Course Code:	**CSC 322**
Course Title:	**Computer Operating System I**
Course Unit:	**3**
Course Duration:	**2**

COURSE DETAILS

Course Lecturer: A. J. Ikuomola
Department of Computer Science
College of Natural Science
University of AgricultureAbeokuta,
Ogun State, Nigeria

Emails: ikuomolaaj@unaab.edu.ng
Office Location: AMREC Building

COURSE CONTENT

UNIT ONE
OPERATING SYSTEM CONCEPT
Introduction
Feature of O.S.
Categories of O.S.
Historical Development
Brief history of MS-DOS
Windows O.S.
Features of Window
A brief history of Window O.S.
Component of an O.S.
Types of O.S
Starting the Computer System
Part of DOS

UNIT TWO
PROCESS MANAGEMENT 1
Basic Concept
Introduction to Scheduling
Interrupt and their Processing

UNIT THREE
PROCESS MANAGEMENT 2: SCHEDULING
Objectives
Criteria for scheduling
High Level Scheduling

Medium Level Scheduling
Low Level Scheduling
Cooperative Level Scheduling
Process in Window

UNIT FOUR
MEMORY MANAGEMENT 1: REAL
Introduction
Process Loading and Swapping
Storage Allocation
Memory Allocation

UNIT FIVE
MEMORY MANAGEMENT 2: VIRTUAL MEMORY

Introduction
Mechanism of Virtual Memory

UNIT SIX
MEMORY MANAGEMENT 3
MS-Dos Memory Management

UNIT SEVEN
INPUT-OUTPUT
Organization of I/O Software and Hardware
Characteristics of I/O Device
Objectives of I/O System
Structure of I/O System

COURSE REQUIREMENTS

This is a compulsory course for students in the Department of Computer Science. In view of this, students are expected to participate in all the activities and have a minimum of 75% attendance to be able to write the final examination.

READING LIST

Brown B. (2000). Operating System Introduction.
http://physinfo.ulb.ac.be/cit_courseware/opsys/os01.htm#whatis
Rinard M.C. Operating System Lecture Notes.
http://oopweb.com/OS/Documents/OSNotes/VolumeFrames.html

Tanenbaum A.S & Woodhull A.S. Operating Systems Design And Implementation , *Second Edition*
ISBN 0-13-638677-6

UNIVERSITY OF AGRICULTURE ABEOKUTA

MODULE ONE
OPERATING SYSTEM CONCEPT

UNIT 1

INTRODUCTION TO OPERATING SYSTEM

An operating system is a set of program implemented either in software or firmware or both that make the hardware usable. It is an organized collection of programs and data that acts as interface between the computer hardware and the users providing users with a set of facilities for program design, coding, maintenance, etc.

It can also be viewed as a resources manager. The resources it manages are:

- Processor
- secondary storage devices
- I/O devices
- memory
- data

Feature of O.S

- Defining the user interface
- allowing user to share data
- sharing hardware among users
- scheduling resources among users
- facilitating I/O
- recovering from error

O.S interface with

- hardware
- program

- user
- administrative personnel
- computer operator
- system programmer
- application programmer

2 Categories of O.S

- **Single user O.S**

  Example
  - MS-DOS
  - OS/2

- **Multi-user O.S**

  Example
  - UNIX
  - XENIX
  - PC MOS
  - LINUX
  - NOVELL Netware
  - WINDOWS –NT
  - WINDOWS-XP

Types of Operating System

The two main types of O.S. are:

(a) Command- driven O.S.: the commands or instruction to this type of O.S. is entered through the keyboard or by using batch files. It is the most flexible way to use an O.S.It is also the most difficult way because syntax of the command have to be learn e.g. MSDOS commands.

(b) Menu-driven O.S. some O.S. employs menu which allows the operator to use icon on the window. It is a batch file selects O.S functions that form the list of choice displayed on the screen.

Some O.S. offer on-screen helps messages to remind the user on how to use system functions while others employ windows as a menu to display the O.S. function available and the status of the files which the user is working on.

A window is an area of the display screen set aside to show the content or status of files; such O.S. allow user to display many windows(say 3) on the screen at the same time.

UNIT 2

HISTORICAL DEVELOPMENT OF OPERATING SYSTEM

O.S like computer hardware has undergone a series of revolutionary changes called generation. In computer hardware, generations have been marked by major advance in component from Vacuum Tube to Transistors to Integrate Circuits to Very Large Scales Integrated Circuit.

ZEROTH GENERATION (1940'S)

These are the early computer system and they were very large and had no operating system. Users communicate with the machine directly in machine language. The job-by-job processing mode was used.Only one user could work at a time. Each user operate the system personally by loading decks, pushing buttons; examining storage locations etc. no batch processing was done then. There was no multi-user (multi-programming) facility. Therefore, the set-up and tear down operations consumed a lot of time.

Setting up: It involved putting the machine in an active state and leading a job individually usually from cards (which was introduced about 1880). This job had the whole memory during duration.

Tearing down: when a job runs to completion or terminates because of some error situation, an operator will load a program to dump the memory. He then removes the cards and printed output. He also takes the machine back to its initial state (interactive state), no other job should be run or should be in the active state if another job is to be processed.

Thus, a small program requiring a little CPU time would take so long to complete because of the set up and tear down times.

FIRST GENERATION (1950'S)

Computing systems had reduced in size (due to the introduction of transistors, current could now flow through wires) though they were still large.

The first generation of operating system was design to automate the set up and tear down of jobs (i.e. to smoothen the transistors between jobs). This was achieved through batch processing. Jobs were gathered in 'batches' such that one job was processed after the other without users interfering.
This means once a job is running; it had total control of the machine. As each job terminated, control was returned to the O.S. that assist the job (house keeping operations) and the next job is read in.

By the late 1990's operating system had the following characteristics

1. Single stream batch processing i.e. program to program transition capabilities in order to reduce the overhead involved in starting a new job.
2. Error recovery techniques that automatically 'cleared up' after a job terminated abnormally and allowed the next job to be initiated with minimal operator intersection.
3. Job control languages that allow users to specify much of the details for refining their jobs, the resources the job requested and accounting.
4. Operating system had standard I/O routines called IDOS(I/O) control services so that users did not have scope concerned with the messy details of machine level decoding of the input and output resources.
5. Paging and virtual storage concepts were introduced but not implemented. Assembly language was introduced.

These systems are often heavily under-utilized. It is far more important for them to be available when needed and to respond quickly than for them to be busy throughout the time. This fact helps explain their cost.

SECOND GENERATION (EARLY 1960)

This OS were batch-oriented and were concerned mostly with improving throughput i.e. work processed per unit time with expensive hardware. Multiprogramming was commonly employed in which general programs were resident in the main memory at the same time and the processor switched rapidly between them (programs). This increased throughput and the imbalance in the speed between I/O devices and processor.

Multiprocessing systems emerged in which several processors cooperate sometimes as independent computer systems communicating with each other, and sometimes as multiple processors sharing a common memory.

Still in the early sixties, time sharing systems using an interactive mode were developed in which user could interact directly with the computer through typewriter terminals. This helped to eradicate the suffering of delays for hours or days on the batch processing environment. Users could now share data and programs. This increased productivity and introduced creativity among users. Most time sharing users then spend their time developing programs or running especially designed application programs. Errors in the earliest phases of the projects were not located until long after the projects were delivered to customers.

This led to the emergence of the field of software engineering in order to facilitate a disciplined and structured approach to the construction of reliable, understandable and maintainable software, e.g. Burroughs introduced an operating system called MCP (Master control program) in 1960.

This O.S had the following features:

- Multiprogramming
- Multiprocessing
- Virtual storage
- O.S written in a high level language
- Source language debugging facilities

THIRD GENERATION (MID 1960S – MID 1970S)

It began effectively with the introduction of IBM system 360 families of computers. Third generation computers were designed to be general-purpose computers. This third generation O.S were multimode systems. Some of them simultaneously supporting batch processing, time sharing of the processing. They are large and expensive.

Throughout the early 70's vendors sold hardware and gave out O.S. Support programs, application programs, documentation and educational manuals for no charge. Thus computer vendors didn't take much responsibility for the O.S.

Unbundling the software from the hardware:

IBM was the first company to unbundle its software from its hardware i.e. they charged separately for each, although IBM continued to supply some basic software for no charge.

- Customers taking responsibility for the quality of their software.
- Vendor began to design their software more modularly so that they could be sold as individual units.
- Users could now shop around for their software.
- Others manufacturer unbundled rapidly.
- Some concepts disappeared and later reappeared indifferent forms paging and virtual storage concepts.
- O.S was developed for families of machine.
- The merging of multiprogramming, batch philosophy with time sharing technology form O.S. capable of handling both batch and time sharing operations.

FOURTH GENERATION (MID 70'S TO PRESENT DATE)

The very large scale integrated circuits were introduced which led to the development of microcomputer. Software was developed for family of machines. Networking was enhanced, such that users were no longer confined to communicating with a single computer in a time shared mode but rather the user may communicate with geographically dispersed systems.This gave room for security problems with geographically passing over various types of vulnerable communications lines. Encryption received attention. Thus it became necessary to encode data so that it is only useful to the receiver.

The highly symbolic mnemonics acronym-oriented user environment was replaced with menu-driven system that guided users through various available options in English language. The concept of virtual machines becomes widely used. Today's user is not concerned with the internal functioning of the machine, but to accomplished work with a computer. Database systems have gained wide acceptance and importance.

Thousands of online databases have become available for access via terminals over communication networks.

The concept of distributed data processing has become firmly established.

UNIT 3
BRIEF HISTORY OF MS-DOS AND WINDOWS O.S

MICROSOFT DISK OPERATING SYSTEM (MS-DOS)

The history of MS-DOS is related to the IBM PC and compatibles towards the end of the 1970s, a number of PCs appeared in the market, based on 8-bit microchips such as inter 8080. IBM decided to introduce the PC into the market, if is possible and released it without having enough to develop its own O.S., at that time; CP/M (by digital reason) dominated the market. In 1979, a small company settles the computer products developed its own O.S., 86-DOS to rest some of its Intel based products (86-DOS was designed to best similar to CP/M). IBM purchased 86-DOS and in collaboration with Microsoft developed a commercial product, MS-DOS version 1.0 which was announced in August 1981. MS-DOS version 1.0 was also referred to as PC-DOS. MS-DOS had some similarities to CP/M, (such as the one level filed storage system for floppy disks) which was important in terms of market acceptance in those days although MSDOS did offer several improvements over CP/M such as:

- a larger disk sector size (512 bytes as opposed to 128 bytes).
- a memory-based file allocation table.

Both of which improved disk file performance.

A summary of the most significant features of versions of MS-DOS are listed below.

