Annai Women's College

(Arts and Science)

Affiliated to Bharathidasan University – Tiruchirapalli.

TNPL Road ,Punnam Chattram, Karur-639 136

Department of Computer Science, Computer Applications & IT

Faculty Name : Mrs.V.Bhuvaneswari,Msc., M.phil.,
Major : III Year - B.Sc(Computer Science)
Paper Code : 16SCCCS6
Title of the Paper : Operating Systems

INDEX

4	**Unit IV** **Device Management Types of Devices-Sequential Access Storage Media-Direct Access Storage DevicesMagnetic Disk Drive Access Times- Components of the I/O SubsystemCommunication among Devices-Management of I/O Requests**	**66 to 83**
5	**Unit: V** **File Management The File Manager -Interacting with the File Manager -File Organization - Physical Storage Allocation -Access Methods-Levels in a File Management System - Access Control Verification Module**	**84 to 99**

Unit I:

Evolution of operating systems- Functions – Different views of OS – Batch Processing, Multiprocessing, Time sharing OS – I / O programming concepts – Interrupt Structure & processing

==========

Operating System :

- An operating system is a program that acts as an interface between the software and the computer hardware.
- It is an integrated set of specialized programs used to manage overall resources and operations of the computer.
- It is a specialized software that controls and monitors the execution of all other programs that reside in the computer, including application programs and other system software.

Objectives of Operating System

The objectives of the operating system are –

- To make the computer system convenient to use in an efficient manner.
- To hide the details of the hardware resources from the users.
- To provide users a convenient interface to use the computer system.
- To act as an intermediary between the hardware and its users, making it easier for the users to access and use other resources.
- To manage the resources of a computer system.
- To keep track of who is using which resource, granting resource requests, and mediating conflicting requests from different programs and users.
- To provide efficient and fair sharing of resources among users and programs.

Evolution of Operating Systems

The evolution of operating systems is directly dependent to the development of computer systems and how users use them. Here is a quick tour of computing systems through the past fifty years in the timeline.

- 1945: ENIAC, Moore School of Engineering, University of Pennsylvania.

- 1949: EDSAC and EDVAC
- 1949 : BINAC - a successor to the ENIAC
- 1951: UNIVAC by Remington
- 1952: IBM 701
- 1956: The interrupt
- 1954-1957: FORTRAN was developed

Operating Systems by the late 1950s

By the late 1950s Operating systems were well improved and started supporting following usages :

- It was able to Single stream batch processing
- It could use Common, standardized, input/output routines for device access
- Program transition capabilities to reduce the overhead of starting a new job was added
- Error recovery to clean up after a job terminated abnormally was added.
- Job control languages that allowed users to specify the job definition and resource requirements were made possible.

Operating Systems In 1960s

- 1961: The dawn of minicomputers
- 1962 Compatible Time-Sharing System (CTSS) from MIT
- 1963 Burroughs Master Control Program (MCP) for the B5000 system
- 1964: IBM System/360
- 1960s: Disks become mainstream
- 1966: Minicomputers get cheaper, more powerful, and really useful
- 1967-1968: The mouse
- 1964 and onward: Multics
- 1969: The UNIX Time-Sharing System from Bell Telephone Laboratories

Supported OS Features by 1970s

- Multi User and Multi tasking was introduced.
- Dynamic address translation hardware and Virtual machines came into picture.
- Modular architectures came into existence.
- Personal, interactive systems came into existence.

Accomplishments after 1970

- 1971: Intel announces the microprocessor
- 1972: IBM comes out with VM: the Virtual Machine Operating System
- 1973: UNIX 4th Edition is published

- 1973: Ethernet
- 1974 The Personal Computer Age begins
- 1974: Gates and Allen wrote BASIC for the Altair
- 1976: Apple II
- August 12, 1981: IBM introduces the IBM PC
- 1983 Microsoft begins work on MS-Windows
- 1984 Apple Macintosh comes out
- 1990 Microsoft Windows 3.0 comes out
- 1991 GNU/Linux
- 1992 The first Windows virus comes out
- 1993 Windows NT
- 2007: iOS
- 2008: Android OS

And the research and development work still goes on, with new operating systems being developed and existing ones being improved to enhance the overall user experience while making operating systems fast and efficient like they have never been before.

Characteristics of Operating System

- **Memory Management** − Keeps track of the primary memory, i.e. what part of it is in use by whom, what part is not in use, etc. and allocates the memory when a process or program requests it.
- **Processor Management** − Allocates the processor (CPU) to a process and deallocates the processor when it is no longer required.
- **Device Management** − Keeps track of all the devices. This is also called I/O controller that decides which process gets the device, when, and for how much time.
- **File Management** − Allocates and de-allocates the resources and decides who gets the resources.
- **Security** − Prevents unauthorized access to programs and data by means of passwords and other similar techniques.
- **Job Accounting** − Keeps track of time and resources used by various jobs and/or users.
- **Control Over System Performance** − Records delays between the request for a service and from the system.

- **Interaction with the Operators** − Interaction may take place via the console of the computer in the form of instructions. The Operating System acknowledges the same, does the corresponding action, and informs the operation by a display screen.

- **Error-detecting Aids** − Production of dumps, traces, error messages, and other debugging and error-detecting methods.

- **Coordination Between Other Software and Users** − Coordination and assignment of compilers, interpreters, assemblers, and other software to the various users of the computer systems.

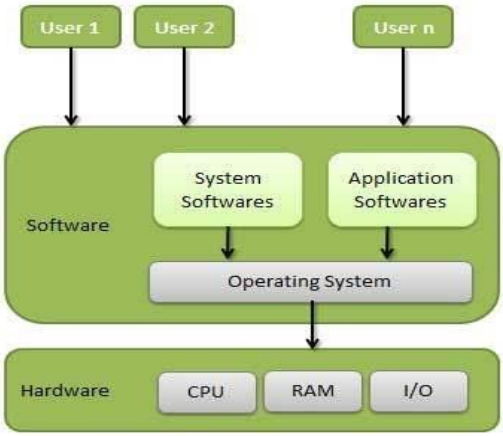
An Operating System (OS) is an interface between a computer user and computer hardware. An operating system is a software which performs all the basic tasks like file management, memory management, process management, handling input and output, and controlling peripheral devices such as disk drives and printers.

Some popular Operating Systems include Linux, Windows, OS X, VMS, OS/400, AIX, z/OS, etc.

Definition

An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.

Following are some of important functions of an operating System.



- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

Memory Management

Memory management refers to management of Primary Memory or Main Memory. Main memory is a large array of words or bytes where each word or byte has its own address.

Main memory provides a fast storage that can be accessed directly by the CPU. For a program to be executed, it must in the main memory. An Operating System does the following activities for memory management −

- Keeps tracks of primary memory, i.e., what part of it are in use by whom, what part is not in use?
- In multiprogramming, the OS decides which process will get memory when and how much.
- Allocates the memory when a process requests it to do so.
- De-allocates the memory when a process no longer needs it or has been terminated.

Processor Management

In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called **process scheduling**. An Operating System does the following activities for processor management −

- Keeps tracks of processor and status of process. The program responsible for this task is known as **traffic controller**.
- Allocates the processor (CPU) to a process.
- De-allocates processor when a process is no longer required.

Device Management

An Operating System manages device communication via their respective drivers. It does the following activities for device management −

- Keeps tracks of all devices. Program responsible for this task is known as the **I/O controller**.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.

File Management

A file system is normally organized into directories for easy navigation and usage. These directories may contain files and other directions.

An Operating System does the following activities for file management −

- Keeps track of information, location, uses, status etc. The collective facilities are often known as **file system**.
- Decides who gets the resources.
- Allocates the resources.
- De-allocates the resources.

Other Important Activities

Some of the important activities that an Operating System performs,

- **Security** − By means of password and similar other techniques, it prevents unauthorized access to programs and data.
- **Control over system performance** − Recording delays between request for a service and response from the system.
- **Job accounting** − Keeping track of time and resources used by various jobs and users.
- **Error detecting aids** − Production of dumps, traces, error messages, and other debugging and error detecting aids.
- **Coordination between other softwares and users** − Coordination and assignment of compilers, interpreters, assemblers and other software to the various users of the computer systems.

Types of Operating Systems:

Operating systems are there from the very first computer generation and they keep evolving with time. In this chapter, we will discuss some of the important types of operating systems which are most commonly used.

Batch Processing Operating System

The users of a batch operating system do not interact with the computer directly. Each user prepares his job on an off-line device like punch cards and submits it to the computer operator. To speed up processing, jobs with similar needs are batched together

and run as a group. The programmers leave their programs with the operator and the operator then sorts the programs with similar requirements into batches. The problems with Batch Systems are as follows ,

- Lack of interaction between the user and the job.
- CPU is often idle, because the speed of the mechanical I/O devices is slower than the CPU.
- Difficult to provide the desired priority.

Time-sharing Operating Systems

Time-sharing is a technique which enables many people, located at various terminals, to use a particular computer system at the same time. Time-sharing or multitasking is a logical extension of multiprogramming. Processor's time which is shared among multiple users simultaneously is termed as time-sharing.

The main difference between Multiprogrammed Batch Systems and Time-Sharing Systems is that in case of Multiprogrammed batch systems, the objective is to maximize processor use, whereas in Time-Sharing Systems, the objective is to minimize response time.

Multiple jobs are executed by the CPU by switching between them, but the switches occur so frequently. Thus, the user can receive an immediate response. For example, in a transaction processing, the processor executes each user program in a short burst or quantum of computation. That is, if **n** users are present, then each user can get a time quantum. When the user submits the command, the response time is in few seconds at most.

The operating system uses CPU scheduling and multiprogramming to provide each user with a small portion of a time. Computer systems that were designed primarily as batch systems have been modified to time-sharing systems.

