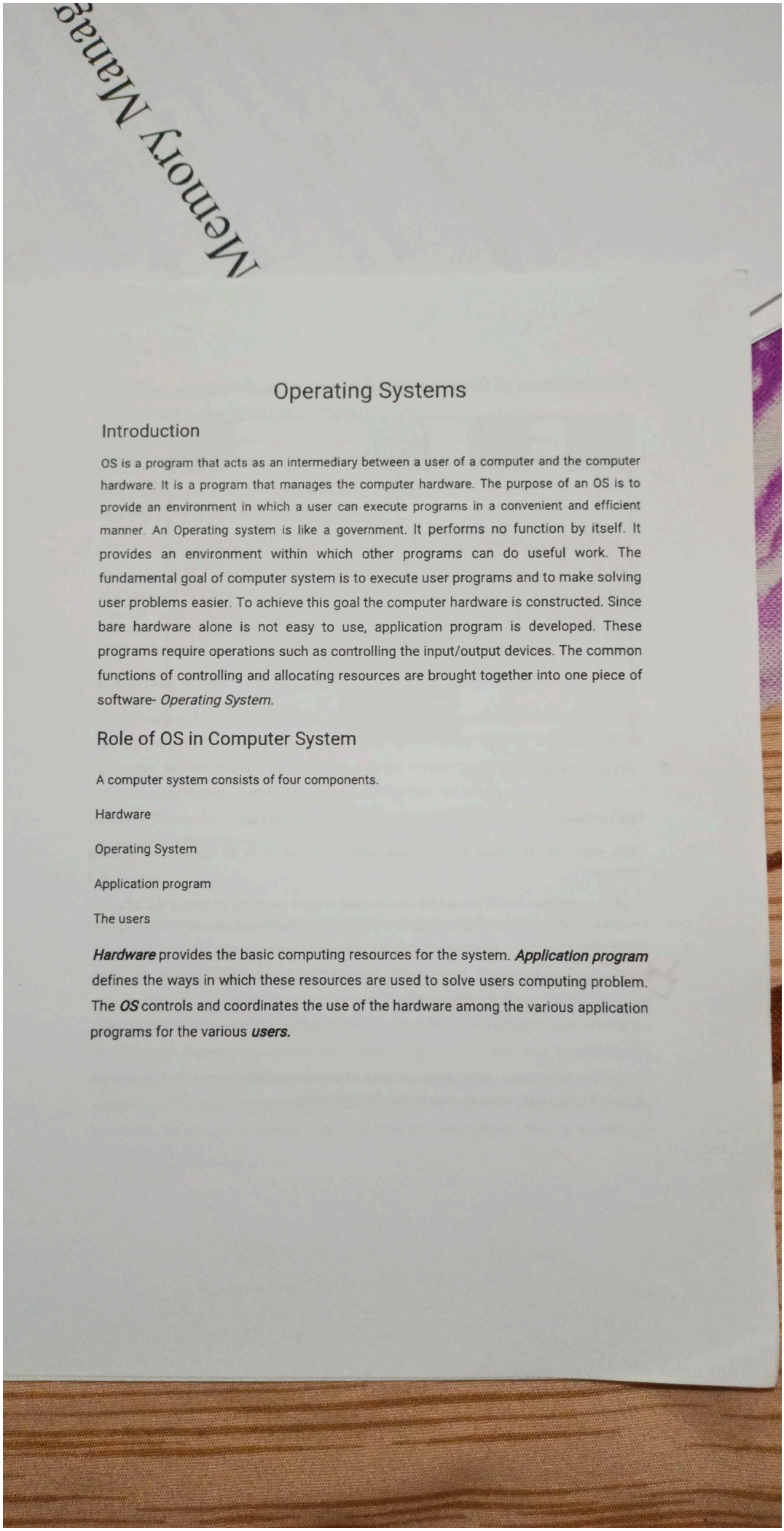


Operating Systems

Introduction

OS is a program that acts as an intermediary between a user of a computer and the computer hardware. It is a program that manages the computer hardware. The purpose of an OS is to provide an environment in which a user can execute programs in a convenient and efficient manner. An Operating system is like a government. It performs no function by itself. It provides an environment within which other programs can do useful work. The fundamental goal of computer system is to execute user programs and to make solving user problems easier. To achieve this goal the computer hardware is constructed. Since bare hardware alone is not easy to use, application program is developed. These programs require operations such as controlling the input/output devices. The common functions of controlling and allocating resources are brought together into one piece of software- *Operating System.*

Role of OS in Computer System

A computer system consists of four components.

