, more reliable and simpler.
- Lots of developments are taking place in computer networking. The data transmission rate is also increasing. Therefore, the use of networking operating system will also increase.
- Virtually all operating systems (specially desktop based operating systems) will be supporting multimedia (video and graphics) applications in the near future.

5.7 SUMMARY

Operating system is an essential component of system software which consists of procedures for managing computer resources. Initially computer were operated from the front console. System software such as Assemblers, Loaders and Compilers greatly improved in software development but also required substantial setup time. To reduce the setup time an operator was hired and similar jobs were batched together.

Batch systems allowed automatic job sequenting by a resident monitor and improved the overall utilization of systems greatly. The computer no longer had to wait for human operations – But CPU utilization was still low because of slow speed of I/O devices compared to the CPU. A new concept buffering was developed to improve system performance by overlapping the input, output and computation of a single job. Spooling was another new concept in improving the CPU utilization by overlapping input of one job with the computation and output of other jobs.

Operating systems are now almost always written in a higher level languages (C, PASCAL etc.). UNIX was the first operating system developed in C language. This feature improves their implementation, maintenance and portability.

Operating system provides a number of services. At the lowest level, there are system calls which allow a running program to make a request from operating system directly. At a higher level, there is a command interpreter which supports a mechanism for a user to issue a request without writing a program.

In this unit, we began with tracing the evolution of operating system through serial processing, batch processing and multiprogramming. On the basis of these

characteristics and these objectives, different types of operating systems were defined and characterized with respect to processor scheduling, memory management, device management and file management.

We also presented different operating system model: Layered structured, Kernel based, virtual machine system and client server model.

At the end we presented a list of clear trends which will dominate the future operating system design.

5.8 ANSWERS TO SELF CHECK EXERCISES

1) System calls provide the interface to a running program and the operating system. User program receives operating system services through the set of system calls.

2) Buffering is a method of overlapping input, output and processing of a single job. More sophisticated from of input/output buffering called SPOOLING (simultaneous peripheral operation on lines) essentially use the disk as a very large buffer for reading and for starting output files. Buffering overlaps input, output and processing of a single job whereas spooling allows CPU to overlap the input of one job with the computation and output of other jobs.

3) Advantage of multiprogramming in that it offers a more efficient approach to increase system performance. Systems supporting multiprogramming approach allow more than one job (program) to utilize CPU time at any moment. More number of programs competing for system resources will result in better resource utilization.

4) In real-time operating system there is a little swapping of program between primary and secondary memory. Most of the time, processes remain in primary memory in order to provide quick response, therefore, memory management in real-time system is less demanding compared to other types of multiprogramming system.

5) In the client server model the operating system is split into parts, each of which only handles one facet of the system, such as file service, process service, terminal service or memory service. This way each part becomes small and manageable.

Moreover, since all the servers such as user-mode processes, and not in kernel mode, they do not have direct access to the hardware. As a result, it a bug in the file.

5.9 KEYWORDS

Assembler : A type of program that translate a programming language into simple instructions that are understood directly by the computer.

Compiler : A program that translates a high-level computer programming language into machine language.

Load : To call up a program or data to the computer's main memory from a storage device.

5.10 REFERENCES AND FURTHER READING

Deital, Harvey M. Introduction to Operating System. Warsaw: Addision Wesely Publishing.

Madwick, Stuart E. and Donovan, John J. Operating System : Concept and Design. New York : McGraw Hill Ints. Edition.

Silberschatz, Abraham and Peterson, James L. Operating System Concepts. Warsaw: Addision Wisely Publishing.

Tanenbaum, Andrew S. and Woodhull, Albert S. (2000). Operating Systems Design and Implementation : New Delhi: Prentice Hall of India.