Advantages of Timesharing operating systems:

- Provides the advantage of quick response.
- Avoids duplication of software.
- Reduces CPU idle time.

Disadvantages of Time-sharing operating systems are as follows −

- Problem of reliability.
- Question of security and integrity of user programs and data.
- Problem of data communication.

Distributed operating System

Distributed systems use multiple central processors to serve multiple real-time applications and multiple users. Data processing jobs are distributed among the processors accordingly.

The processors communicate with one another through various communication lines (such as high-speed buses or telephone lines). These are referred as **loosely coupled systems** or distributed systems. Processors in a distributed system may vary in size and function. These processors are referred as sites, nodes, computers, and so on.

The advantages of distributed systems are as follows −

- With resource sharing facility, a user at one site may be able to use the resources available at another.
- Speedup the exchange of data with one another via electronic mail.
- If one site fails in a distributed system, the remaining sites can potentially continue operating.
- Better service to the customers.
- Reduction of the load on the host computer.
- Reduction of delays in data processing.

Network operating System

A Network Operating System runs on a server and provides the server the capability to manage data, users, groups, security, applications, and other networking functions. The primary purpose of the network operating system is to allow shared file and printer access among multiple computers in a network, typically a local area network (LAN), a private network or to other networks.

Examples of network operating systems include Microsoft Windows Server 2003, Microsoft Windows Server 2008, UNIX, Linux, Mac OS X, Novell NetWare, and BSD.

The advantages of network operating systems are as follows −

- Centralized servers are highly stable.
- Security is server managed.
- Upgrades to new technologies and hardware can be easily integrated into the system.
- Remote access to servers is possible from different locations and types of systems.

The disadvantages of network operating systems are as follows −

- High cost of buying and running a server.
- Dependency on a central location for most operations.
- Regular maintenance and updates are required.

I/O Programming Concepts:

I/O Operation:

An I/O subsystem comprises of I/O devices and their corresponding driver software. Drivers hide the peculiarities of specific hardware devices from the users.

An Operating System manages the communication between user and device drivers.

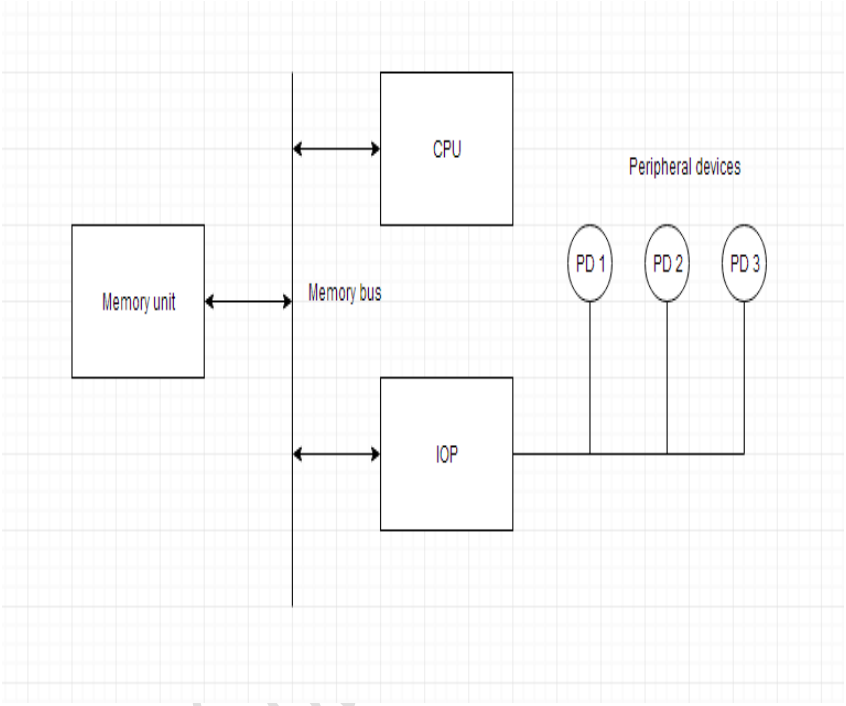
- I/O operation means read or write operation with any file or any specific I/O device.
- Operating system provides the access to the required I/O device when required.

Input/output Processor

An input-output processor (IOP) is a processor with direct memory access capability. In this, the computer system is divided into a memory unit and number of processors. Each IOP controls and manage the input-output tasks. The IOP is similar to CPU except that it handles only the details of I/O processing. The IOP can fetch and execute its own instructions. These IOP instructions are designed to manage I/O transfers only.

Block Diagram Of IOP

It is a block diagram of a computer along with various I/O Processors. The memory unit occupies the central position and can communicate with each processor. The CPU processes the data required for solving the computational tasks. The IOP provides a path for transfer of data between peripherals and memory. The CPU assigns the task of initiating the I/O program. The IOP operates independent from CPU and transfer data between peripherals and memory.



The communication between the IOP and the devices is similar to the program control method of transfer. And the communication with the memory is similar to the direct memory access method.

In large scale computers, each processor is independent of other processors and any processor can initiate the operation.

The CPU can act as master and the IOP act as slave processor. The CPU assigns the task of initiating operations but it is the IOP, who executes the instructions, and not the CPU. CPU instructions provide operations to start an I/O transfer. The IOP asks for CPU through interrupt.

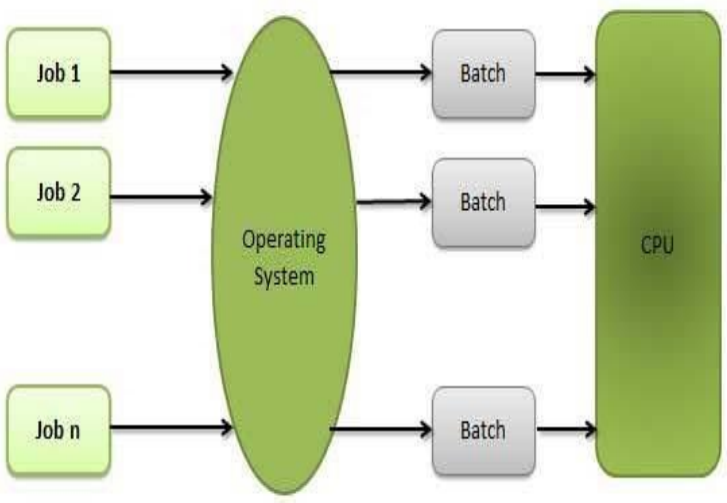
Instructions that are read from memory by an IOP are also called *commands* to distinguish them from instructions that are read by CPU.

Commands are prepared by programmers and are stored in memory.

Command words make the program for IOP. CPU informs the IOP where to find the commands in memory.

Batch processing OS

Batch processing is a technique in which an Operating System collects the programs and data together in a batch before processing starts.



An operating system does the following activities related to batch processing −

- The OS defines a job which has predefined sequence of commands, programs and data as a single unit.
- The OS keeps a number a jobs in memory and executes them without any manual information.
- Jobs are processed in the order of submission, i.e., first come first served fashion.
- When a job completes its execution, its memory is released and the output for the job gets copied into an output spool for later printing or processing.

Advantages

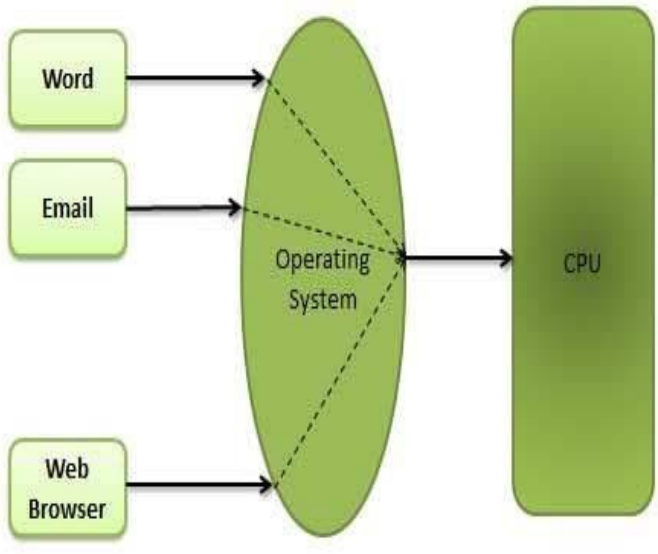
- Batch processing takes much of the work of the operator to the computer.
- Increased performance as a new job gets started as soon as the previous job is finished, without any manual intervention.

Disadvantages

- Difficult to debug program.
- A job could enter an infinite loop.
- Due to lack of protection scheme, one batch job can affect pending jobs.

Multitasking:

Multitasking is when multiple jobs are executed by the CPU simultaneously by switching between them. Switches occur so frequently that the users may interact with each program while it is running. An OS does the following activities related to multitasking −

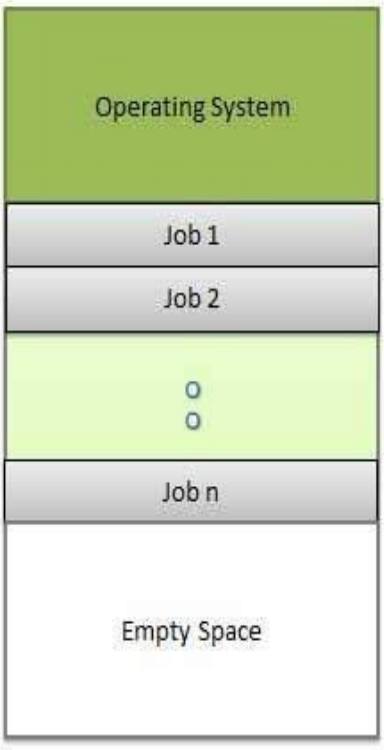


- The user gives instructions to the operating system or to a program directly, and receives an immediate response.
- The OS handles multitasking in the way that it can handle multiple operations/executes multiple programs at a time.
- Multitasking Operating Systems are also known as Time-sharing systems.