Hardware

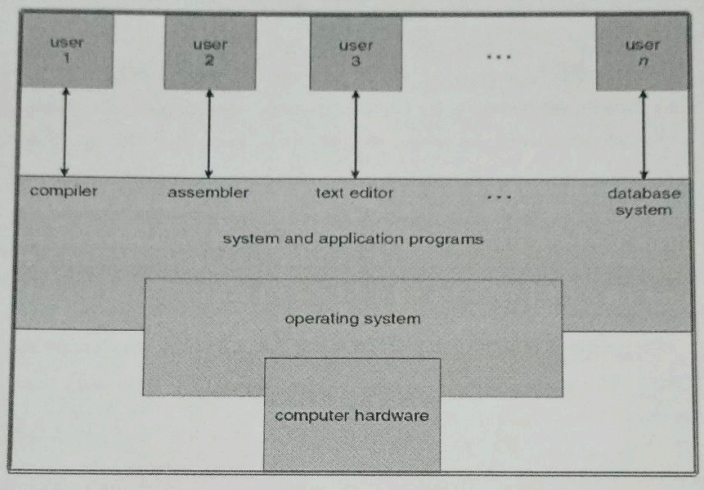
Operating System

Application program

The users

Hardware provides the basic computing resources for the system. ***Application program*** defines the ways in which these resources are used to solve users computing problem. The ***OS*** controls and coordinates the use of the hardware among the various application programs for the various ***users.***

Abstract View of Components of a computer system



Objectives of OS

Convenience - An OS makes a computer more convenient to use.

Efficiency- An OS allows the computer system resources to be used in an efficient manner.

Ability to evolve - An OS should be constructed in such a way as to permit the effective development, testing, and introduction of new system functions without interfering with service.

Functions of OS

Memory management -The o/s keeps track of the memory, which parts are in use and by whom.

Process management - The o/s keeps track of processors and the status of processes. It decides who will have a chance to use the processor

- Device management - The o/s keeps track of the devices, channels, control units and decides what is an efficient way to allocate the device.
- Information management - O/S keeps track of the information, its location, use, status etc. and decides who gets use of the resources, enforce protection requirements.
- Protection - An o/s is to protect the user from unauthorized access of his files or data. And also it should protect itself from users
- Error Handling - An o/s must respond to errors by taking the appropriate actions.

For a computer to start running, it needs to have an initial program to run. This initial program is called bootstrap program. It is stored in ROM or EEPROM. It is known by the general term firmware, within the computer hardware. It initializes all aspects of the system, from CPU registers to device controllers to memory contents. The bootstrap program must know how to load OS and to start executing that system.

Evolution of OS

To understand key requirements for an OS, we have to consider how OS have evolved. An OS may process its workload serially or concurrently. That is, resources of the computer system may be dedicated to a single program until its completion, or they may be dynamically reassigned among a collection of active programs in different stages of execution. In evolution of OS we are going to discuss Serial processing, Batch processing and Multiprogramming.

Serial Processing

With the earliest computers, the programmer interacted directly with the computer hardware. These computers were run from a console consisting of display lights, toggle switches, all are some form of input devices, and a printer. The user places a program and its input data on an input device and the loader transfers the information from that input device to memory. After transferring the control to the loaded program by manual or automatic means, execution of the program begins. If an error halted the program, the error condition was indicated by the lights. If the program proceeded to a normal completion, the output appeared on the printer/screen. Another program called loader, automates the process of loading executable programs into memory. Once in memory, the

program may be rerun with different set of input data.

Machine language programs were common in early systems. Inputs were given by console switches or hexadecimal keyboard. Programming caused low utilization of both user and machine. Advent of new input/output devices, language translators etc brought a significant step in system utilization.

These early systems presented two main problems:

Scheduling

Most installations used a hard copy sign-up sheet to reserve computer time. Typically, a user could sign up for a block of time in multiples of a half hour. A user might sign up for an hour and finish in 45 minutes. This would results in wasted computer processing time. On the other hand, the user might run into problems, not finished in the allotted time, and be forced to stop before resolving the problem.

Setup time

A single program, called a job, could involve loading the compiler plus the high level language program(source program) into memory. Saving the compiled program (object program) and then loading and linking together the object program and common functions. Each of these steps could involve mounting or dismounting the tapes or setting up card decks. If an error occurs, the user has to go back to the beginning of the set up sequence. Thus a considerable amount of time was spent just in setting up the program to run.

Simple Batch Systems(Batch Processing)

Since the early computers were very expensive, it was very important to maximize processor utilization. The wasted time due to scheduling and setup time was unacceptable. To improve utilization, the concept of batch OS was developed. The first batch OS was developed in the mid- 1950s by General Motors for use on IBM 701 and then IBM 704 etc. By 1960number of vendors had developed batch OS for their computer systems. IBSYS- the OS for IBM.

an I/O command for one job and proceed with the execution of other job while the I/O is carried out by the device controller. DMA 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. When the I/O operation is complete, the processor is interrupted and control is passed to an interrupt handling program in the OS. The OS will then pass control to another job.

Device by device utilization is explained in the following figure.

Utilization Histograms

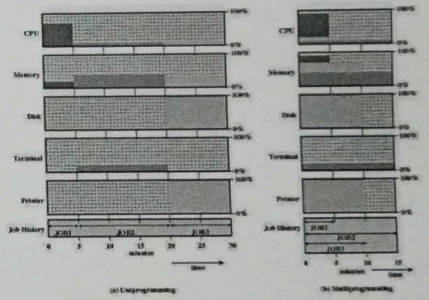


Figure 2.6 Utilization Histograms

Three jobs ... attribute listed ...