VERSION	DATE	FEATURES
1.0	1981	Base on IBM PC Designed to cater for floppy disks and therefore used a simple file storage system, similar to CP/M. A memory-based file allocation table. - A larger disk sector 512 bytes.
2.0	1983	-Based on IBM PC/XT with 20 MB hard disk -A hierarchical file directory to simplify the vast storage. - Installable device drives.
3.0	1984	- Based on IBM PC/AT with 20 MB hard disk. - Supported RAM disked

3.2	1986	3.5 inch disks Support for IBM Token Ring Network.
3.3	1987	-Support for new IBM PS/2 computers -1.44Mb floppies. -Multiple 32MB disk partitions -Support for expanded memory system.
4.0	1988	-Simple window based command shell -Up to 2 Gigabyte disk partition
5.0	1991	-Improved memory management -improved shell and extended command -2.88MB floppies
6.0	1993	-improved memory management -Doubling the disk space by compressing files for floppies and hard disks -Interlink: a program that transfers files within computers. -Antivirus facility that can remove more than 800 viruses from your system -Improved extended commands -A diagnostic facility: a program that gathers and displays technical information about your system. -Power is a program that conserves battery power when applications and hardware devices are idle

STARTING THE COMPUTER SYSTEM

The process of starting up the computer is known as BOOTING. When you put on the computer system, it starts up (boot) by itself i.e. it carries out a self-test on the memory and other components. You insert DOS disk #1 in the drive if DOS is not on the hard disk. If the self-test is successful, the disk driver will spin and the following operations are performed.

- ROMBIOS is loaded. It is a short program that can read the cost of the I/O system.
- MSDOS.SYS is loaded which initialized the hardware of the computer.
- I/O.SYS is also loaded.
- COMMAND.COM. , It loads the COMMAND.COM.
- AUTOEXEC.BAT of the system, takes over control and runs the batch file

Part of DOS

DOS is made up of several programs, a command processor, an input/ output system and several utilities.DOS is normally supplied in between 2-2 magnetic disk media. The first diskette contains the command processor and other necessary system files for starting up the system.

The other diskettes are the supplementary diskettes that contain the several utility programs. MSDOS.SYS, 10.SYS and the command processor/ shell (which is also known as COMMAND.COM on the disk #1) are the only files needed to start up (or boot) the computer system.

Function of COMMAND.COM File

(1) it handles critical errors if the disk drive doors is left open or the keyboard is left uncorrected.COMMAND.COM is responsible for putting a message on the screen to notify the user.

(2) It handles critical errors interrupts. COMMAND.COM takes care of all demands for attention by parts of the computer. e.g. the user can stop an operation being processed.

(3) It is responsible for placing the DOS prompt (A OR C) on the screen. It interprets all commands given to it and tells the rest of DOS what to do.

(4) It performs an end-of-program housekeeping i.e. COMMAND.COM takes care of making the computer's memory available for other programs.

(5) The input / output system is made up of two parts which consists of the files and a ROM chip.

BIOS (basic input output system) comprise the fundamental routine that controls the keyboard, VDU and other peripheral devices. It consists of a ROM chip and the file MSDOS.SYS.

Operating system which is the main file handler on the computer system is the IO.SYS.

The two files MSDOS.SYS and part of DOS files, both are hidden from view. The utilities are individual program files found on DOS diskette. Examples include:

FORMAT.COM	-	format disks
DISKCOMP.COM	-	compare disks
DISKCOPY.COM	-	copy disks.
MOVE.EXE	-	move files.

WINDOWS OPERATING SYSTEM

Definition of Windows

Windows is an Operating System that supervises other programs in the computer. Windows provides a user friendly graphic environment, referred to as Graphical User Interface (GUI) for the user. An interface is the common boundary between the user and the computer. Its features influence the effectiveness of the user on the system.

Features/Benefits of Windows

- Windows provides an easy to use graphical user interface (GUI) to run our programs and applications.
- Windows allows us to run more than one task or program at the same time. This is called multi-tasking. For example, you can be typing your document and also listen to favourite CD playing on the CD-ROM Drive.
- Windows allows us to use more than one input device to work with. You can use the keyboard, a mouse or joystick to work.
- Windows allows us to add new equipment to our computer. Without switching it off. We can add a printer, CD ROM etc. and the computer will recognize it automatically. This is called Plug and Play capability.
- Windows allows us to save our files with names we like. This name can be as long as 255 characters, which is contrary to DOS that allowed a maximum of eight characters for filenames.
- Enhanced backup and restore functionality supports more drives and the latest hardware.
- Application loading, system startup, and shut down time are faster.
- Support for DVD and digit audio delivers high-quality digital movies and audio direct to your TV or PC monitor.
- Windows supports Microsoft Internet Explorer, which includes Outlook, a new e-mail client and collaboration tool. The Active Desktop Interface puts Internet and Intranet pages directly on a user's desktop.

A brief history of Windows System

Version	Date	Summary of main features
1	1985	Ran on Intel 286 processors. Attempt to emulate Apple interface. Provided basic iconic program and file facilities. Used tiled windows
2	1988	Used overlapping windows. 286 and 386 version produced.
3	1990	Protected mode of Intel 386 used up to 16 Mbytes of memory available. Proportional fonts. Highly successful.
3.1	1991	Introduced OLE. Support for True Type fonts.
3.11	1992	Update to 3.1.
3.11 WfW	1992	Windows for Workgroups; a version with integrated network capability.
OS/2 1.0	1991	Intended as a multi-tasking replacement for MS-Dos. Ran on Intel 286 processors. Supported MS-Dos and Windows 3.1 applications.
OS/2 2.0	1992	Supported MS-DOS and Windows 3.1 applications.
OS/2 2.1	1993	Better performance and support for Windows 3.1 applications. 32-bit graphics engine.
NT 3.1	1992	First version, labeled '3.1' for continuity with the basic Windows series. First MS-Dos independent operating system from Microsoft, designed from scratch. Extensive 32-bit working.
NT 3.5	1994	Also known as Daytona. Various improvements to performance, 32 bit working and networking
OS/2 Warp	1994	Special version of OS/2 which is installed 'on top' of existing Windows 3.1 installation. Designed to run satisfactorily in 4 Mbytes of main memory and

		provides pre-emptive scheduling, threads and memory protection.
95	1995	Also known as Chicago. Upgrade from, and compatible with, 3.1 but with support for 32 bit applications and 'MS-Dos free'. Usable on smaller machines than NT. Support for OLE 2. New design of user interface with more object-oriented features.
NT 4.0	1996	Introduction of new-look Windows 95-style user interface. Incorporates support for Internet and DCOM
98	1998	Windows 98 introduced

UNIT 3

COMPONENT OF AN OPERATING SYSTEM

An O.S. is primarily a provider and manager of machines resources: processors, main memory, secondary storage, I/O devices. Access to these resources is centralized and controlled by various modules of the system. In addition, the O.S. provides other services such as interface, data security etc. the O.S. structure or shell could therefore be illustrated as thus:

The O.S. shell is a multi-layered set of programs:
The kernel is at the centre of the shell. It is referred to as the firmware and usually in ROM as part of the hardware. In some system, it is separated from the supervisor. It is limited to the initial startup operations and programming in machine language.

The Residence/Component/Structure of an O/S

The O.S. is made up of 2 main parts:

(1) the system programs

(2) the program library

(1) The System Programs

The basic function of O.S. surrounds the supervisor and they include I/O management, memory management, file management, and job control. When put together, these functions form a group of programs called system programs. System programs under the control of the supervisors yields the complete O.S., other utilities and application programs surround these functions.

The system program comprises of:

(a) The Supervisor

The supervisor (also known as O.S or DOS) surrounds the kernel. It is the interface between user jobs and system resources. It also controls and coordinates system resources. It is the heart of the O.S

(b) Input/Output management

It is concerned with management of I/O devices. It controls spooling and buffering, multi-tasking and overlapping, time- sharing, and networking.

(c) Memory management

It is concerned with the management of RAM. It includes the allocation of RAM for various purposes, background program priorities and virtual memory systems.

Its operations include:

- Keeping track of the
- Deciding on the allocation
- Allocating memory
- De-allocating the portion of memory used when it is no longer in use.

(d) File management

It is concerned with files stored on secondary storage media. These files can be copied, sorted, displayed and removed among other functions.

Its operations include:

- providing access to files and regulate the use of I/O devices
- supervising the transfer of files between the O/I devices and processor (and vice versa).
- controlling the storage and retrieval of files to be processed by the computer.
- The control function establishes the user right to access or modify files. The storage function is the technique of representing files on storage devices.
- The retrieval function involves the implementation of locating files and arranging them for the processor.

(e) Job Management or Job Control Function of O.S.

It is concerned with the management of jobs. It is a function of the process management. A job is one or more programs that are logically related e.g. a program that calculated the G.P.A. of students, a student's M.I.S. which is made up of several programs.

Functions

- To provides the control from job to job or program to program;
- It is enhanced by the availability of job control languages (JCL)

Job Control Languages (JCL)

Job control languages are used to specify and control the running of batch jobs generally on large computers. JCL programming is an important work, some programmers called system programmers

specialize in this activity alone. Most users and system programmers use such terms as system commands and DOS commands when referring to instructions that tell the O.S. what to do.

Job control functions include executing programs on demand batch programs for command starts for automatic execution of O.S. Functions.

e.g.

OPERATION	**UNIX**	**DOS**
Copy files	**C>**	Copy
Display directory	Ls-L	Dir
Rename File	Mv-	Rename
Compare files	Cmp	Comp
Display contents of files	At	Type

In an old mainframe system (IBM 360/370), the JCL for Fortran compiler include the following:

```
//job s426 name = Ayo, O.      Dept=computer
// option link
// exce Fortran
                    FORTRAN program
//execlinked
//exec.
/*                   Data
/&
```

The job identifies a user's job and its requirements to the system.

The JCL for GWBASIC (a microcomputer-based basic computer)

```
load          Files
list          Cnt
run
save
```

Sometimes, a sequential series of system commands are store as separate file for execution as a program. Such a file of system commands is known as a batch file, exe. File, shell programs or JCL program. Batch files are very useful for setting up memory configurations, I/O device identifiers, and other

housekeeping chores that the computer operator must perform regularly. They are also used to established a sequence of application programs to run.

(f) Process management

It is concerned with the assignment of the physical processor (s) to processes. It includes the following:

- the job scheduler
- the processor scheduler
- the traffic controller

(g) Transient utilities –utilities on the system disks;COPY, FORMAT, DISKCOPY e.t.c.

(2) The Program Library

The program library comprises of:

(a) Language Translators

A language translator converts a high level language program (usually referred to as a source program) to its equivalent machine language program referred to as an object program.

There are three types of translators:

(i) Compiler: A complier converts only high level language programs. It translates an entire program into its machine equivalent before the program is executed. The translation process is called compilation. Compiler translates the whole of the High Level Language.

(ii) Interpreter: The interpreter translates program written in high level language one at a time i.e. completely translating and executing each other instruction before going to the next statement.

(iii) Assembler:These are programs which translate a source program written in an assembly or programming language into a machine code object program.

	Complier	**Interpreter**
1	Translate all at once	Translate one by one
2	Execution is faster	Execution is slow
3	A little but tedious	Easy to use and learn
4	All error's given at once	One at a time

(b) Application Packages

They are ready made programs designed in a standard way of applications which are common to a group of people. They are either:

- tailor made (developed by a team of computer people) or
- bought off- shelf.

(c) Other Utilities –PCTOOLS, NORTON, VIRUS programs e.t.c

Utility program or service program are general purpose programs that can be used in conjunction with many application program e.g. copying files, Disk copy, sorting files in ascending or descending order by using some control keys, checking the integrity of disks, formatting disk, creating batch files using DOS editor, undeleting files, disk compression and to perform diagnostic routines.