- These Operating Systems were developed to provide interactive use of a computer system at a reasonable cost.
- A time-shared operating system uses the concept of CPU scheduling and multiprogramming to provide each user with a small portion of a time-shared CPU.
- Each user has at least one separate program in memory.
- A program that is loaded into memory and is executing is commonly referred to as a **process**.
- When a process executes, it typically executes for only a very short time before it either finishes or needs to perform I/O.
- Since interactive I/O typically runs at slower speeds, it may take a long time to complete. During this time, a CPU can be utilized by another process.
- The operating system allows the users to share the computer simultaneously. Since each action or command in a time-shared system tends to be short, only a little CPU time is needed for each user.
- As the system switches CPU rapidly from one user/program to the next, each user is given the impression that he/she has his/her own CPU, whereas actually one CPU is being shared among many users.

Multiprogramming

Sharing the processor, when two or more programs reside in memory at the same time, is referred as **multiprogramming**. Multiprogramming assumes a single shared processor. Multiprogramming increases CPU utilization by organizing jobs so that the CPU always has one to execute.



An OS does the following activities related to multiprogramming.

- The operating system keeps several jobs in memory at a time.
- This set of jobs is a subset of the jobs kept in the job pool.
- The operating system picks and begins to execute one of the jobs in the memory.

- Multiprogramming operating systems monitor the state of all active programs and system resources using memory management programs to ensures that the CPU is never idle, unless there are no jobs to process.

Advantages

1) High and efficient CPU utilization.
2) User feels that many programs are allotted CPU almost simultaneously.

Disadvantages

- CPU scheduling is required.
- To accommodate many jobs in memory, memory management is required.

Interrupts:

When a Process is executed by the user and when a user Request for another Process then this will create disturbance for the Running Process. This is also called as the **Interrupt**.

Interrupts can be generated by User, Some Error Conditions and also by Software's and the hardware's. But CPU will handle all the Interrupts very carefully because when Interrupts are generated then the CPU must handle all the Interrupts Very carefully means the CPU will also Provides Response to the Various Interrupts those are generated. The Interrupt has Occurred then the CPU will handle by using the Fetch, decode and Execute Operations.

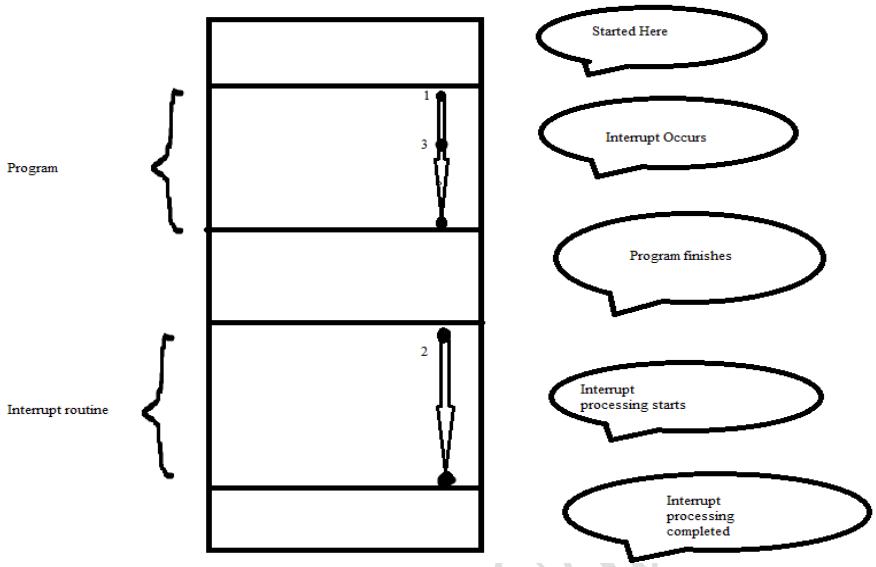
Interrupt structure and processing:

There are various ways to implement and define interrupt mechanisms.

Definition:

An interrupt is(1) A response to an asynchronous or exceptional event (2)Automatically saves the current CPU status to allow later restart (3) Causes an automatic transfer to a specified routine. The processor was executing a program;then the interrupt occurred and forced the transfer to the interrupt routine2.

Termination of the interrupt routine the processor resumed processingthe original program at point3



Interrupt Mechanism:

The "state" or current condition of the CPU is stored in a doubleword register called the program status word.

The PSW is used to control instruction sequencing and to hold and indicate the status of the system in relation to the program currently being executed.

The active or controlling PSW is the "current"PSW.

By storing the PSW during an interruption,the status of the CPU can be preserved for inspection or reloading.

loading a new PSW,or part of one,the state of the CPU can be changed.

System mask		Flags	Interruption code

ILC	CC	Program mask	Instruction address

The programmer has at his disposal various status switching instructions.

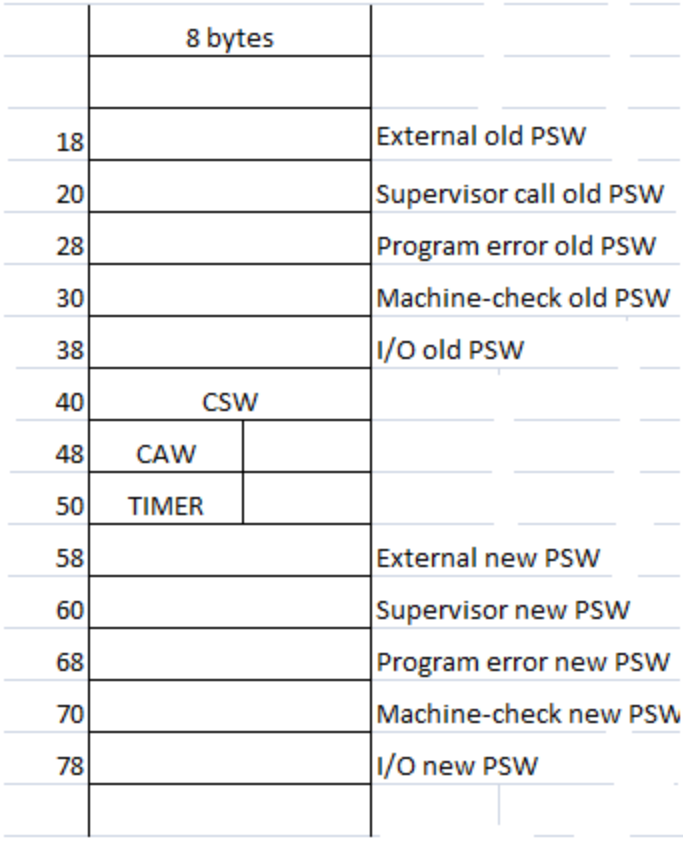
These include: Load Psw (LPSW),set program mask (SPM),set system mask(SSM),supervisor call(SVC),and set storage key(SSK).

Most of these are privileged instructions and can be executed only in supervisor mode.

Each of the five classes of interrupts-I/O,program,supervisor call,external,and machine check-has associated with it two doublewords, called "old"and"new" PSWs,stored in the main memory

at predetermined storage locations.

Interrupt hardware mechanism (1) stores the current PSW in the old position (2) loads the current PSW from the new position (3) there is usually an LPSW instruction to make the old PSW current again.



Interrupt handler processing:

The interrupt routine can access the appropriate old PSW to ascertion the condition that caused the interrupt.

The old PSW contains an interrupt code and the location of the program being executed when the interrupt occurred.

The program interrupt and external interrupt causes has a unique interrupt code : invalid operation =01; privileged operation =02; fixed point overflow=08,etc..,for program interrupts.

The PSW interrupt code indicates the channel and device causing the interrupt .

The CSW status field ,which is automatically set at the same time, contains information that indicates the cause of the interrupt ;CSW bit 36=1 means channel end,CSW bit 37=1 means device end.

Interrupt queuing:

(1) Completely masking interrupts while processing an interrupt
(2) Temporarily masking interrupts until the old PSW is safely copied and stacked elsewhere.

Example of Exceptional Interrupt processing:

A new exponentiation opcode , EXP R1,R2,which takes c(r1)c(cr2)→c(r1).The machine opcode for EXPis 00.Executes a 00 opcode a program interrupt will occur.

The programmer must provide the proper environment for the interrupt by setting the program interrupt new PSW with the address of his interrupt routine.

(1) A program interrupt code of X'0001' must be in bits 16-31 of the program PSW starting at location 40.
(2) An instruction length code of B'01' must be in bits 32 and33 of the program old PSW indicating a halfword instruction.
(3) Finally, the operation code must be checked by subtracting 2 from the current instruction address in bits 40-63 of the program old PSW to get the location of the interrupt instruction.

The unused word at location X'54' as a temporary save area in order to establish a base register.

Example of Asynchronous Interrupt Processing:

It keeps testing the I/O status (TIO) waiting for the I/O to complete .More than 99 percent of the time is really wasted.

The CPU will then spent most of its time performing some computation. When an interrupt signals that it is necessary to start the next I/O, the processor will quickly start the I/O and then resume the computation.

The "useful computation" is to count from 1to 1,000,000.

(1) Setting I/O interrupt new PSW to the address of the I/O interrupt handler
(2) Primary the"pump"by starting the first I/O to read a card
(3) Setting the system mask to accept I/O interrupts

Even if no errors occur,normal functioning may cause multiple interrupts: three may be

(1) Channel –end interrupts

(2) Control-unit-end interrupts
(3) Device-end interrupts

The interrupt handler only executes about 13 instructions to process each interrupt. If we assumed a processor speed of one million instructions per second, that is ,13 µs.
I/O handler could be written by using double buffering, to increase card read/card print rate from 500 per minute to 1,000 per minute.
I/O handlers must handle other concerns, such as error conditions and when to stop the I/O functioning.
Unit I Completed

Unit II:

Memory Management – Single Contiguous Allocation- Partitioned Allocation – Relocatable Partitions allocations – Paged and Demand paged Memory Management – Segmented Memory Management – Segmented and Demand paged Memory Management – overlay Techniques – Swapping

---

In operating systems, **memory management** is the function responsible for managing the computer's primary memory.