Attributes	Job
Type of Job	Heavy compute
Duration	5 min
Memory Required	50
Need	

Three jobs JOB1, JOB2 and JOB3 are submitted for execution at the same time with attribute listed in the following table.

Attributes	JOB1	JOB2	JOB3
Type of Job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory Required	50 M	100 M	75 m
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

For a simple batch environment, these jobs will be executed in sequence. Thus JOB1 completes in 5 minutes. JOB2 must wait until the 5 minutes are over and then completes 15 minutes after that. Job3 begins after 20 minutes and completes at 30 minutes from the time it was initially submitted.

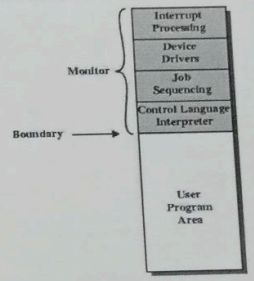
Suppose that the jobs are running concurrently under a multiprogramming OS. Since there is little resource contention between the jobs, all three can run in nearly minimum time while coexisting with the others in the computer. JOB1 will still require 5 minutes to complete, but at the end of that time JOB2 will be one-third finished and JOB3 half finished. All three jobs will have finished within 15 minutes.

As with simple batch systems, a multiprogramming batch system must rely on some hardware features. The important one is the hardware that support I/O interrupts and DMA(Direct Memory Access). With interrupt driven I/O or DMA, the processor can issue

The central idea behind batch processing scheme is the use of a piece of software known as the monitor. With this type of OS, the user has no direct access to the processor. The user submits the jobs on cards or tape to a computer operator, who batches the jobs and place the entire batch on input device, for use by the monitor. Each program is constructed to branch back to the monitor, when it completes processing, at which point the monitor automatically begins loading the next program. To understand this, let us see the two points of view: **monitor** and that of **processor**

Monitor point of view

Memory Layout for Resident Monitor



The monitor controls the sequence of events. For this to be like that, the monitor must always be in main memory and available for execution. That portion is referred to as the **resident monitor.** The rest of the monitor consists of utilities and functions that are loaded as subroutines to the user program at the beginning of any job that requires them. The monitor reads in jobs one at a time from the input device (typically a card reader or magnetic tape drive). As it is read in, the current job is placed in the user program area and control is passed to this job. When the job is completed, it returns control to the monitor, which immediately reads in the next job. The results of each job

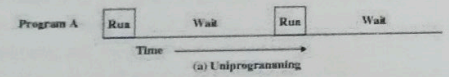
are sent to an output device, such as a printer, for delivery to the user.

Processor point of view

At certain point, the processor is executing instructions from the portion of main memory containing the monitor. These instructions cause the next job to be read into another portion of main memory. Once a job has been read in, the processor will encounter a branch instruction in the monitor that instructs the processor to continue execution at the start of the user program. The processor will then execute the instructions in the user program until it encounters an ending or error condition. Either event causes the processor to fetch its next instruction from the monitor program.

Multiprogrammed Batch Systems

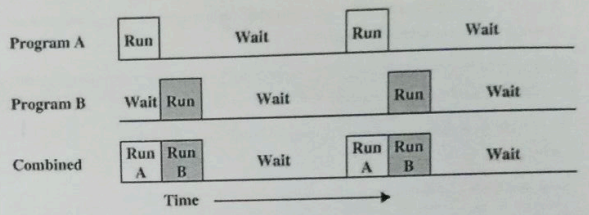
Even with the automatic job sequencing provided by a simple batch operating system, the processor is often idle. This is because the I/O devices are slow compared to the processor.



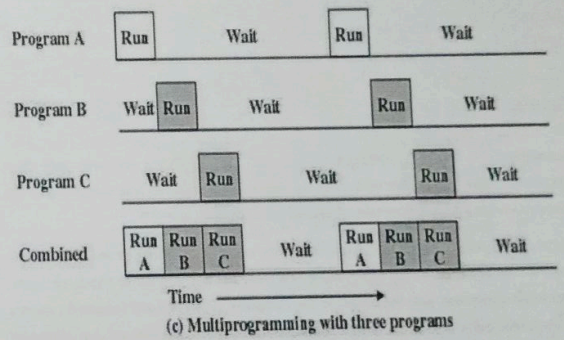
(a) Uniprogramming

In the above figure, we have only a single program which is referred to as uniprogramming. The processor spends a certain amount of time executing, until it reaches an I/O instruction. It must wait until that I/O instruction concludes before proceeding.

This inefficiency is not necessary. We know that there must be enough memory to hold the OS.(resident monitor)and one user program. Suppose that there is a room for OS and two user programs. When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O. This is shown in the below figure.



Furthermore, we might expand memory to hold three, four, or more programs and switch among all of them as shown in the below figure. This approach is known as multiprogramming or multitasking.



(c) Multiprogramming with three programs

To illustrate the benefit of multiprogramming, consider the following example.

Consider a computer with 250 Mbytes of available memory, a disk, a terminal and a printer.