The disk on which the O.S. resides is usually called the system resident device, sys-res or DOS-res. This disk is usually the diskette or hard disk. Not all programs making up the shell are available in RAM at once, the program available in RAM are called resident or internal system programs. They may be executed by the user directly or through application programs. The program that are not available in RAM are called transient or external system program/ transients and have to be loaded into the RAM prior to its execution.

More recently, a layer for the user interface was introduced between the basic functions of the O.S. and the translators/application packages layer. The user interface is the user gateway into the computer, enabling the required human computer interaction to take place later.

MODULE TWO

PROCESS MANAGEMENT 1

UNIT 1

BASIC CONCEPT

1.1 INTRODUCTION

The concept of a process is central to the study of a modern operating system. It was first used by the designers of Multics in 1960. Since then, process (which is used interchangeably with task) has been given many definitions.

Definition of a Process

- a program in execution
- that which a processor executes
- that unit of code (i.e. program) that needs a processor
- the entity to which processor are assigned
- the animated spirit of a procedure
- the locus of control of a procedure in execution
- that which is manifested by the existence of a process that control the dispatched unit.

Many other definitions have been given and there is no universally agreed upon definition but the program in execution concept seems to be most frequently used.

When a user initiative a program the O.S. operates a process to represent the execution of the program. The process, thus created consists of the machine code image of the program in the memory, the Process Control Block (PCB) structure (discussed below) and possibly other structures used to manage the process during its lifetime.

The processor at any instant can only be executing one instruction from one program but several processes can be sustained over a period of time by assigning each processwhich becomes temporarily inactive. The process whenstarted has complete control of the processor until either:

(i) the process issued I/O request by means of system call or

(ii) interrupt occurs.

Each process has a data structure called the process control block (PCB)which is created by the O.S. This block contains the following information that assist the O.S. to locate all key information about the process:

- the cursor state of the process
- the process identification
- the priority level of the process
- pointers to allocate the process in memory
- pointers to allocate resources
- a register save area

1.2 INTRODUCTION TO SCHEDULING

In a multiprogramming environment, several processes will be competing for the processor. At any instant, only one process will be running while the others will be ready waiting for the processor or in some other wait condition. The O.S has to determine the optimum sequence and timing of assisting processes to the processor. This is referred to as **SCHEDULLING**.

Scheduling can be exercise at three distinct levels:

- high level
- medium level and
- low level

Note that not all system utilities or identify all three.

High level scheduling (long term or job scheduling) (H.L.S) decide whether a new job should be admitted into the system or not. It sometimes known as admission scheduling. It was only useful in the older systems.

Medium level scheduling (or intermediate scheduling) is concerned with the decision to temporarily remove a process from the system (in order to reduce the system load or to process).

Low level scheduling (short term or processor scheduling) decides on the ready process to be assigned to the processor. This level is often called the dispatcher but the term is more accurately refers to as the actual activity of transferring control to the selected process.

UNIT 2

PROCESS STATE

2.1 INTRODUCTION

A process goes through a series of discrete state during its lifetime in an unprocessed system (i.e. a mono or multiprogramming system with a single CPU), only one process can be running at any instant of time. Several processes may be ready while many may be blocked.

This is illustrated in a 3-state model referred to as the process state diagram below.

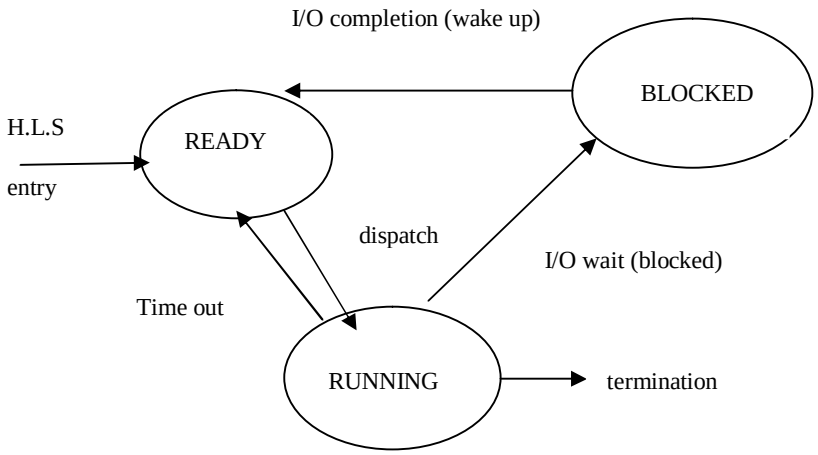


Figure 1: Three state model: process state diagram

- A process is in the **READY** state when it could use a CPU if one was available.
- A process is in the **RUNNING** state if it currently has the CPU
- A process is in the **BLOCKED** state if it waited for an event to happen (e.g. I/O completion) before it can proceed.

The following state transitions are therefore possible from the model above.

- Dispatch (process name): ready running
- Time out (process name): running ready
- Wake Up(process name): blocked ready
- Blocked (process name): running blocked

The only state transition initiated by the user process is block while other three transitionsare initiated by entities external to the process.

An example of the life cycle of process in UNIX is given below:

We assume that there are two processes involved:

(a) User A arrives and using the shell command interpreter and types in a program name, say, SOLVIT.

(b) The shell attempts to find the program and if successful, the program code will be loaded and a system call used to generate a process corresponding to the execution of the SOLVIT program.

(c) In order to physically represent the process within the computer, the O.S creates a data structure called process control block (PCB) in memory.

(d) The process 'SOLVIT' will now run, the process is said to be in the RUNNING **STATE**.

(e) After a while, SOLVIT needs to read some data from the hard disk (or diskette) and issues an appropriate system call. Since it will now have to wait until the file management subsystem complies with its request, the process is unable to continue. Process is now in the BLOCKED STATE.

(f) In the meantime, user B want to run a program say, myprog and type a suitable command. A now process is created for MYPROG and since SOLVIT is currently idle, executions begin with myprog which is now RUNNING.

(g) The I/O delay which is blocking SOLVIT now end and SOLVIT want to restart. However, it cannot, because MYPROG is using the processor. SOLVIT is now said to be the READY state processes in the READY and held in a queue and dealth with using various scheduling schemes which will be discuss later.

(h) The O.S scheduling now decides that MYPROG has enough processor time and moves MYPROG to the READY queue. Note that MYPROG become READY not blocked, since it did not issue an I/O request.

(i) SOLVIT is restarted and enter the RUNNING states once more

(j) The action of switching between active processes and waiting for I/O transfers continue throughout the lifetime of processes. Usually there are more than 2 processes competing. For the processor but the same general principle apply.

(k) Eventually SOLVITcompletes its task and terminates. It leaves the running state and disappears from the system.

2.2 OPERATION ON PROCESSES

System managing processes must be able to perform certain operations on them these operations include:

- Create a process
- Destroy a process
- Suspend a process
- Resume a process
- Change a process
- Block a process
- Wakeup a process
- Dispatch a process

Process could also be initiated by a user process such that a single program activated by a user could result eventually in several separate processes running simultaneously.

(a) Create a process

The process creating a new process is called the parent, while the created process is called the child. The act of creating a new process is often called spawning a process (i.e. a process can spawn a new process). A child process could itself spawn resulting in a tree of processes. Only one parent is needed to create a child. Such a creation yields a hierarchical process structure in which each child has a parent, but each parent may have many children.

Creating a process involved the following operations;

- name the process
- insert its name in the system list of the processes
- determine the process initial priority
- create the Process Control Block (PCB)
- allocate the initial resources

(b) Destroy a process

When a process terminates (or is destroyed) it will supply some return code to its parent indicating the success/failure of the child process mission. The process resources are return to the system lists or table and its PCB is erased.

A process spawned by the O.S. itself will report back to the system. It is possible for a parent process to terminate before a child process in this case the return code of the child is collected by the next higher level parent of the O.S.

(c) Suspend a process

When a process is suspended, it cannot process until another process resumes it. Suspension is an important operation and has been implemented in a variety of ways on different systems.

Suspension normally last for brief periods of time. They are often performed by the O.S to remove certain process temporarily during a peak loading situation. For long time suspension, the process resources and memory space are freed (this depends on the nature of the resource).

(d) Resume a process

Resuming or activating a process involves restoring it from the point at which it was suspended.

(e) Change a process

Changing the priority of the process normally involvesmodifying the priority value of the process in its PCB.

2.3 THE FIVE STATE MODELS

The three models are not sufficient to describe the behavior of the processes (i.e. their operation). The model is therefore extended to a given five model to allow for other possible event SUSPENDING AND REMOVING a process.

When a process is suspended, it becomes dormant until it is resumed by the system or user. A process can be suspended for a number of reasons which are:

(a) The most significant: the process is swapped out of the memory by the memory management system in order to free memory for other processes (this decision is taken by the scheduling system).
(b) The process is suspended by the user for a number of reasons such as during debugging, to investigate the partial result of the process.
(c) Some processes are designed to run periodically e.g. monitor system usage etc.

A process can be suspended while in one of the following states: READY, RUNNING or BLOCKED. This giving rise to two other states, namely, READY SUSPENDED and BLOCKED SUSPENDING.A RUNNING process which is suspended becomes READY SUSPENDED.

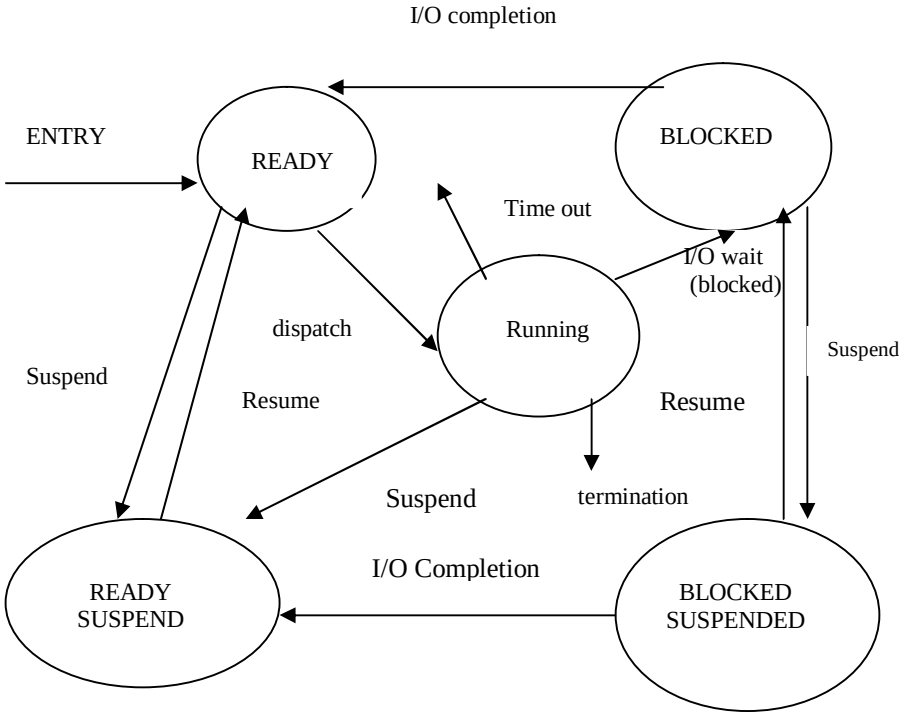


Figure 2: Five State Model - Process state diagram.

The following transitions are therefore possible from the model above.