1. **Keeping track of the status of each location of primary memory.**

The memory management function keeps track of the status of each memory location, either *allocated* or *free*.

2. **Determining allocation policy for memory.**

   It determines how memory is allocated among competing processes, deciding which gets memory, when they receive it, and how much they are allowed.

3. **Allocation technique.**

   When memory is allocated it determines which memory locations will be assigned.

4. **Deallocation Technique and policy.**

   It tracks when memory is freed or *unallocated* and updates the status.

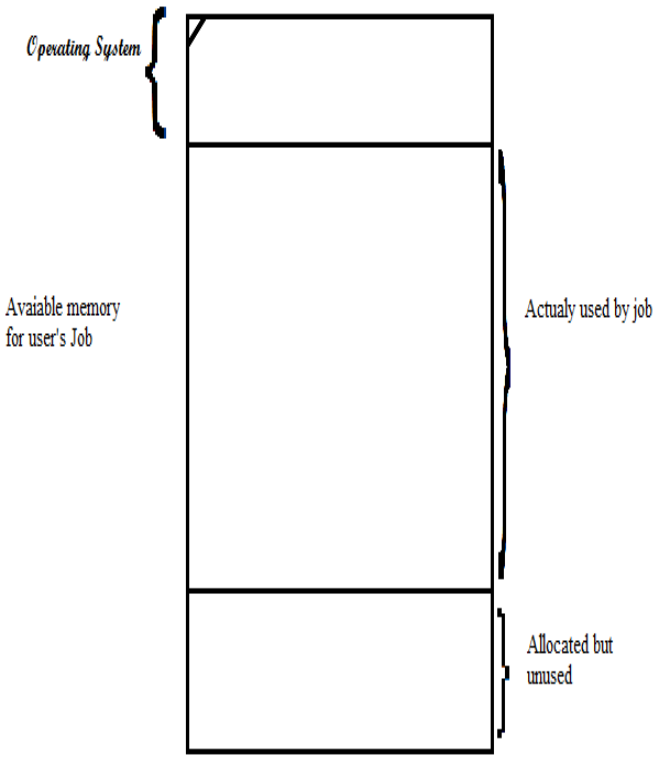
Memory management techniques

1. Single Contiguous Memory Management
2. Partitioned memory management
3. Relocation partitioned memory management
4. Paged memory management
5. Demand Paged memory management
6. Segmented memory management
7. Segmented and demand-paged memory management
8. Other memory management schemes.

Single contiguous allocation

Single allocation is the simplest memory management technique. The entire computer's memory, usually with the exception of a small portion reserved for the operating system, is available to the single application.

It is associated with small stand alone computers (mini computers) with simple batch

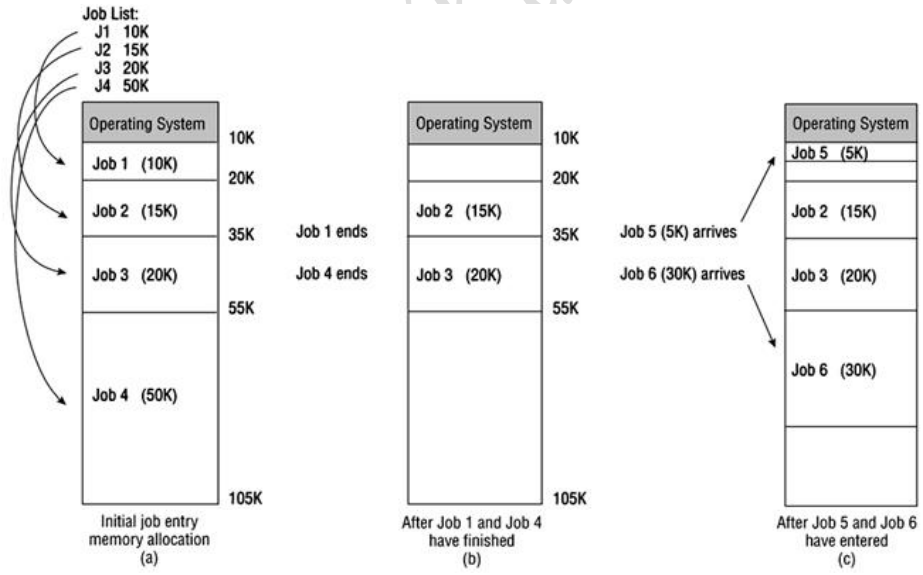


operating system. For ex IBM OS/360, IBM 1130, IBM 7094.These systems there is no multiprogramming and one –to-one process. MS-DOS is an example of a system which allocates memory in this way. An embedded system running a single application might also use this technique.

Memory is conceptually divided into 3 contiguous regions. A portion of memory is permanently allocated to OS. Remaining memory is available for user's job. A single job being processed one portion of memory is allocated and used. Leaving memory is allocated but unused memory.

For ex, if there is 256kb of memory, A simple OS may occupy 32kb, Leaving 224kb allocated for user's job. Any one job may require 64 kb, and then 160 kb of memory are unused. But entire 224 kb memory is available for User's job.

A system using single contiguous allocation may still multitask by swapping the contents of memory to switch among users. Early versions of the Music operating system used this technique.



Program is loaded in its entirety into memory and allocated as much contiguous space in memory as it needs.

1. **Keeping track of memory**-it is allocated entirely to one job.
2. **Determining factor on memory policy**-the job gets all memory when scheduled.
3. **Allocating of memory** –all of it is allocated to the job.
4. **Deallocation of memory** –when the job is done,all memory is returned to free status.

Advantages of Single Contiguous Allocation:

1. Jobs processed sequentially in single-user systems
2. Requires minimal work by the Memory Manager
3. Register to store the base address Accumulator to keep track of the program size

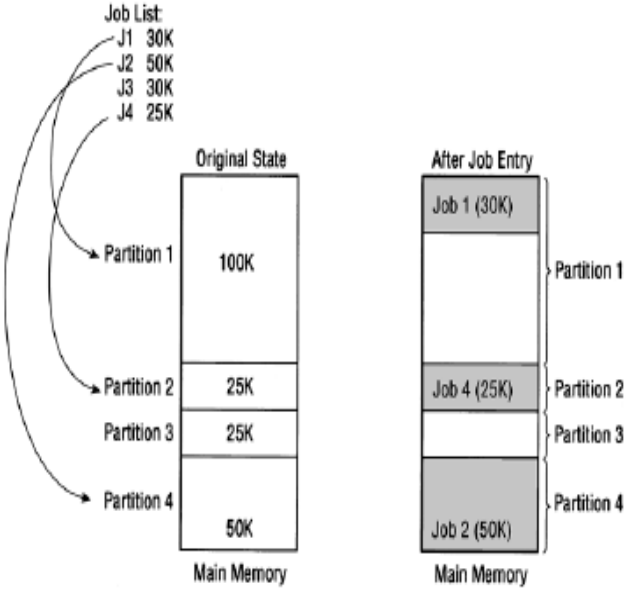
Disadvantages of Single Contiguous Allocation:

1. Doesn't support multiprogramming
2. Not cost effective

Partitioned Memory Management

Partitioned allocation divides primary memory into multiple *memory partitions*, usually contiguous areas of memory. Each partition might contain all the information for a specific job or task. Memory management consists of allocating a partition to a job when it starts and unallocating it when the job ends.

Partition Size	Memory Address	Access	Partition Status
100K	200K	Job 1	Busy
25K	300K	Job 4	Busy
25K	325K		Free
50K	350K	Job 2	Busy



Partitioned allocation usually requires some hardware support to prevent the jobs from interfering with one another or with the operating system. The IBM System/360 used a *lock-and-key* technique. Other systems used *base and bounds* registers which contained the limits of the partition and flagged invalid accesses. The UNIVAC 1108 *Storage Limits Register* had separate base/bound sets for instructions and data. The system took advantage of memory interleaving to place what were called the *i bank* and *d bank* in separate memory modules.

Four functions of memory management:

1. **Keeping track of the status of each partition** (eg., "in use" or "not in use", size)
2. **Determining who gets memory** – this is handled largely by the job scheduler.
3. **Allocation** –an available partition of sufficient size is assigned.
4. **Deallocation** –when the job terminates, the partition is indicated "not in use" and is available for future allocation.

Advantages:

1. More efficient utilization of the processor and I/O device.
2. It requires no special costly hardware.
3. The algorithm used are simply and easy to implement.

Disadvantages:

1. Fragmentation can be a significant problem.
2. Even if memory is not fragmented,the single free area may not be large enough for a partition.
3. It does require more memory than a single contiguous allocation system in order to hold more than one job.
4. Contiguous allocation, memory may contain information that is never used.

