4.0 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.

In the previous unit, we mainly introduced different types of software: and categories of programming languages. All these 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 and types of operating systems will also be broadly covered here.

4.1 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
- list future trends in operating system
- trace the history of operating systems.

4.2 WHAT IS AN OPERATING SYSTEM?

An operating system is an essential component of a computer system. The primary objective 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.

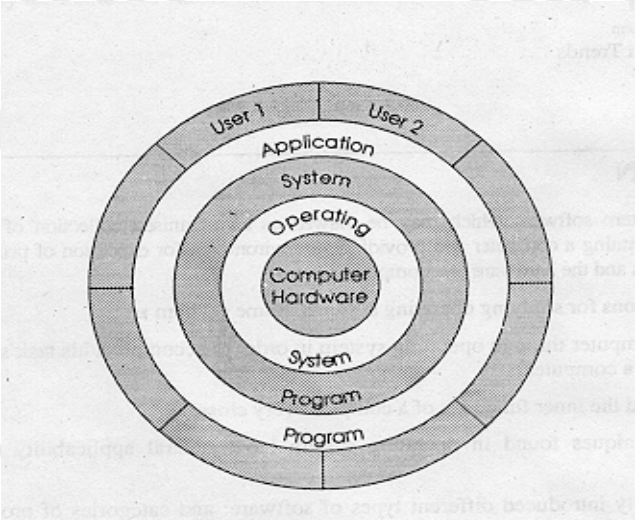


Figure 1: Component of computer system

From the figure, 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. here 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 now a days 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 function 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 that 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 out of paper and so on).
- Close both files after the entire file is copied.

As we can observe, a user program takes 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.

4.3 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 an interleaved fashion or in 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**.

4.3.1 Serial Processing

Programming in 1s and 0s (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 counter register (a register which keeps track of which instruction of a program is being executed) 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 tapes and language translators (Compiler/Assemblers) brought a significant step in computer system utilisation. Program, being coded into programming language, is 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 utilisation of resources.

4.3.2 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 he runs them in that order, he would have to set up for FORTRAN program environment (loading the FORTRAN compiler tapes) then set up COBOL program and

finally FORTRAN program again. If he runs the two FORTRAN programs 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 information 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

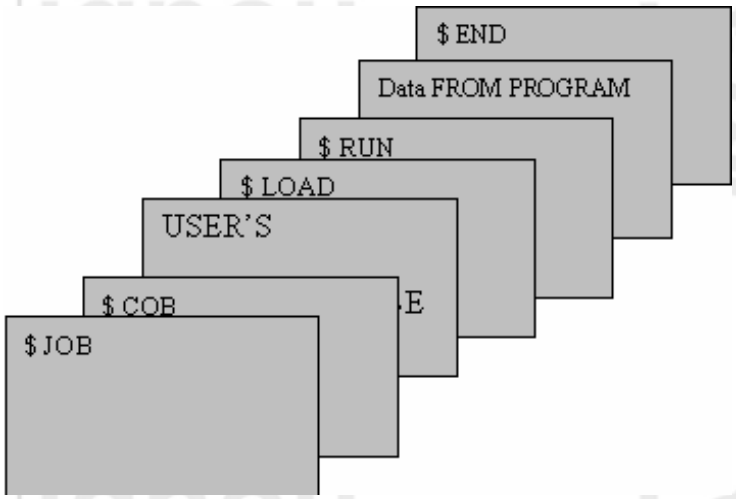


Figure 2: Card deck for Cobol Program for a simple batch system

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 utilisation 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 utilisation 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 without intervention by CPU was a major development. While CPU is executing, DMA can transfer data between high-speed input/output devices and 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 input/output device controls the speed of execution, 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 (figure 3) for reading and for storing output files.

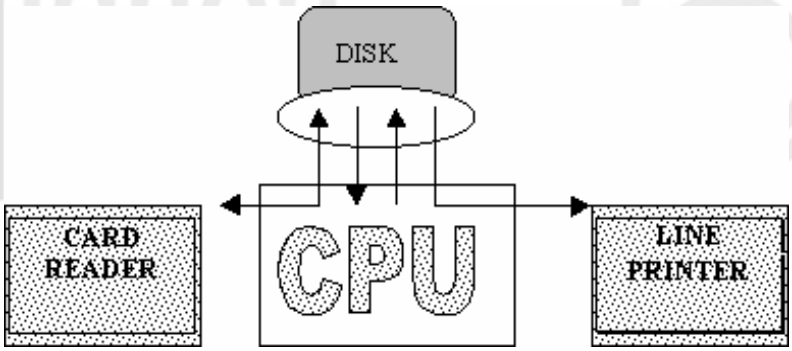


Figure 3: Spooling

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.

4.3.3 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 utilisation, systems supporting multiprogramming approach allow more than one job (program) to utilise CPU time at any moment. More number of programs competing for system resources, better will be resource utilisation.