- A **ready** process may be suspended only by another process **SUSPEND** (process name): ready→ suspended.
- A **ready** process may be made ready by another process **RESUME** (process name):ready-→ suspendedready.
- A **blocked** process may be suspended by another process **SUSPEND** (process name):blocked blocked-suspended.
- A **blocked-suspended** process may be made to resume by another process **RESUME** (process name): blocked-suspended blocked.
- A **running** process may be suspended by another process **SUSPEND** (process name): running ready-→ suspended
  Including all previous transition discussed in the 3-state model.
- A **ready** process may be dispatched by another process **DISPATCH** (process name): ready→running
- A **running** process may be made time out by another process **TIME OUT** (process name): running→ ready
- A **running** process may be blocked by another process **BLOCK** (process name): runningblock
- A **block** process may be completed by another process **COMPLETION** (process name): block→ ready
- A **blocked suspended** process may be completed by another process **COMPLETION**(process name): blockedsuspendedready suspended.

UNIT 3

INTERRUPT AND THEIR PROCESSING

3.1 Definition of an Interrupt

An interrupt is an event that alters the sequence in which the processors execute instructions. It is usually generated by electric signals by the computer hardware. It is a response to an asynchronous or exceptional event that automatically saves the current CPU status to allow later restart and causes the automatic transfer to a specific routine (the interrupt handler).

When an interrupt occur, the Operating System:

(i) Gains control,

(ii) Saves the state of the interrupted process, usually in the interrupt 'process' PCB (Process Control Block),

(iii) Analyses the interrupt and,

(iv) Transfer control to the specific routine to handle the interrupt(interrupt handler).

3.2 Types of Interrupt

The five types of interruptare:

- program interrupt
- supervisor call interrupts (SVC)
- I/O interrupt
- External interrupt
- Machine check interrupt

(a) Program Interrupt

A program interrupt is generated by error continuously arising within the user programs. It is generated by the processor.

Typical events are:

(i) Address violation : is an attempt to address outside a program's memory space. It is referred to as the protection interrupt. This could also be applied to an attempt by a user program to execute a privileged instruction (O.S service).

(ii) Execution of an invalid instruction e.g.

N = 0

```
DO 10 I = 1, N, 2
10 CONTINUE
```

(iii)Arithmetic overflow: An attempt to divide a number with zero

(b) Supervisor Call Interrupt (SVC)

A supervisor call (SVC) interrupt is generated by a running process that executes an acceptable SVC instruction. A SVC instruction is a user generated request for a particular system service such as performing I/O, obtaining more storage or communicating with other user/systems' operator. Typical examples of unacceptable SVC instructions are:

(i) An attempt by program to access a portion of a job that is not currently in RAM.

(ii) An attempt to access a segment that is not presently in the memory.

(c) Input Output (I/O) interrupt

An I/O interrupt is generated by the controller of an I/O device to signal the normal completion, start up, occurrence of an error or failure condition of an I/O device.

Typical events are:

An invalid I/O command e.g.

```
READ (1, 5) NAME, AGE
5 FORMAT (12, A4).
```

An I/O channel ends its job i.e. An I/O channel end interrupt occurs, when the channel finishes its job before the device does which is normal e.g. The output channel transfer data from the memory to the printer's interrupt buffer. The channel will finishes transferring the last batch of data before the printer finishes printing the job.

(d) External Interrupt

An external interrupt is generated by any machine component. It is not necessarily signal for a fault.

Typical events are:

- a user aborts/terminates his program
- a signal to the O.S. of a user's time slice. This is generated by the internal clock within the processor.
- the recipient of a signal from another processor on a multiuser processor system

(e) Machine Check Interrupt

A machine check interrupt is generated by the malfunctioning of the hardware.

Typical events are:

- the screen showing 'fixed disk controller bad'
- the VDU showing 'keyboard bad'

3.3 Program Status Word (PSW)

The PSW is a special region that gives the complete picture of the system at any time. It is dynamic in nature and change very frequently to show the current picture of the system.

3.4 Interrupt Handlers (I.H)

The O.S includes routines called IH to process each type of interrupt. There are 5 types of interrupt handlers namely: SVC, IH, External IH, program and machine check IH. IH's are stored in a special area in RAM.

The sequences of Events that occur when an interrupt occurs (interrupt processing) are:

(i) The processor stops executing the job or program,

(ii) The O.S. saves the current state of the CPU (i.e. the interrupted process) in the OLD PSW.

(iii)Control is transferred to the I.H.

(iv) The interrupt becomes the current process. The appropriate I.H. required is selected. The address of this I.H. is stored in the NEW PSW.

(v) The I.H. load the NEW PSW from its position to the CURRENT PSW(the CURRENTPSW now contains the address of the appropriate I.H)

(vii) The IH analyses and processes the interrupts (the problem is solved)

(viii)The IH signals the system at the completion of its task.

(ix) The OLD PSW is reloaded to the CURRENT PSWand the next instruction on the interrupted program is executed.

This implies that there is only one CURRENT PSW though its content changes periodically. The action of holding the current state of a process which has been temporarily stopped and the starting of another process is called content switching (or content change).

MODULE THREE
PROCESS MANAGEMENT 2

UNIT 1

SCHEDULING

Objectives

The overall scheduling is intended to meet some objectives in terms of the system's performance and behaviour. The scheduling system should:

Objectives of scheduling

The overall scheduling effort is intended to meet some objectives in terms of the system's performance and behavior. The scheduling system should-

- Maximize the system throughput.
- Be 'fair' to all users. This does not mean all users must be treated equally, but consistently, relative to the importance of the work being done.
- Provide tolerable response (for on-line users) or turn-around time (for batch users).
- Degrade performance gracefully. If the system becomes overloaded, it should not 'collapse', but avoid further loading (e.g. by inhibiting any new jobs or users) and/or temporarily reduce the level of service (e.g. response time).
- Be consistent and predictable. The response and turn-around time should be relatively stable from day to day.

Criteria for Scheduling

In making decisions about the scheduling of processor work, a number of criteria can be taken into account by the operating system. The precise effect of these factors become more evident shortly when we describe specific scheduling algorithms. The criteria to be considered include the following:

- Priority assigned to job.
- Class of job; i.e. batch or on-line or real-time. On-line users require a tolerable response time, while real-time systems often demand instant service.
- Resource requirements; e.g. expected run-time, memory required, etc.

- I/O or CPU bound; i.e. whether job uses predominately I/O time or processor time. This criterion is often of consequence because of the need to balance the use of processor and the I/O system. If the processor is absorbed in CPU- intensive work, it is unlikely that the I/O devices are being serviced frequently enough to sustain maximum throughput.
- Resources used to date; e.g. the amount of processor time already consumed.
- Waiting time to date; i.e. the amount of time spent waiting for service so far

It can be seen that some of these factors are 'static' characteristics which can be assessed prior to commencement of the process execution. Of particular interest in this respect is the notion of a priority. This is a value which can assign to each process and indicates the relative 'importance' of the process, such that a high priority process will be selected for execution in preference to a lower priority one. Scheduling in this way on the basis of a single priority value enables rapid decisions to be made by the scheduler. An initial priority can be assigned to each process, in some schemes, the priority is static and is used as a basis for scheduling throughout the life of the process, and while in other schemes the priority is dynamic, being modified to reflect the changing importance of the process. The priority can be supplied by a user or could be derived from the characteristics of the job, or both.

UNIT 2

SCHEDULING SCHEME

1. HIGH LEVEL SCHEDULING (HLS)

The High Level Scheduling (HLS) controls the admission of jobs into the system; i.e. it decides which newly submitted jobs are to be converted into process and be put into READY, queue to compete for access to the processor. This activity is only really applicable to batch systems, since in an on-line environment; processes will be admitted immediately unless the system is fully loaded. In this event, new logins will be inhibited (rather than being queued).

New jobs entered into the system will be put into a queue awaiting acceptance by the HLS. The principal control which the HLS exercises is ensuring that the computer is not overloaded, in the sense that the number of active processes (the degree of multiprogramming) is below a level consistent with efficient running of the system.

If the system loading is considered to be at its acceptable maximum, new processes may only be admitted when a current process terminates.

If the loading level is below maximum, a waiting process will be selected from the queue on the basis of some selected algorithm. This may be a simple decision such as First-Come-First-Served (FCFS) or it may attempt to improve the performance of the system using a more elaborate scheme. A possibility in this respect is known as shortest job first (SJF) which selects the waiting job which has the shortest estimated run time.

2. MEDIUM LEVEL SCHEDULING (MLS)

Medium level scheduling is applicable to systems where a process within the system (but not currently running) can be swapped out of memory on to disk in order to reduce the system loading. Although a process sitting in the READY or BLOCKED queue is not actively using the processor, it is nonetheless competing for processor use and will consume processor time at some future times. The medium level scheduling attempts to relieve temporary overloading by removing process from the system for short periods. Ideally, the process chosen for swapping out will be currently inactive; they will be re-introduced when the loading situation improves. Meanwhile, the processes are held in the READY SUSPENDED or BLOCKED SUSPENDED state.

3. LOW LEVEL SCHEDULING (LLS)

The low level scheduling (LLS) is the most complex and significant of the scheduling levels. Whereas the high and medium level schedulers operate over time scale of seconds or minutes, the LLS makes critical decisions many times every second. The LLS will invoke whenever the current process relinquished control, which, as we have seen, will occur when the process calls for an I/O transfer or some other interrupt arises. A number of different policies have been devised for use in low level schedulers, each of which has its own advantages and disadvantages. These policies can be categorized as either Preemptive or Non-Preemptive policies.

Preemptive/ Non-preemptive policies

In a preemptive scheme, the LLS may remove a process from the RUNNING state in order to allow another process to run.

In a non-preemptive scheme, a process, once given the processor, will be allowed to run until terminates or incurs an I/O wait; i.e. It cannot 'forcibly' lose the processor. (Non-preemptive mode is also known as 'run to completion', which is slightly inaccurate, since the process will lose the processor if it incurs an I/O wait). Note that in a non-preemptive scheme the running process retains the processor until it 'voluntarily' gives it up; it is not affected by external events.

A preemptive scheme will incur greater overheads since it will generate more context switches but is often desirable in order to avoid one (possibly long) process from monopolizing the processor and to guarantee a reasonable level of service for all processes. In particular, a preemptive scheme is generally necessary in an on-line environment and absolutely essential in a real-time one.

Cooperative scheduling

Earlier versions of windows (up to version 3.11) appear to provide non-preemptive scheduling. In fact, the technique used is rather primitive; the responsibility for releasing control is placed in the hands of the application programs and is not managed by the operating system. That is, each application, when executing and therefore holding the processor, is expected periodically to, relinquish control back to the windows scheduler. The operation is generally incorporated into the event processing loop of each application; if there are no event messages for that application requiring action; control is passed back to the scheduler. This mode of working is called cooperative scheduling. Its main disadvantage is that the operating system does not have overall control of the situation and it is possible for the whole computer to freeze. Note that it differ from a conventional

non-preemptive system in that in the latter, the process will lose control as soon as it requires an I/O operation. At this point, the operating system regains control.