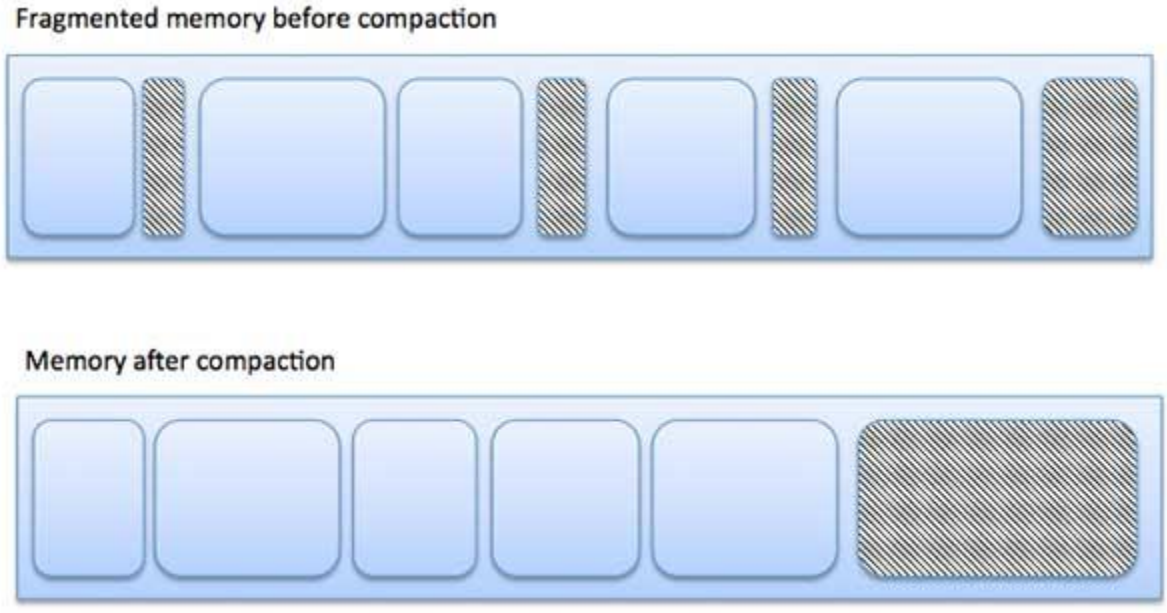
Fragmentation

As processes are loaded and removed from memory, the free memory space is broken into little pieces. It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused. This problem is known as Fragmentation.

Fragmentation is of two types −

S.N.	Fragmentation & Description
1	**External fragmentation** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
2	**Internal fragmentation** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −

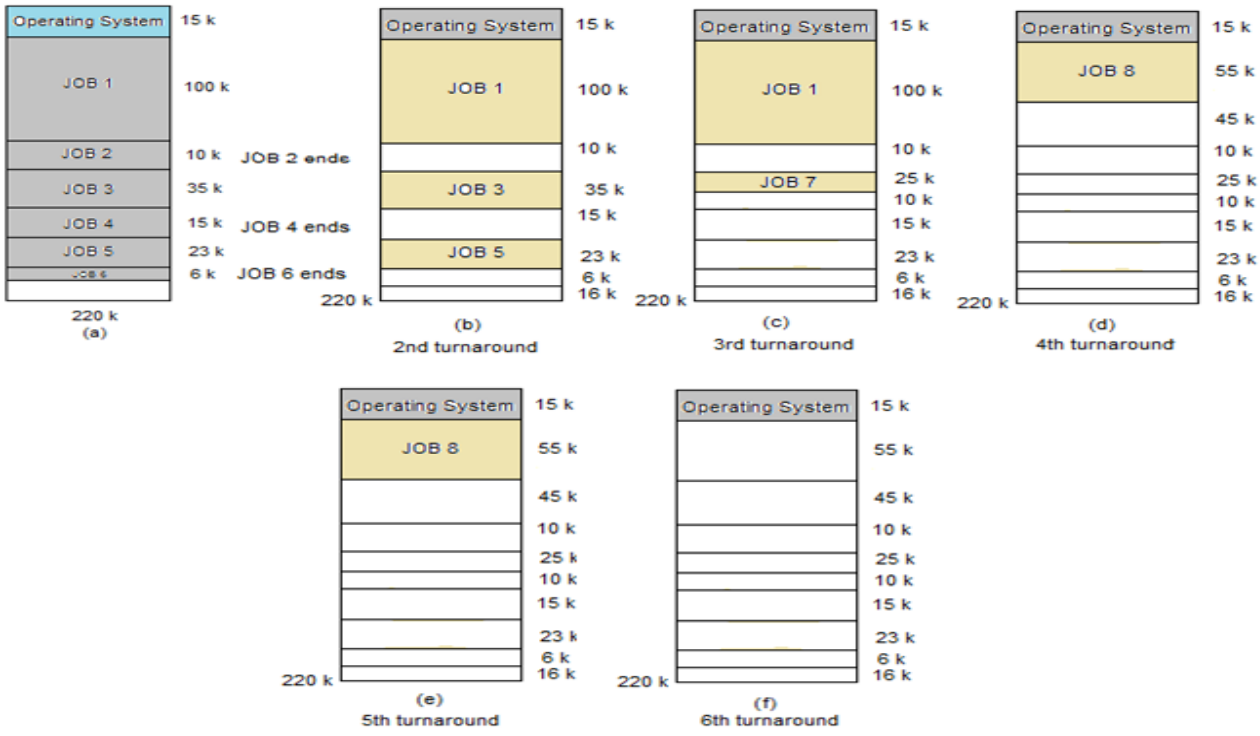


External fragmentation can be reduced by compaction or shuffle memory contents to place all

free memory together in one large block. To make compaction feasible, relocation should be dynamic.

The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

Relocatable Partition Memory Management:Partitions may be either *static*, that is defined at Initial Program Load (IPL) or *boot time* or by the computer operator, or *dynamic*, that is automatically created for a specific job. IBM System/360 Operating



System *Multiprogramming with a Fixed Number of Tasks* (MFT) is an example of static partitioning, and *Multiprogramming with a Variable Number of Tasks* (MVT) is an example of dynamic. MVT and successors use the term *region* to distinguish dynamic partitions from static ones in other systems.

Partitions may be *relocatable* using hardware *typed memory*, like the Burroughs Corporation B5500, or base and bounds registers like the PDP-10 or GE-635. Relocatable partitions are able to be *compacted* to provide larger chunks of contiguous physical memory. Compaction moves "in-use" areas of memory to eliminate "holes" or unused areas of memory caused by process termination in order to create larger contiguous free areas.

Some systems allow partitions to be *swapped out* to secondary storage to free additional

memory. Early versions of IBM's *Time Sharing Option* (TSO) swapped users in and out of a single time-sharing partition.

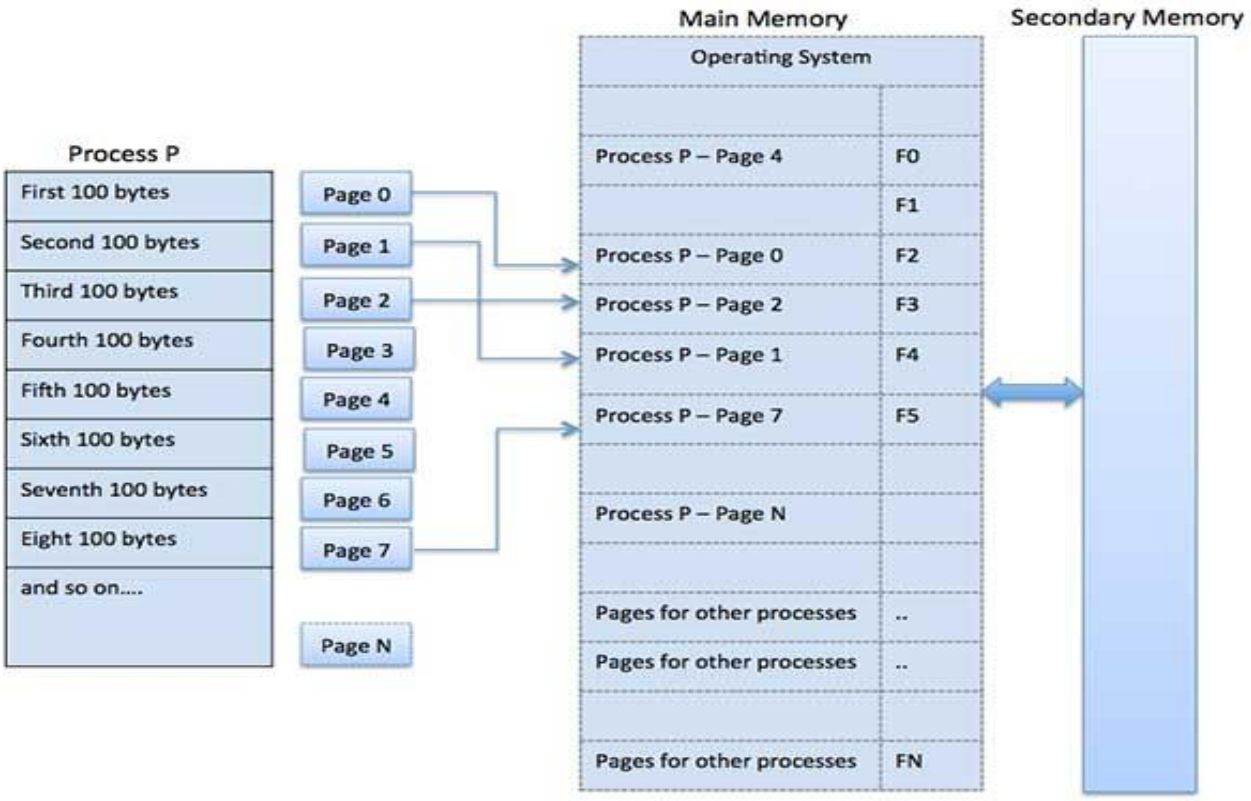
Advantages:

1. The relocatable partition scheme eliminates fragmentation and thus makes it possible to allocate more partitions.
2. This allows for a higher degree of multiprogramming, which results in increased memory and processor utilization.