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

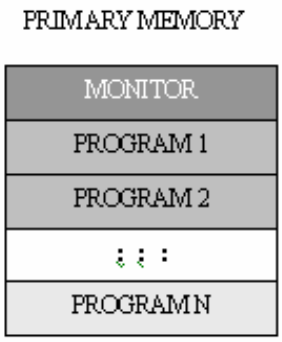
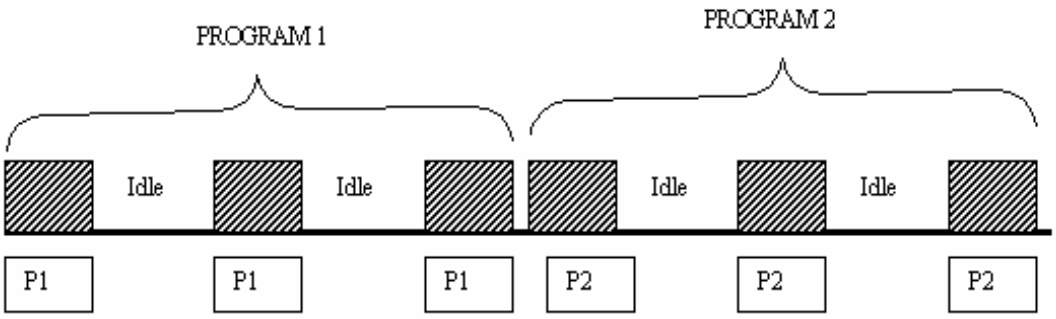
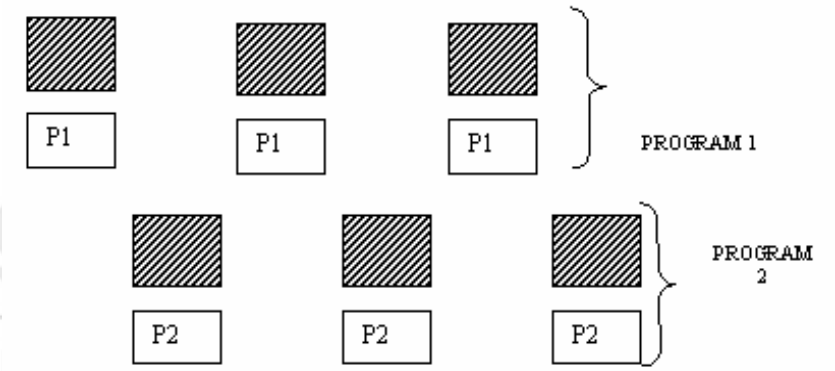


Figure 4: Memory layout in multiprogramming environment

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 (Figure 5a), the CPU would sit idle. In a multiprogramming system, (Figure 5b) operating system will simply switch over to the next process (program 2).



(a) Execution in sequential programming environment



(b) Execution in multiprogramming environment

Figure 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 utilisation 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 is beyond the scope of this block.

Check Your Progress 1

1. What is a system call?

   ……

   ……

2. Define the essential difference between

   (a) Spooling

(b) Buffering

……

……

3. What are the main advantages of multiprogramming ?

……

……

4.4 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.

4.4.1 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 or 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 user. Batch operating systems allow little or no interaction between users and executing programs. The 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.

4.4.2 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 utilisation with very minor differences. Different forms of multiprogramming operating system are **multitasking, multiprocessor** and **multi-user 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 supports two or more active processes simultaneously. Multiprogramming operating system is operating system which in addition to supporting multiple concurrent process (several 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.

Multi-user operating system:

It allows simultaneous access to a computer system through two or more terminals. Although frequently associated with multiprogramming, multi-user 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 multi-user operating system. On the other hand, general-purpose time sharing systems (discussed later in this section) incorporate features of both multi-user and multiprogramming operating system. Multiprocessor operation without multi-user 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.

The term **multitasking** is described 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 co-ordinate their activities by exchanging information.

Real-time Systems:

It is another form of operating system which are used in environments 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 utilisation 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 processes. The processor is normally allocated to the highest priority process among those, which are ready to execute. Higher priority process usually pre-empt execution of lower priority processes. This form of scheduling called, priority based pre-emptive scheduling, is used by a majority of real-time systems.

4.4.3 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 into remote machines and copy files from one machine to another machine.

Some of typical characteristics of network operating systems which may be different from distributed operating system (discussed in the next section) are the followings:

- 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 authorisation 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.

4.4.4 Distributed Operating System

A distributed operating system is one that looks to its users like an ordinary centralised 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 centralised 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 utilisation 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 breakthrough 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.

Check Your Progress 2

1. Define the essential differences between the time-sharing and real time operating systems.

   ……

   ……

2. List the main differences between network operating systems and distributed operating systems.

.....

.....

4.5 SUMMARY

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

Batch systems allowed automatic job sequencing by a resident monitor and improved the overall utilisation of systems greatly. The computer no longer had to wait for human operations - but CPU utilisation 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 utilisation by overlapping input of one job with the computation and output of other jobs.

Operating systems are now almost always written in 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 different types of operating systems were defined and characterised.

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

4.6 MODEL ANSWERS

Check Your Progress 1

1. System call is a way of acting with the operating system of a computer

2. Buffering and spooling both tries to smooth out speed difference between the CPU and I/O devices, however, buffering does it only for a single job ongoing which SPOOLING allows input of one job with the computation and output of other job.

3. The main advantages of multiprogramming are:

   * better CPU throughput
   * better resource utilisation
   * can supports multiple users at a time.

Check Your Progress 2

1. The essential difference between time sharing and real, line systems are the way CPU time is utilised. In time sharing system jobs are given equal time slot, whereas in real time system a priority scheme exist and the jobs with high priority are given time first.

2.

Network Operating System	Distributed Operating System
♦ A single machine has its own O.S.	All the machines are controlled by single O.S.
♦ The files of a user are kept on known	Files of a user can be on any machine but user will

machine	get it as and when desired.
♦ Less efficient processing	More efficient processing.

4.7 FURTHER READINGS

Milan Milenkovic, Operating Systems Concepts and Design, Second Edition, Tata McGraw-Hill.