3.1 Low level scheduling policies

The following section describes a number of common scheduling policies which are listed below:

- First –Come-First-Served (FCSF)
- Shortest job first (SJF)
- Round Robin (RR)
- Shortest Remaining Time (SRT)
- Highest Response Ratio Next (HRN)
- Multi-level Feedback Queues (MFQ)

(a) First –Come-First- Served (FCFS)

Also known as first out (FCFS). As the name implies, the FCFS policy simply assigns the processor to the process which is first in the READY queue; ie has been waiting for the longest time. This is a non-preemptive scheme, since it is actioned only when the current process relinquishes control.

FCFS favors long jobs over short ones, as can be illustrated by the following example. If we assume the arrival in the READY queue of four processes in the numbered sequence, we can calculate how long each process has to wait.

Table 1: Illustrate FCFS policy

	(a)	**(b)**	**(c)**
Job	**Est. Run Time**	**Waiting**	**Ratio b/a**
1	2	0	0
2	60	2	0.03
3	1	62	62
4	3	63	21
5	50	66	1.32

If we make the nationally fair assumption that the waiting time for a process should be commensurate with its run time, then the ratio of waiting- time/run- time should be about the same for each job. However, we can see from column (c) above that this ratio for small jobs 3 and 4 is

very large, while being reasonable for long jobs. The above example is admittedly somewhat contrived but it does indicate how method can be unfair to short processes.

Another problem with FCFS is that if a CPU-bound process gets the processor, it will run for relatively long periods uninterrupted, while I/O bound processes will be unable to maintain I/O activity at a high level. When an I/O-bound process eventually gets the processor, it will soon incur an I/O wait, possibly allowing a CPU- bound process to re-start. Thus the utilization of the I/O devices will be poor.

FCFS is rarely used on its own but is often employed in conjunction with other methods.

Merits/Demerits

- it favors long jobs over short ones
- if a CPU-bound process gets the processor, it will run for relatively long period uninterrupted while an I/O-bound process will unable to maintain high I/O activity.
- FCFS is rarely used on its own but is often employed in conjunction with other methods.
- It is not useful in a time sharing environment because it cannot guarantee a good response to time

(b) Shortest Job First (SJF)

It is also known as shortest job next (SJN), this non-preemptive policy reveres the bias against short jobs found in the FCFS scheme by selecting the process from the READY queue which has the shortest estimated run time. A job of expected short duration will effectively jump past longer jobs in the queue. It is essentially a priority scheme where the priority is the inverse of the estimate run time. If we use this scheme on the jobs in table 1 above, we get a rather different picture as shown in table 2 below.

Table 2 illustrate SJF policy

	(a)	**(b)**	**(c)**
Job	**Est. Runtime**	**Waiting**	**Ratio b/a**
3	1	0	0
1	2	1	0.5
4	3	3	1.0
5	50	6	0.1
2	60	56	0.9

This appears to be much more equitable, with no process having a large wait to run-time ratio. The example does not reveal difficultyin the scheme, however, a long job in the queue may be delayed indefinitely by a succession of smaller jobs arriving in the queue. In the example of table 2, it is assumed that the job list is constant, but, in practice, before time 3 is reached when job 5 is due to start, another job of length, say, 10 minutes could arrive and be placed ahead of job5. This queue jumping effect could recur many times, effectively preventing job 5 from starting at all; this situation is known as starvation.

SJF is more applicable to batch working since it requires that an estimate of time be available, which could be supplied in the job control language (JCL) commands for the job. It is possible for the operating system to derive a substitute measure for interactive processes by computing an average of run durations (i.e. periods when the process is in the RUNNING state) over a period of time. This measures likely to indicate the amount of time the process is likely to use when in next gets the processor.

Merits/Demerits

- A long job may be delayed indefinitely by a succession of smaller jobs arriving in the queue.
- It is more applicable to batch jobs. Since this method requires that an estimated of run time be available which could be supplied in the JCL commands for the job.

(c) Shortest Remaining Time (SRT)

SRT is a preemptive version of SJF. At the time of dispatching, the shortest queued process, say job A, will be started; however, if during running of this process, another job arrives whose run-time is shorter than job A's remaining run-time then job A will be preempted to allow the new job to start. SRT favors short jobs even more than SJF, since a currently running long job could be ousted by a new shorter one. The danger of starvation of long job also exists in this scheme. Implementation of SRT requires an estimate of total run-time and measurement of elapsed run-time.

Merits/Demerits

- It favors short jobs better than SJF. Since a currently running long job could be ousted (put cut) by a new shorter one.
- The danger of starvation of long jobs also exists in this scheme. The implementation of SJF requires an estimate of total run time and measurement of elapsed run time.

(d) Highest Response Ratio Next (HRN)

This scheme is derived from the SJF method, to reduce SJF's bias against long jobs and to avoid the danger of starvation. In effect, HRN derives a dynamic priority value based on the estimated run time and the incurred waiting time. The priority for each process is calculated from the formula:

$$\textbf{Priority, P} = \frac{\text{Time waiting + run time}}{\text{run time}}$$

The process with the highest priority value will be selected for running. When process first appears in the READY queue, the 'time waiting' will be zero and hence P will be equal to 1 for all processes. After a short period of waiting however, the shorter jobs will be favored; e.g. consider two jobs A and B, with run times of 10 and 50 minutes respectively. After each has waited 5 minutes, their respective priorities are:

A: $P = \frac{(5 + 10)}{10} = 1.5$

B: $P = \frac{(5 + 50)}{50} = 1.1$

On this basis, the shorter job A selected. Note, however, if A had just started (wait time = 0)B would be chosen in preference to A. as time passes, the wait time will become more significant. If B had been waiting for, say, 30 minutes then its priority would be:

B: $P = \frac{(30 + 50)}{50} = 1.6$

This technique guarantees that a job cannot be starved, since ultimately the effect of the wait time in the numerator of the priority expression will predominate over shorter jobs with a smaller wait time.

Merits/Demerits

A job cannot be staved since ultimately the effect of the wait in the numerator of the priority expression will predominate over short jobs with a smaller wait time.

(e) Round Robin (RR)

In the Round Robin scheme, a process is selected for running from the READY queue in FIFO sequence. However, if the process runs beyond a certain fixed length of time, called the time quantum, it is interrupted and returned to the READY queue. In other words, each active process is given a 'time slice' in rotation. The RR technique is illustrated in figure 6 below:

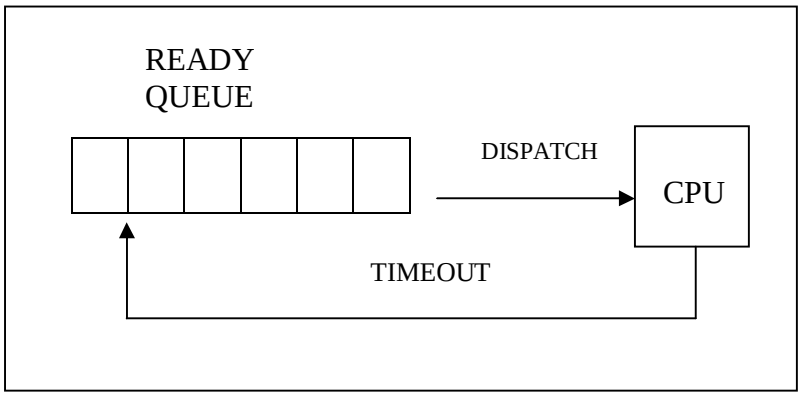


Figure 3: Round Robin Scheduling

The timing required by this scheme is obtained by using a hardware timer which generates an interrupt at pre-set intervals. RR is effective in timesharing environments, where it is desirable to provide an acceptable response time for every user and where the processing demands of each user will often be relatively low and sporadic. The RR scheme is preemptive, but preemption occurs only by expiry of the time quantum.

By its nature, RR incurs a significant overhead since each time quantum brings a context switch. This raises the question of how long the time quantum should be. As is often the case, this decision has to be compromise between conflicting requirements. On the one hand, the quantum should be as large as possible to minimize the overheads of context switches, while on the other hand, it should not be so long as to reduce the users' response times. It is worth noting that if the quantum size is increased sufficiently, the scheduling approaches FCFS. In the FCFS scheme, context switches will take place when the current process cannot continue due to issuing an I/O request. If the time quantum in the RR scheme is comparable in length to the average time between I/O requests, then the two schemes will be performing in a similar fashion. Ideally, in an interactive environment, most processes will be I/O bound, so that they will be incurring I/O waits, and hence yielding the processor, before expiry of the time quantum. This indicates the general order of size for the time quantum, but depends in a somewhat unpredictable way on the particular loading and job mix on the system. In practice, the quantum is typically of the order of 10 to 20 milliseconds.

Merits/Demerits

i. It is effective in time sharing environment, where it is desirable to produce an acceptable response time for every user and where processing demands each user will often be relatively low and sporadic.

ii. It incurs a significant overhead since each time slice brings a context switch. However, the overhead is kept low by efficient context switching mechanism and providing adequate storage for the processes to reside in RAM at the same time.

In practices, the quantum is typically of the order of 10-20 milliseconds.

(f) Multi-Level Feedback Queues (MFQ)

The policies described above are relatively limited in their ability to cope with wide variability in the behavior of the processes in a system. Ideally, a scheduling scheme should give priority to short jobs and favor I/O-bound jobs, while otherwise being 'fair' to all processes. The MFQ scheme is an attempt to provide a more adaptive policy which will treat processes on the basis of their past behavior.

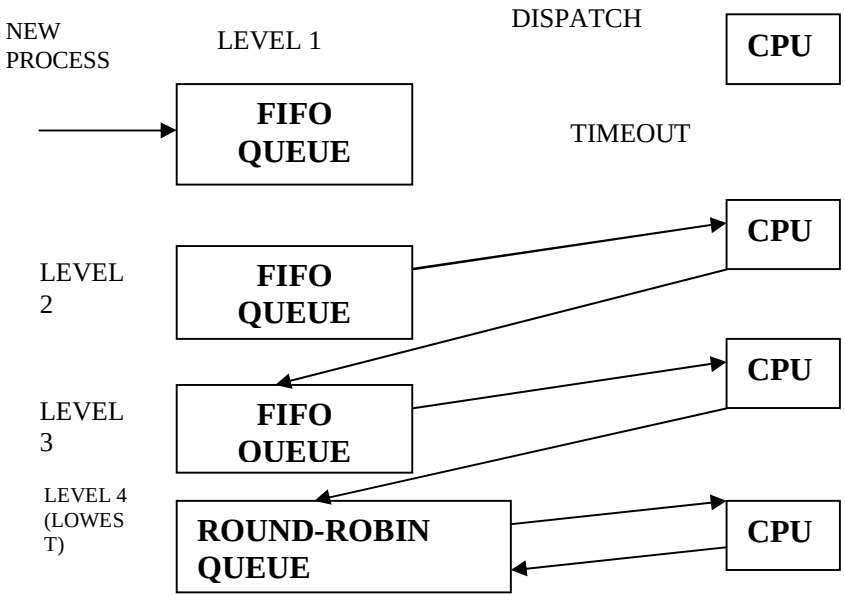


Figure 4: Multi-level Feedback Queues

Figure 4 above show a typical set up for a MFQ system. It consists of a number of separate queues of entries which represent active processes. Each queue represents different priority, with the top queue being highest priority and lower queues successively lower priorities. Within each queue, the queued processes are treated in a FIFO fashion, with a time quantum being applied to limit the amount of processor time given to each process. Processes in a lower level queue are allocated the processor unless all the queues above that queue are empty. A new process enters the system at the end of the top queue and will eventually work its way to the front and be dispatched. If it uses up its time quantum, it moves the end of the queue in the next lower level, with the exception of the lowest queue where a round robin scheme applies (i.e. it simply moves to the end of that queue).if it relinquishes the processor due to a wait condition, the process leaves the queuing system. As a process uses more and more CPU time, it will migrate down the levels, thus obtaining a reducing level of access.