Disadvantages:

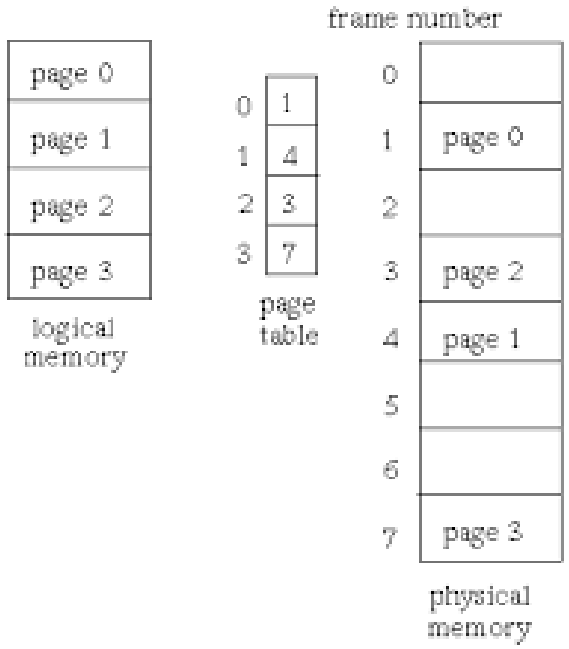
1. Relocation hardware increases the cost of the computer and may slow down the speed.
2. Compaction time may be substantial.
3. Some memory will still be unused because even though it is compacted, the amount of free area may be less than the needed partition size. This wasted space is approximately equal to one –half the average job partition size.
4. As with partitioned allocation, memory may contain information that is never used. Furthermore, a job's partition size is limited to the size of physical memory.

Paged memory management



Paged allocation divides the computer's primary memory into fixed-size units called *page frames*, and the program's virtual *address space* into *pages* of the same size. The hardware memory management unit maps pages to frames. The physical memory can be allocated on a page basis while the address space appears contiguous.

Usually, with paged memory management, each job runs in its own address space. However, there are some single address space operating systems that run all processes within a single address space, such as IBM i, which runs all processes within a large address space, and IBM OS/VS2 SVS, which ran all jobs in a single 16MiB virtual address space.



A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

Paged memory can be *demand-paged* when the system can move pages as required between primary and secondary memory.

Four functions of memory management:

1. **Keeping track of status** – accomplished through two sets of tables:
   a) Page Map Tables – one per address space, each containing one entry for each page.
   b) Memory Block Table – one in system, containing one entry for each memory block with information on the use of that block (eg., allocate or available).
2. Determining who gets memory - this is largely decided by the job scheduler. Memory may be assigned to the user simply by giving him the first set of free blocks found.
3. **Allocation**- all pages of the job must be loaded into assigned blocks and appropriate entries made in the Page Map Table and Memory Block Table.

4. **Deallocation** – when the job is done, blocks must be returned to free status by adjusting entries in block table.

Address Translation

Page address is called **logical address** and represented by **page number** and the **offset**.

Logical Address = Page number + page offset

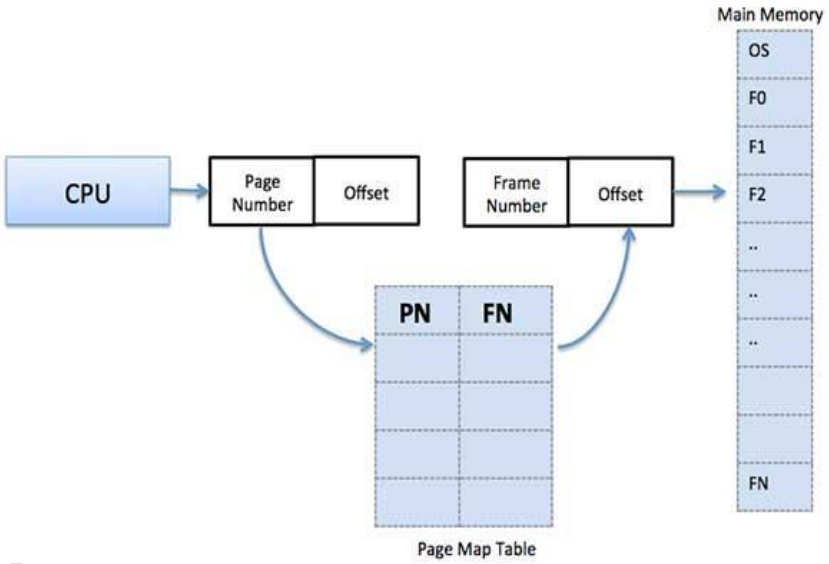
Frame address is called **physical address** and represented by a **frame number** and the **offset**.

Physical Address = Frame number + page offset

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.

Advantages and Disadvantages of Paging

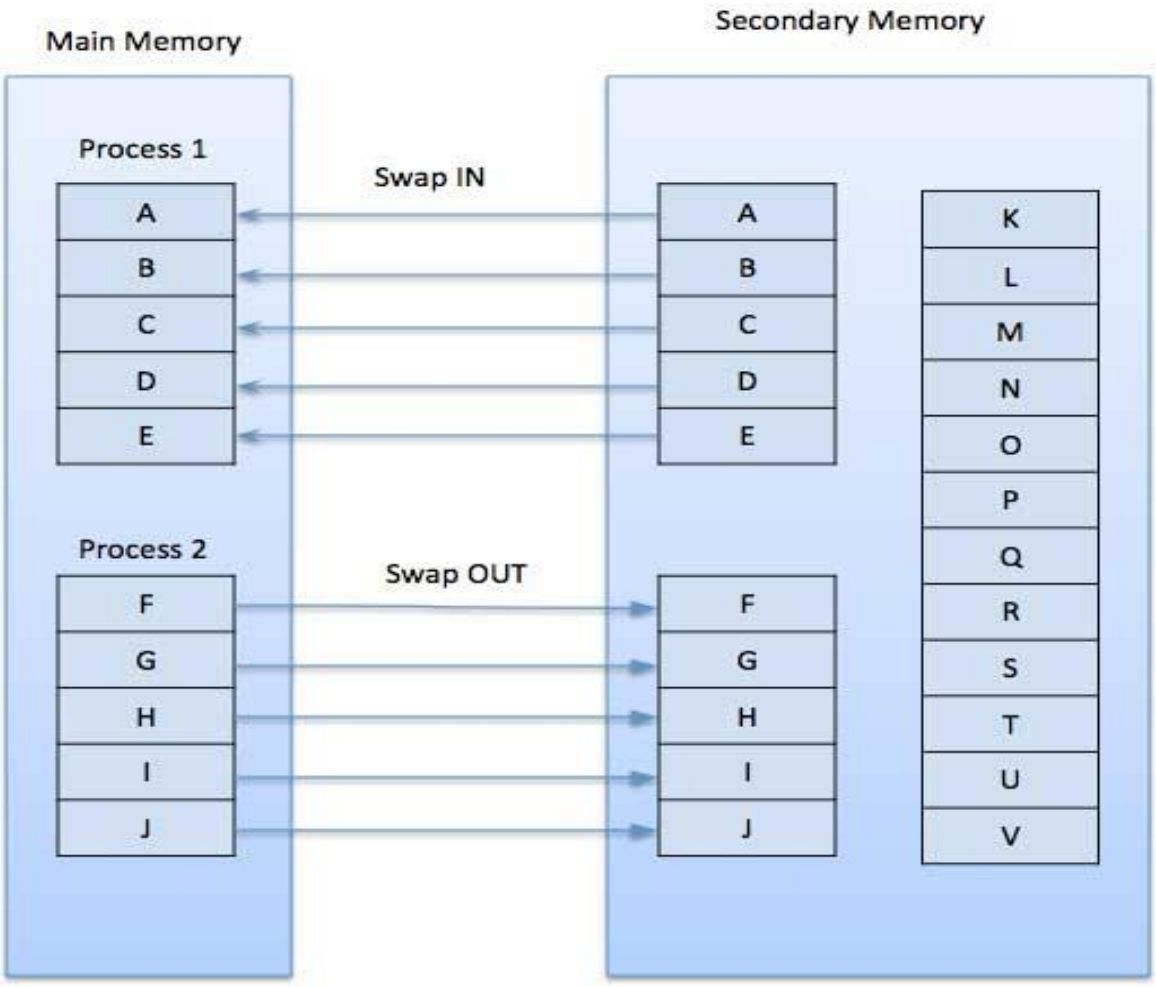
Here is a list of advantages and disadvantages of paging −



- Paging reduces external fragmentation, but still suffer from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Demand Paging

A demand paging system is quite similar to a paging system with swapping where processes reside in secondary memory and pages are loaded only on demand, not in advance. When a context switch occurs, the operating system does not copy any of the old program's pages out to the disk or any of the new program's pages into the main memory Instead, it just begins executing the new program after loading the first page and fetches that program's pages as they are referenced.



While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Four functions of memory management:

1. **Keeping track of status – this is accomplished through three sets of tables:**
   a. Page Map Tables – one per address space.
   b. Memory Block Tables – one in system.
   c. File Map Tables – one per address space.
2. The policy of who gets memory and when- partially determined by the job scheduler, but, on a dynamic basis, it is also determined by the demand page interrupts.
3. **Allocation –** when a block must be allocated, an available block must be found and the status of the block altered.
4. **Deallocation** – if it is not possible to find an available block for allocation,one of the allocated memory blocks must be deallocated and reassigned. When a job terminates, all the blocks it was using become available.

Advantages:

Following are the advantages of Demand Paging −

- Large virtual memory.
- More efficient use of memory.
- There is no limit on degree of multiprogramming.

Disadvantages:

- Number of tables and the amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.
- Due to the lack of an explicit constraint on a job's address space size or amount of multiprogramming, it is necessary to develop approaches to prevent thrashing.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

Reference String

The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference. The latter choice produces a large number of data, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.