This arrangement militates to some extent against long processes and starvation is a possibility. Various modifications exist to the basis scheme which attempt to meet some of its problems. One option is to use increasing size of time quantum on lower levels, so that when a lower queue process does get a chance it holds the processor for a longer period. Also, some systems promote processes to a higher level queue when they have spent a certain amount of time in a queue without being serviced.

Merit/Demerits

- The scheme does not favor long jobs.
- Starvation is a possibility.

PROCESSES IN WINDOWS

Current versions of window systems have very elaborate process management facilities. The major features are:

- every process is created as a single executing thread; the process can create additional threads
- the scheduler operates over all threads.
- In contrast to earlier Windows versions, each process has its own virtual address space, so that one process cannot affect the memory space of another.
- Like UNIX, OS/2 creates processes in a hierarchical fashion. When a process spawns another process, the later is considered a 'child' of the former and it inherits its environment. In contrast, Windows 95 and NT maintain no formal parent-child relationship between processes although the environment is copied.

Dynamic Link Libraries

The current and earlier versions of window systems use Dynamic Link Libraries (usually called simply DLLs). These are special form of executable EXE program file, their essential features being that the code routines within the DLL are only linked into the application program at run-time and that the DLL code is shareable.

Tutorial Questions

1. What would be the effect of the system running too many I/O intensive jobs

   Answer

   The jobs be sustained by relatively little processor activity and the processor will be under-utilized.

2. Distinguish between preemptive, non-preemptive and cooperative scheduling methods.

   Answer

   In non-preemptive methods, the running process can only 'loss' the processor voluntarily, by terminating or by incurring an I/O wait. In a preemptive method, the low level scheduler can re-allocate the processor some other basis; e.g. Timeout or arrival of a high priority process. In cooperative scheduling a process must voluntarily relinquish control. The operating system has no control over this.

3. What would be the effect, using the FCFS scheme, if the running process got stuck in an infinite CPU loop?

Answer

The process once running would dominate the processor. Not interrupts may occur (from I/O devices, say) and will be serviced, but the process will be re-started thereafter. The process could be stopped by kill command.

MODULE FOUR
MEMORY MANAGEMENT

UNIT 1
REALMEMORY MANGEMENT

Introduction

The term 'memory' and 'storage' have been used interchangeable in the literature. There are two major types of storage, thus:

(a) primary storage (RAM, MAIN MEMORY)
(b) Secondary storage (BACKING STORAGE)

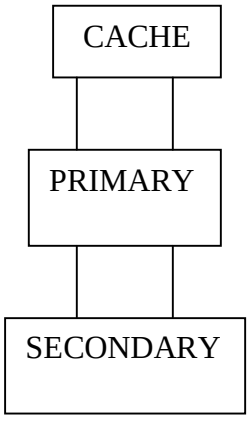
The main memory is essential within the computer for various reasons:

- to enable process exist,
- to store instructions which are interrupt by the processor, and
- a work space and transient storage medium for various kinds of data9objects) such as O.S. Data (process table, file description tables etc), user program code and data, video storage space etc.

The main memory is made up of semi-conductor DIODES. It is limited in a size because it is expensive but it's faster than secondary storage:
The secondary storage is less expensive and therefore has a larger storage but is slower than RAM, e.g. Diskette, hard disk, magnetic tape, magnetic disk.
Main memory is part of the CPU but secondary storage is an external peripheral. Programs and data that are not needed immediately are kept in the secondary storage until when needed and the brought into the main memory for execution or reference.
Thus, RAM is access more quickly than backing storage.

In systems with several level of storage, a great deal of shuffling goes on which programs and data are moved back and front between various level. The shuffling consumes system resources such as CPU time that would otherwise be used for reproduction.

Diagram below



Cache storage was introduce in 1960's. It's is a high speed storage that is faster than primary storage. It's is also extremely expensive when compared with RAM and therefore only relatively small cache are used. Instruction that need to process many times over 1000 or more are moved to cache memory. Historically, the number of different memory management technique have been used, and in the processes of evolution has been superseded by superior methods. To some extent, history has repeated itself in that many older techniques were resurrected for application in microcomputer which appears in the late 1970's.

The principal problem to be handle by O.S. Memory manager are:

- to provide the memory space to enable several processes to executed at the same time.
- to provide satisfactory level of performance (i.e. process execution speed) for the system user's.
- to protects each process from each other.
- where desired, to enable sharing of memory space between processes.
- to make the addressing of memory space as transparent as possible for the programmer.

Note that although several processes may be active and hence occupying memory space, at instant of time only one instruction is been executed.

Process Loading and Swapping

In order to execute a process, the program code has to be transferred from secondary storage to primary storage; this activity is called process loading.

In some system, the reverse operation is also possible i.e. the transfer of a process from primary store to secondary store. This is done in order to accommodate and consequently execute another process. The overall effect being the swapping of two processes. The transfer to secondary storage must preserve the entire state of the process including current data value and status of all registers.

Swapping systems do not usually employ the standard file management facilities. This is because of usual block orientation and general space management overheads which makes standard file the swapping activity and special software manages the data transfer at high speed. It is performed in order to allow another process to be started.

Storage Allocation

There are two broad types of processes

- contiguous storage allocation
- non-contiguous storage allocation

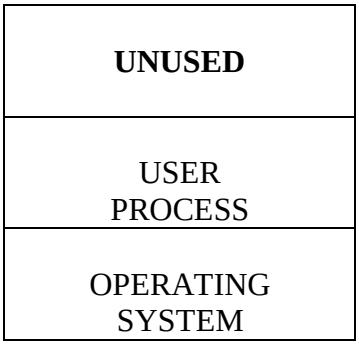
Earliest computing systems required contiguous storage allocation each program had to occupy a single contiguous block of storage locations.

In non-contiguous storage allocation, a program is divided into several blocks or segment that may be placed throughout main storage in pieces not necessarily adjacent to one another.

Memory Allocation Methods

1. **The single user (process) system**

It was introduced in the zeroth and first generation of O.S. It was a simple memory management technique that allowed only one user at a time to work on the machines. A single-user's process was loaded into the free space area of the memory and all machine resources were at his disposal i.e. the process to be executed is loaded into the free space of the memory; in general, a part of the memory space will be unused.



Merits/Demerits

- Programs were limited to the size of the memory
- Computing resources were generally wasted.

Protection facilities

A single boundary register in the CPU contained the highest numbered of instruction used in the O.S. each time a user program referred to a storage address, the boundary register was checked to ascertain that user program was not about to destroy the O.S.If the user tried to enter into the O.S, the instruction will be intercept and the job terminates with an appropriate error message.

However, the user needs to access the O.S., for services such as I/O for which the user is given a specific instruction (called the SVC inst) with which to request for services.

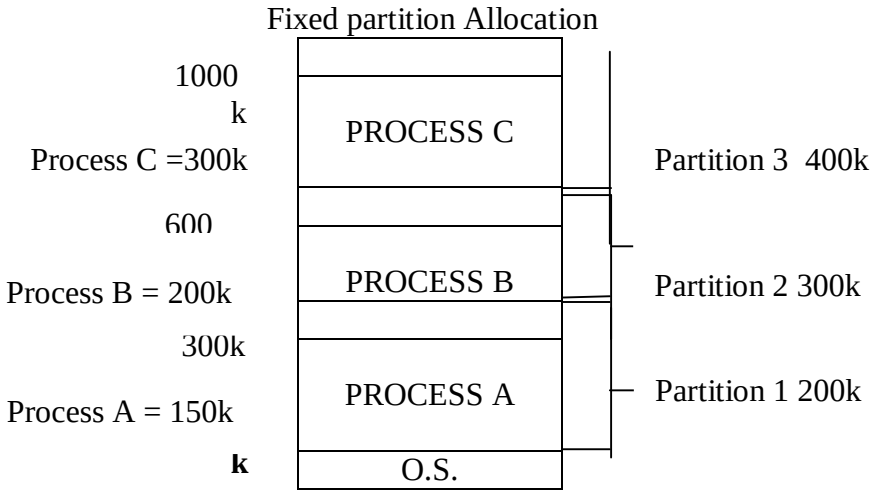
Usefulness

- In simple systems such as games computer.
- Early MSDOS operated this way.

2. **Fixed partition memory**

This scheme was introduced in the second generation of O.S. to improve utilization of computer resources. The memory was divided into a number of separate fixed areas, each of which could hold a process.

The memory is been partition into block the process where we have used part hole called internal fragmentation.



The memory above is shown consisting of three areas of sizes 200k, 300k and 400k respectively, each of which hold a process.
In practice, the number of partition s will be controlled by the system manager. This control would depend on:

- The amount of memory available and
- The size of processes to be run.

The partitions would be set up with a range of partition size, so that a mixture of large and small processes could be accommodated. Each partition would typically contain unused space which might be large when put together. The occurrence of wasted space is referred to as 'internal fragmentation'. The word 'internal' refers to wastage within the space allocate to a process.

Several processes resides in the memory and compete for system resources such as a job currently waiting for I/O will yield the CPU to another job that is ready to perform calculations. Thus both I/O and CPU operations can occurs simultaneously. This greatly increased system throughput and CPU/I/O utilization.

Merits/Demerits

- Utilization of CPU and I/O devices are greatly improved.
- System throughput (work/unit time) increases.
- Multiprogramming required more storage space than a single user system.
- The fixed partition sizes can prevent a process from running due to the unavailability of a sufficient partition sizes.
- Internal fragmentation wastes space, which collectively could accommodate another process.

Protection Facilities

Several bound (limit registers are employed; one for each process. Each limit register contains the address of the low and high boundaries of each fixed partition.

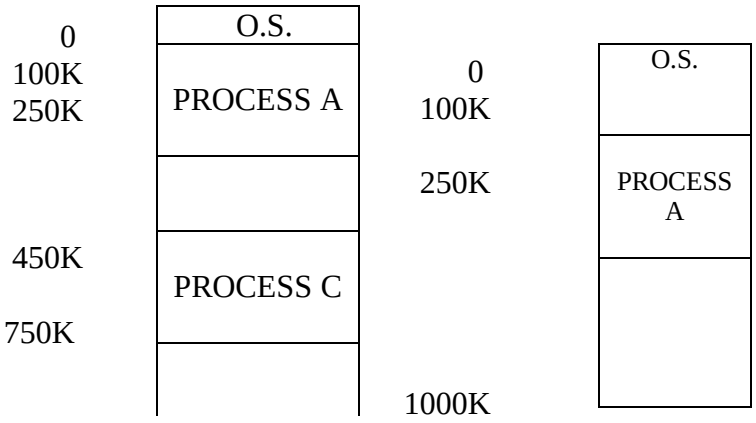
Usefulness

- Employed in early multiprogramming computer systems such as the IBM 360 machine.

3. Variable Partition Memory

O.S designers observing the problems of fixed partition memory decided that an obvious implementation would be to allow jobs (process) occupy as much as they need (short of the whole memory). No fixed boundaries were observed; instead each job (process) is given as much storage as required.
Processes are loaded into consecutive areas until the memory filled up, or, more likely, the remaining space is too small to accommodate another process.

When the process terminates, the space it occupied is freed and becomes available for the loading of a new process. As processes terminate and space is freed, the free spaces appear as a series of 'hole' between the active memory area. The O.S must attempt to load an incoming process into a space large enough to accommodate it. It can often happen when a process cannot be started because none of the holes is large enough though the total free space is more than the required size. This situation is illustrated in the figure below in which process B and D have terminated. Distribution of the free memory space is termed 'external fragmentation'. The term 'external' refers to space outside any process allocation.



(a) Fragmentation when process B and D terminates.
(b) Coalescing when process C terminates.

It will frequently happen that a process adjacent to one or more holes will terminate and free its space allocation. This results in two or three adjacent holes, which can then be viewed and utilized as a single hole, as indicated in the fig. above. The process of merging adjacent holes to form a single larger hole is called coalescing and is a signification factor in maintaining fragmentation within usable limit.

Merit/Demerits

- External fragmentation reduces the utilization of the memory.
- Variable number of processes can be handled at any time.

Usefulness

- It has been successfully used in many computer systems.

Storage Placement Policies

When new processes have to be loaded using the variable partition scheme, it is necessary to try to select the 'best' locations in which to place them i.e. to select the series of hole which will, overall, provide the maximum overall throughput for the system bearing in mind that an inefficient allocation can delay the loading of a process. An algorithm used for this purpose is termed, a placement policy. A number of such policies have been device and is described below:

a. **Best fit policy**
An incoming process is placed in a hole which it fits most tightly and leaves the smallest amount of unused space.

b. **First fit policy**
An incoming process is placed in first available hole which can accommodate it.

c. **Worst fit policy**
An incoming process is placed in the hole which leaves the maximum amount of unused space i.e. which logically must be current largest hole. Best fit is intuitively appealing, since it appear to minimize wasting space. By contrast, worst fit does not look promising. However, worst fit could possibility leaves a hole large enough to accommodate another process while best fit is trying to minimizes the unused space it could create a hole too small to be useful.
In practice, the best fit and the first fit generally prove to be the most effective. The figure below shows a typical situation. A new process is to be loaded. The effect of the choice using each of the 3 algorithms can be seen.

OS
PROCESS A
200K
PROCESS B
300K
PROCESS C
200K
PROCESS D
400K

Overhead

Any memory management schemes will have inner some operating overhead. In the case of the variable partition scheme, the system must keep tract of the position and size of each hole, taking account of the effect of coalescing

The first fit scheme would simply select the first item on the list. While the best fit and the worst fit would need to scan the full list before deciding.

4. Variable Partition Allocation with Compaction

The fragmentation problem encountered in the previous method can be tackled by physically moving resident processes in the memory in order to close up the holes and hence bring the free space into a single large block. The process is referred to as 'compaction'. Sometimes it is referred to as 'bugging' the memory or 'garbage collection'.

Overhead

It is clear that compaction has desired effort of making the free space more usable by incoming processes but this is achieved at the expense of large scale memory of current processes. All processes would be suspended while the reshuffle takes place, with attendant updating of process context information such as load address. Such activity would be a major overhead in any system.

Merits/Demerits

- Total free space is more usable for incoming processes.
- It consumes system resources that could otherwise be used productivity.
- The system must stop everything whenperforming the compaction. This can result in response times for interactive users and could be devasting in real time systems.
- Compaction involves relocating processes in storage.
- Reduced fragmentation.
- With a rapidly changing process mix, it is necessary to compact frequently. The consumed system resources might not satisfy the benefit of compaction.

Protection Facilities

Several limit registers are employed one for each process. Each limit register is established when the process is dispatched and variable partition.

In practice, the compaction scheme has been used due to the fact that its overhead and added complexity tend to minimizes its advantage over the non-compacted scheme.

5. Simple Paging

In a page system, each process is divided into a number of fixed sizes 'chuck' called pages, typically 4kb in length. The memory space is also viewed as a set of page frames of the same size.
The loading process now involves transferring each process page to some memory page frame.

The figure above shows that there are 3 free pages in memory which are available for suppose that process B terminates and releases its allocation of pages, giving us the situation on it figure below.

We now have two disconnected regions of free pages. However, this is not a program in a paging system because the allocation is a page by page basis. The pages of processes held in memory frames do not be contiguous or even in the correct order.

ASSUME THAT TWO processes require to be loaded. Process D needs three pages and process E, for pages. These are allocated to any free memory pages (fig. b). Paging reduces the problem of fragmented free space, since a process can be distributed over a number of separates holes. AFTER A PERIOD OF OPERATION, the pages of active processes could become extensively intermixed; producing (fig. c).

Merits/Demerits

1. Paging alleviative the problem of fragmented free space since a process can be distributed over a number of operate holes.
2. Space utilization and consequently the system throughput are improved.

Usefulness

1. It is seldom used in practice, virtual systems preferred. This is because having got to the level of sophistication required by these schemes, superior systems can be obtained with relatively little effort.

6. Simple Segmentation

Paging achieves its objective by subdividing a process into a number of fixed sized chucks. Segmentation subdivided a process into a lot of variable length chuck called segments.
Segmentation is similar to the variable partition allocation method except that a process can be loaded in several partitions segments solving the problem of allocation unto available free space. Segments can be of any length, up to a maximum value determined by the design of the system. They can be positioned independently in the memory therefore provides more efficient utilization of free areas.

Under a paging system, the process subdivisions are physical entities not related to the logical structure of the process in any way. However the segment is a segmentation scheme correspond to the logical divisions of the process and defined explicitly by the programmer.

Typically, the segments defined by a programmer would reflect the modular structure of the process: e.g. data in one segment, each subroutine or a group of related subroutines in a number code segments. The programmer must be aware of the maximum segment size during design of the segments.

Merits/Demerits

1. Full process is stored in memory.
2. Full defined scheme
3. Some external fragmentation is possible.

UNIT 2

VIRTUAL MEMORY MANAGEMENT

Virtual memory is an extension of the main memory. It is a storage space that is not used within the CPU but is treated as such. The extended storage space is on devices such as diskettes, hard disks, tapes, etc. this is needed to meet the user's various needs because primary storage is expensive.
The benefits of virtual memory are obtained at some cost in system complexity. This is explain further below:

Mechanisms of Virtual Memory

- Virtual paging
- Virtual segmented systems
- Combined paged and segmented systems
- Virtual machines

1. Virtual Paging

When a new process is initiated, the system loader must load at least one page from secondary storage into real memory; i.e. the page containing the execution start point for the process. This process is known as PAGE IN. when execution of the process commences execution will processed through subsequent instructions beyond the start point. This can continue as long as memory reference generated by this page are also within the same page. However, after some time, the references (addresses) generated will refer to page outside the real memory; virtual address is created. An interrupt is generated indicating that the requested page is not in memory.

Page Fault: a signal demanding for the requested page which is not in RAM: hence, the term demand paging is used for this technique. The system loader will try to oblige by loading the request into a free memory page frame (PAGE IN) and execution can proceed. A series of pages faults are generated this way are accumulated in real memory. This subset is referred to as the resident set of the process. When the process terminates, the O.S. releases all pages belonging to the process making them available for other processes.
Usually, there will be many processes competing for real memory space. Consequently, the available real memory will become full of pages belonging to these processes. If a page fault then occurs and removed the currently loaded page from the page frame to the data set (secondary storage), it is referred to as PAGE OUT and this event is called the page replacement.

Merits/Demerits

1. Minimal space wastage.
2. Large virtual address space
3. Page replacement scheme required.
4. Protection facilities
   A hardware register beholds the page number for the current process

Sharing

While it is possible for 2 or more processes to share real memory pages, this is rarely attempted. Since the contents of the pages is generally unknown. One of the merits of paging system is that it is largely transparent to its programmer.

Page Replacement Policy

When a new page required to be brought into the memory, it may be necessary to remove one currently in residence. When the page is remove from memory, it is be necessary to write it back to secondary storage if it is 'dirty i.e. it has modified while in memory. The M bit in the page table is used to indicate dirty pages.

The algorithm to chosen which page will be replaced are referred to as page replacement policies:

- Least recently used (LRU) :replace the page which has least recently been used
- Not recently used (NRU):replace the page which has been used least frequently during some immediately preceding time internal
- First-in-first-out (FIFO) : replace the page which has been resident longest.

The Least Recently Used (LRU) Policy selects for replacement of page whose time since last reference is greatest. This would notionally require that a time stamp recordings is made for a page frame at the time of each reference. The selected page would then have the oldest time stamp. The overhead of maintaining such a value would be considerable, as the time taken to find out the oldest value.

In practice, a related but simple policy is used NRU. Each page frame has associated with it, a page 'page referenced' bit; at intervals, the operating system resets all of these bits to zero. Subsequent reference to a page will set its page referenced bit top 1, indicating that this page has been used during the current interval. The NRU policy simply selects for replacement any page with a page referenced bit of zero.

The CPU two has a storage protection key. While the storage protection key in the CPU is say, 2 corresponding to user B.

User B's program may refer only to other blocks of storage with the same storage protection key of 2, these key are strictly under control of the O.S.

The First-in-First-out (FIFO) method selects for removal of the page which has been resident in memory for the longest time. The motivation for this approach is the assumption that such a page is likely to be no longer in use.

2. Virtual Segmented System

In virtual segment scheme, the segments of a process are loaded independently, and hence may be stored in any available memory positions or indeed may not be in memory at all. Virtual segmented systems have a number of advantages among which are:

- Facilities use of dynamic memory; if the application requests additional memory space to 'grow' a data structure for example, the segment can be granted more space and if necessary moved to another location in memory.
- Segmentation facilities sharing of code and data between processes.
- Logical structure of the process is reflected in the physical structure by reinforces the locality principle.

Sharing Facilities

Since segments are logical entities holding programmer defined procedures and data object, précise control of access and sharing is possible. Segments usually have access to attributes which are specified in the segment descriptor.

Possible attributes are:

Read access: the segment can be read.

Write access: the segment can be modified

Execute access: the segment may be executed.
Append access: the segment may have data added to the end.

It is useful in systems:

- Where several users require the same software e.g. a computer or text editor. These can be loaded as a sharable segment and accessed by each on line.
- Window environment: this is because they offer the use of shared libraries which contain a number of routines necessary for commonly required functions e.g. window management.

3. Combined Paging and Segmentation

The respective merits of paging and segmentation can be combined into a paged segmented system. Segmentation is 'visible' to the programmer and meets many of his requirements. Paging provides transparent memory management which eliminates wasted space and facilities complex optimum action techniques.

A combined paging/segmentation system appears to the programmer to be simply a segmented system. However, the segments defined are subdivided at the physical level into a number of fixed length pages. A segment smaller than a page will use a whole page and typically one page within a segment will be partially unfilled.

Merits

A page segmented systems is quite complex but it provides a powerful environment for modern computers, giving the programmer control over process structure while efficiently managing memory space e.g. OS/2 Microsoft windows. IBM MVS/ESA.

4. Virtual Machines (VM)

A virtual machines makes a single real machines appear to users as a several real machines each with its own processor memory and O/I devices system. It may appear to the computer or may have the characteristics of an imaginary computer.

One of the earliest virtual machines systems was IBM, VM, O.S on IBM 370 machines. A single IBM 370 machines could act as several 'smaller' 370's, each complete with its own processor and memory i.e. each user appears to have all the peripherals processor and memory to himself whereas he is sharing the same CPU with other users. It is a top range O.S and thus very powerful.

Usefulness

1. Provides continuity in the use of earlier machines or O.S. IBM VM allowed users to continue to run other application systems based or the earlier IBM 360 processes such as DOS VS while developing new applications using the DOS / VSE(virtual storage excention) and DOS/VMS (virtual machines storage). O.S. it was also used in ICL VME systems.
2. The enhanced mode of the Intel 80386 processor can emulate multiple 8086 processor.

UNIT 3
MS DOS MEMORY MANAGEMENT

MS DOS evolved from a humble beginning and now incorporates many features not envisaging the beginning of its history. As a consequence, it is in many respects untidy due to fundamental limitation of its early design.

Significant Characteristics of MS-DOS are

- It is designed around Intel 86 series of processors.
- It uses a basic memory format which is fixed in its design since early versions

The original Intel 8080 processor on which MS DOS was designed used a simple 16 bit address scheme, giving an addressable range of only 64kb. In order to improve on this the newer Intel 8086 and 8088 chips were later employed which were able to address up to IBM while presenting the same basic 16 bit address scheme, thus being compatible with the older system. This was achieve by the introduction of segment registers to the chip architecture. The Intel 8086 and 8088 processors use a set of 4 segment registers, each of 16 bits, which provide a base address for the addressing of separate segments of the active process. The segment registers are.

CS **Code Segment**: points to segment containing execution codes.
DS **Data Segment**: points to segment containing process data
SS **Stack Segment:** points to segment containing process stack
ES **Extra Segment:** points to segment used for application specific purpose.

The basic memory model of MS DOS was that a process consists of 4 segments, locatable independently within the available address space. The 16 bits of the segment registers provides only IB addressability to obtain IMB which needs a 20 bits (2 =mb), the 16 bits segment register value is shifted left 4 bit, effectively multiplying it, by 16. Hence, the effective base addresses can only adopt values at intervals of 16 called paragraphs, but these values extend up to 1Mbyte. E.g.

Address value from instruction;	0000 0000 0000 1111
Segment Register Content:	0000 0000 0000 1011
Left shifted register content:	0000 0000 0000 1011 0000
Effective address (20 bit):	0000 0000 0000 1011 1111

The maximum segment address is 1111 1111 1111 1111 0000 (Hex FFFO) which is 16 bytes less than 1 Mbyte. Based on this addressing scheme, MS DOS was mapped out within 1Mb space.
Note that the available user programs space called the transient program area (TPA) has 640kb less space for O.S e.t.c. prior to MS DOS V5; this left about 560kb, with the introduction of V5 much of the OS and the devices drivers have repositioned above the 640kb and consequently about 600kb are available for user program.

Upper Memory Area
The 88k of space above the conventional memory area of 640kb is called the upper memory area (UMA)
The UMA is not considered as part of the total memory of your computer because program cannot store information in this area. This area is normally reserved for running our systems hardware such as the monitor. Information can mapped or copied from another memory to part of the UMA left by the system. These unused parts are called upper memory blocks.

Usefulness
For running programs that use expanded memory.

Overlaying
This technique was available in older systems. MS DOS reintroduced it as a means of overcoming the 640kb limit. The essence of overlaying is that the object program is constructed as a number of separate modules called overlays which can be loaded individually and selectively into the same memory area. MS DOS provides a system call enable a program to load another object file, execute it and then regain control.

The object program consists of a roof section which is always in memory and two or more loadable overlays. The whole system has to be managed at the program level and care must be taken to avoid frequent use of overlays.

Merits/Demerits
1. To reduce the code size by putting it in used frequently into separate overlays e.g. initialization and error routines which are rarely required
2. Useful in splitting large object program with small sizes of data
3. Not useful in applications and large number of data memory such as spreadsheet and programs

Extended and Expanded Memory
Intel 8086/8008 processors were followed by the 80286, 80386 and 80486 processors which enables the processing of memory far beyond 1Mb. However, the design of MS DOS was locked into the 1Mb unit at least as far as MS DOS process was concerned.
Systems designers soon found ways of utilizing additional memory.

1. Extended Memory (XMS)
It refers to memory installed above the 1Mb man but it is not accessible directly by MS DOS programs. This space is used directly for other purpose contributing to the performance of the system such as RAM DISK (makes use of a portion of memory as if it were a harddisk), disk caching (i.e. programs reduce the time computer spends reading data from the harddisk)

Merits
- Fast and efficient for programs that uses it.
- A program called the XMS manager then HIHEM.SYS makes it easier for programs to use XMS.

Windows Memory Management
There are, of course, several version of Windows and related systems and each of these has different memory management methods. Earlier versions, like MS-DOS, were constrained by the limited architecture of the processors used in PCs. Windows 3.1 actually provided three modes of operation,

the choice of these being dependent on the processor available and other factors. These modes are described belows:

Real Mode: In this mode, Windows use only the basic 640 Kbytes of main memory accessible to MS-DOS, and can run using an Intel 8086 or better.

Standard Mode: This is the normal Windows 3 mode; it allows use of extended memory (i.e. main memory above 1 MByte).

Enhanced Mode: This mode utilizes the virtual memory capabilities of the Intel 80386 processors or better. In addition to the processor, enhanced mode required at least 2 Mbytes of memory. It allows multi-tasking of non-Windows programs.

Although the enhanced mode utilized virtual memory techniques, all the running processes share the same address space. To control fragmentation Windows is capable of moving blocks of code and data within memory. Memory addressing is performed using 16-bit segmented addressing; i.e. addresses consist of the contents of a segment register plus a 16-bit displacement as in MS-DOS. The application programmer's interface to this memory system is termed the Win16 APL.
Windows also uses DLLs (dynamic link libraries) to conserve memory space; DLLs are executable program files containing shareable code that is linked with an application program at run time. Windows itself consists of a number of DLLs, and 'common' code such as device drivers are implemented as DLLs. This technique reduces the demands on memory space since the same code is used by several running processes.

With the introduction of Windows NT and 95 and OS/2, memory management has improved dramatically. These systems use 32-bit, flat memory addressing, providing a massive 4 Gbytes of address space without the need to use segmented addressing. This memory model can be managed using the Win32 API.

MODULE FIVE

INPUT – OUTPUT

Organization of I/O software and hardware

The Input-Output system constitutes one of the four pillars on which a computer stands, the others being the processor, the main memory and the file system. It is generally viewed as being the least satisfactory member of this quarter because of its relative slowness and lack of consistency. These characteristics are consequence of the nature of I/O devices and their role in trying to provide communication between the microsecond domains of the computer. The range of I/O devices and the variability of their inherent nature, speed, specific design, etc, make it difficult for the operating system to handle them with any generality.

Characteristics of I/O Devices

Characteristic	Examples
Data Rate	Disk: 2 Mbytes/sec Keyboard: 10-15 bytes/sec.
Unit Rate	Disk: blocks of 512, 1024 etc bytes Screen: single characters
Operations	Disk: read, write, seek etc. Printer: write, move paper, select font
Error Conditions	Disk: read errors Printer: paper out

Objectives of I/O System

Efficiency

Perhaps the most significant characteristic of the I/O system is the speed disparity between it and the processor and memory. Because I/O devices inevitable involve mechanical operations, they cannot compete with the microsecond or nanosecond speed of the processor and memory. The design of the I/O system largely reflects the need to minimize the problems caused by this disparity. Of central importance is the need to make the I/O devices – in other words to operate them at maximum efficiency.

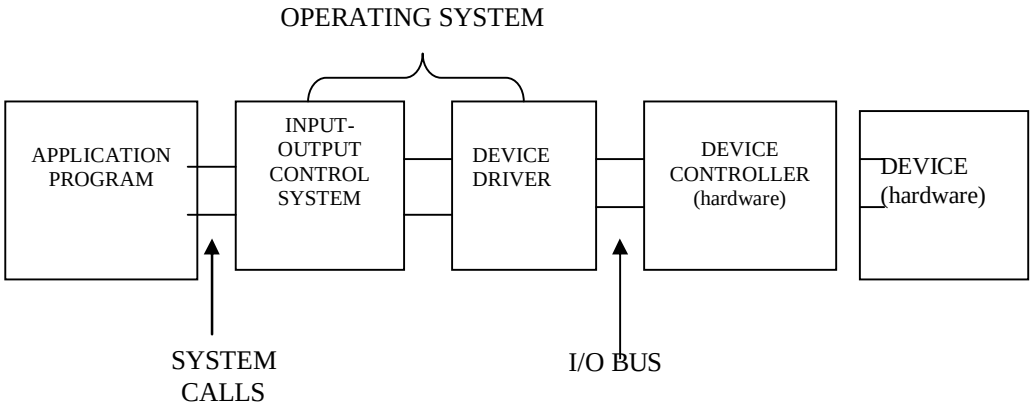
Generality and device independence

I/O devices are typically quite complex mechanically and electronically. Much of this complexity is in the realm of the electronic engineer and is of no interest to the user or even the programmer of the computer. The average user, for example, is not aware of the complexities of positioning the heads on a disk drive, reading the signal from the disk surface, waiting for the required sector to rotate into position etc. Users and programmers are distanced from these realities by layers of software which provide various levels of abstraction. In addition to these inherent complexities, devices also vary enormously in their design, mode of operation, interface protocols etc, producing yet more potential headaches for the operating system. Since it is not reasonable to expect the operating system to be cognizant of the detailed operation of every device with which it may have to communicate, a means of insulating it from these complexities must be found.

The principal objectives of the I/O systems are:

- To maximize the utilization of the processor
- To operate the devices at their maximum speed and
- To achieve device independence as far as possible.

Structure of I/O System



- **Application Program**
  Within the application, I/O activity is expressed in user-oriented terms, such as 'read record 21 from file xyz'. Such instructions in a high level language are translated into corresponding system calls which invoke operating system functions. Note that even at the system call level, the instructions are expressed in logical terms, largely independent of the device used.

- **Input-output control system (IOCS)**
  This term is used to denote that part of the operating system which deals with I/O related system calls. It performs initial processing and validation on the request and routes it to the appropriate handler at the next stage. It is also where the general management of I/O interrupts takes place.

- **Device drivers**
  A device driver is a software module which manages the communication with, and the control of, a specific I/O device, or type of device. It is the task of the device driver to convert the logical requests from the user into specific commands directed to the device itself. For example, a user request to write a record to a floppy disk would be realized within the device driver as a series of actions, such as checking for the presence of a disk in the drive, locating the file via the disk directory, positioning the heads etc.

- **Device controllers**
  A device controller is a hardware unit which is attached to the I/O bus of the computer and provides a hardware interface between the computer and the I/O device itself. Since it is connects to the computer bus, the controller is designed for the purposes of a particular computer system while at the same time it conforms in interface terms with the requirements of the actual I/O device.

- **Device**
  I/O devices are generally designed to be used in a wide range of different computer systems. For example, the same laser printer could be used on MS-DOS, APPle and UNIX systems.