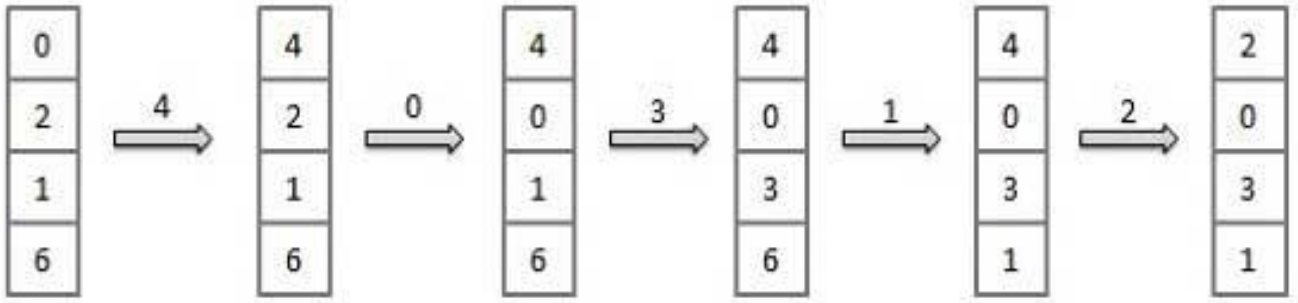
- For example, consider the following sequence of addresses − 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0

First In First Out (FIFO) algorithm

- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



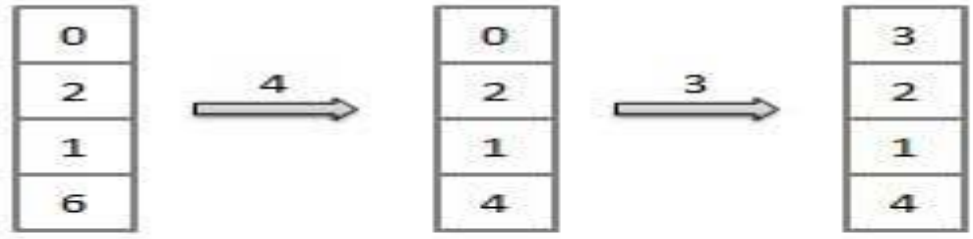
Fault Rate = 9 / 12 = 0.75

Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



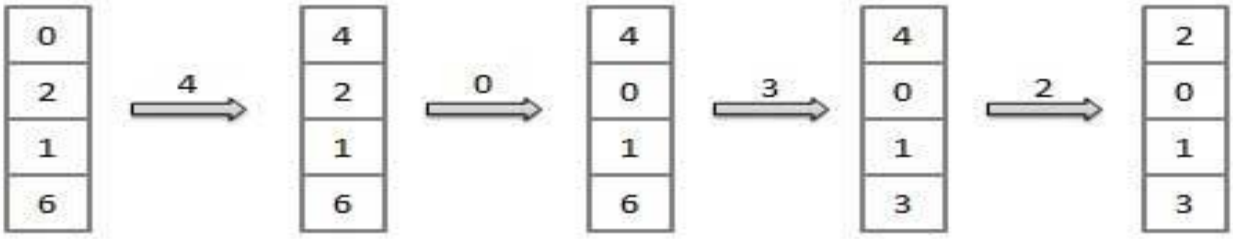
Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



Fault Rate = 8 / 12 = 0.67

Page Buffering algorithm

- To get a process start quickly, keep a pool of free frames.
- On page fault, select a page to be replaced.

- Write the new page in the frame of free pool, mark the page table and restart the process.
- Now write the dirty page out of disk and place the frame holding replaced page in free pool.

Least frequently Used(LFU) algorithm

- The page with the smallest count is the one which will be selected for replacement.
- This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

Most frequently Used(MFU) algorithm

- This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

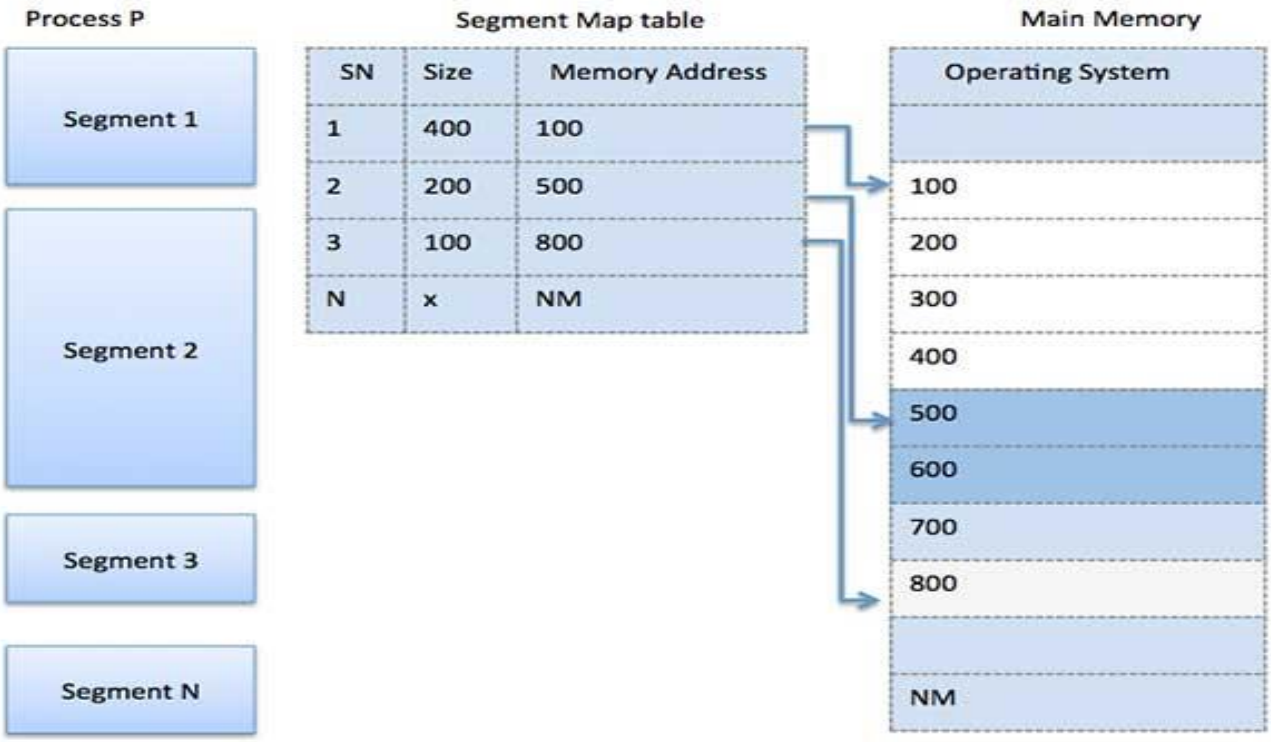
Segmented memory management

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the



segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.

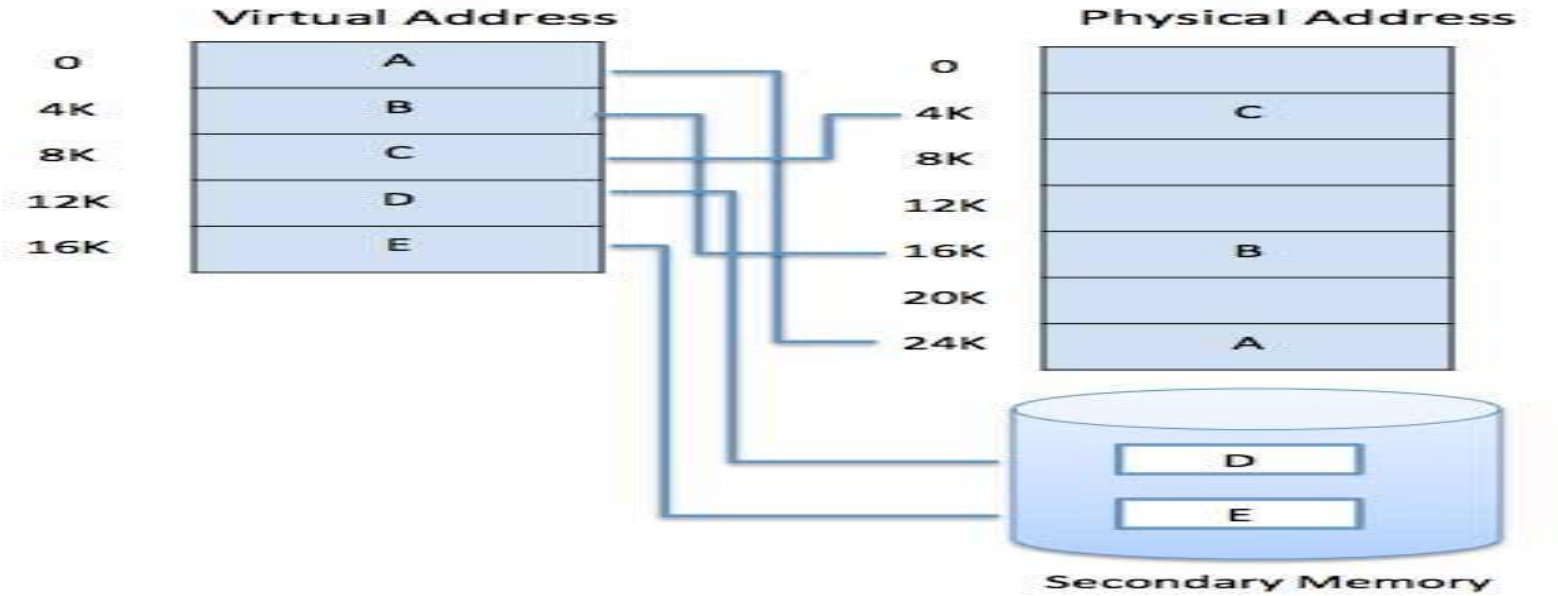
Virtual Memory:

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

- User written error handling routines are used only when an error occurred in the data or computation.
- Certain options and features of a program may be used rarely.
- Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
- The ability to execute a program that is only partially in memory would counter many benefits.
- Less number of I/O would be needed to load or swap each user program into memory.
- A program would no longer be constrained by the amount of physical memory that is available.
- Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.



Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given here,

Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Four functions of memory management:

1. **Keeping track of status is done through four main sets of tables:**
   a. Segmented Map Tables – one per address space.
   b. Unallocated Area Tables – one in system.
   c. Active Reference Table – one per address space.
   d. Active Segmente table – one in system.
2. The policy of who gets memory and when may be determined statically by the job scheduler if virtual memory use is limited
3. **Allocation** – when a segment must be allocated, a large enough available area must be found.
4. **Deallocation** – if it is not possible to find a large enough available area for segment allocation, one or more of the currently allocated segments must be deallocated.

Advantages:

1. Eliminate fragmentation
2. Provide virtual memory
3. Allow dynamically growing segments or automatic bounds checking: increased during execution "out-of-range" reference can be detected by the size component.
4. Dynamic linking and loading: most systems space statically during the linking and start of execution. For a large program, time consuming execution.
5. Facilitate shared segments: Square root routine, it is wasteful and possibly undesirable to have two separate copies exits in main memory.
6. Enforced controlled access: Access to each segment should be controlled. A table of constants should be restricted to read accesses only, whereas a work-space segment may be written as well as read.

Segmented and demand paged memory management:

Instead of treating each segment as a single contiguous entity,each one can be subdivided into pages.

By physically manipulating these pages, rather than the entire segment, problems of compaction, secondary storage handling, and limitation on segment size are removed.

The choice of page and segment size is determined by control register settings made by the operating system.

The following description applies to the 4K-byte page and 64K-byte segment settings.

1) Segment number

2) Page number
3) Byte number

Indicates the location and length of the current Segment Map Table. Each Segment Map Table Entry indicates the location and length of a Page Map table- unless the segment exception interrupt indicator is set. Each Page Map Table Entry (PMTE) indicates the corresponding block number for each page of that segment – unless the page exception interrupt indicator is set.

Map Table address field in each SMTE should require a full 24-bit address. However, by requiring PMTs to start at an address that is a multiple of 8, only 21bits are needed in the SMTE

In order to prevent serious performance degradation dut to access to the SMT and PMT , a Translation Lookaside Buffer(TLB)- an associative scratch pad memory- is automatically used by the hardware to eliminate most SMT and PMT accesses. The use of the TLB can generally keep address translation overhead below 10 percent.

Advantages:

The combination of segmentation and demand paging provides all the advantages previously listed in sections

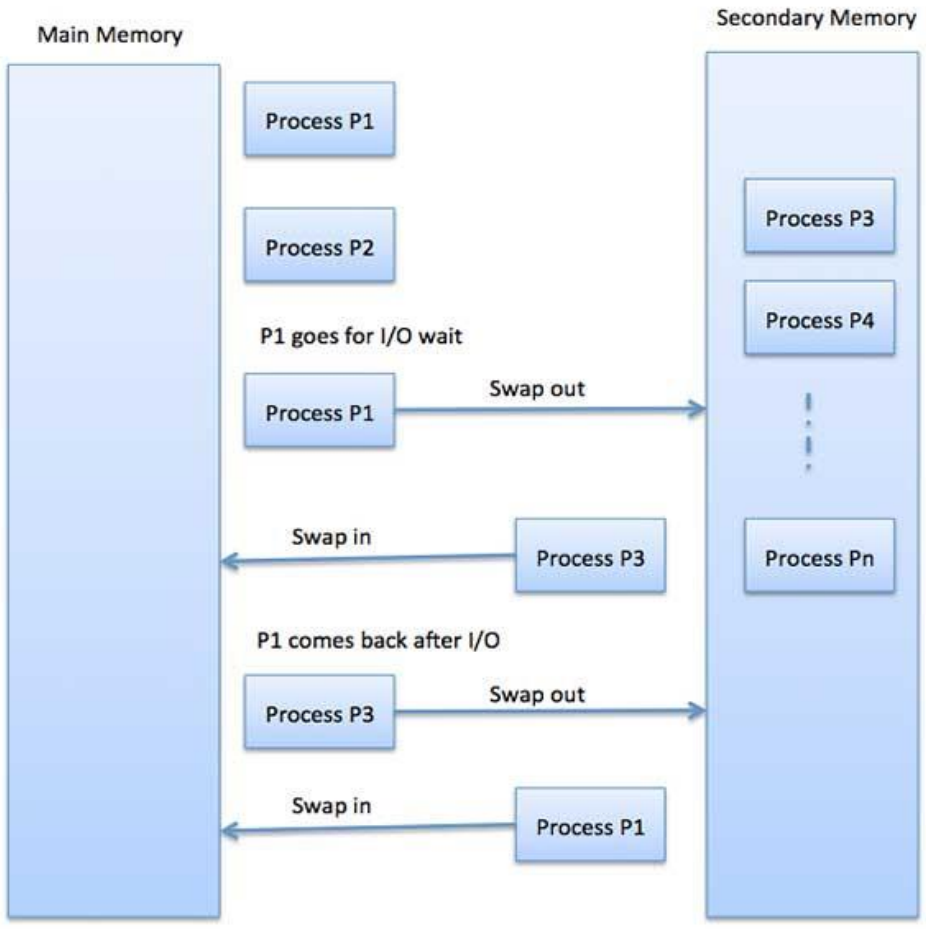
Disadvantages:

1. Hardware cost, processor overhead, and complexity.
2. Page breakage and memory needed for SMTs and PMTs persist.

Swapping:

Swapping is a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason Swapping is also known as a technique



for memory compaction.

The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

2048KB / 1024KB per second

= 2 seconds

= 2000 milliseconds

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

Swapped between main memory and secondary storage.

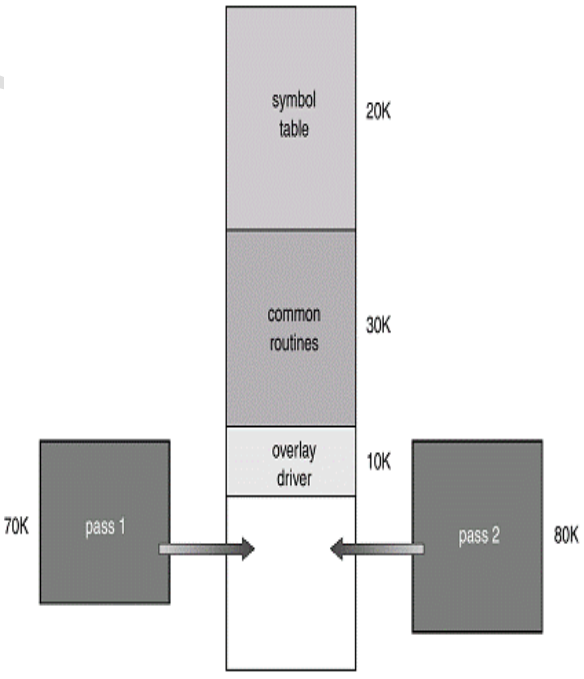
The early M.I.T compatible Time-Sharing System, one job was in memory at a time was swapped onto secondary storage to allow another job to run in main memory.

Many contemporary small-scale timesharing system use this swapping technique.

Overlays:

To Enable a process to be larger than the amount of memory allocated to it,we can use overlays. The idea of overlays is to keep in memory only those instructions and data that are needed at any given time. When other instructions are needed, they are loaded into space occupied previously by instructions that are no longer needed. We illustrate the concept of overlays with the example of a two-pass compiler. Here are the various specifications:

1. 2-pass assembler/Compiler
2. Available main memory: 150k
3. Code size :200k
   - ⇨ Pass 1 ----------70k
   - ⇨ Pass 2 ----------80k
   - ⇨ Common routines---------30k
   - ⇨ Symbol table----------20k



Common routines, symbol table, overlay driver, and pass1 code are loaded into the main memory fro the program execution to start. When pass 1 has finished it's work, pass 2 cade is loaded on top of the pass 1 code(because this code is not needed anymore). This way, we can execute a 200k process in a 150k memory.

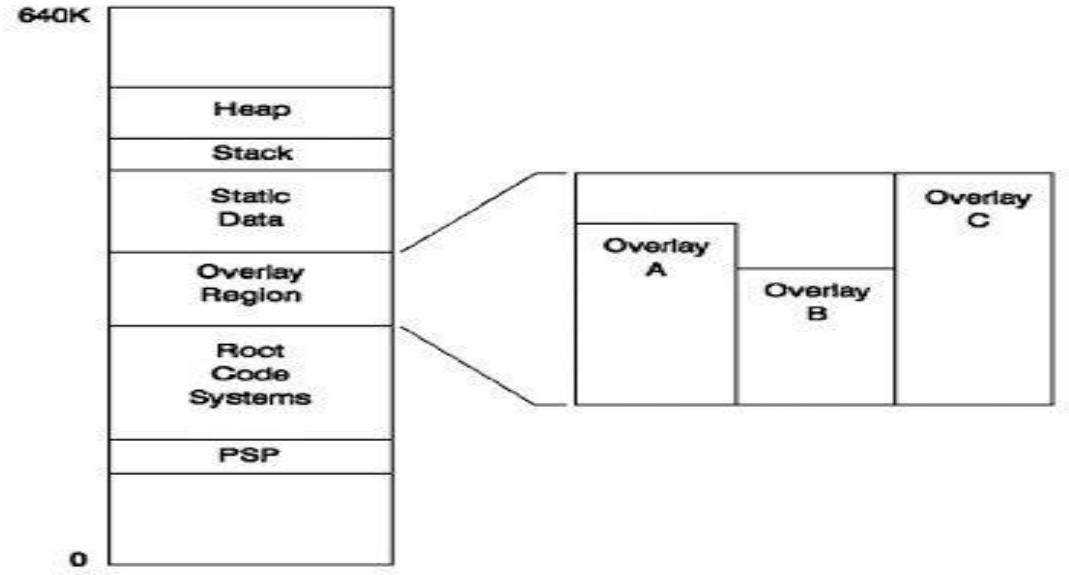
The problems with overlays are,

1. You may not be able to partition all problems into overlays
2. Programmer is responsible of writing the overlays driver.

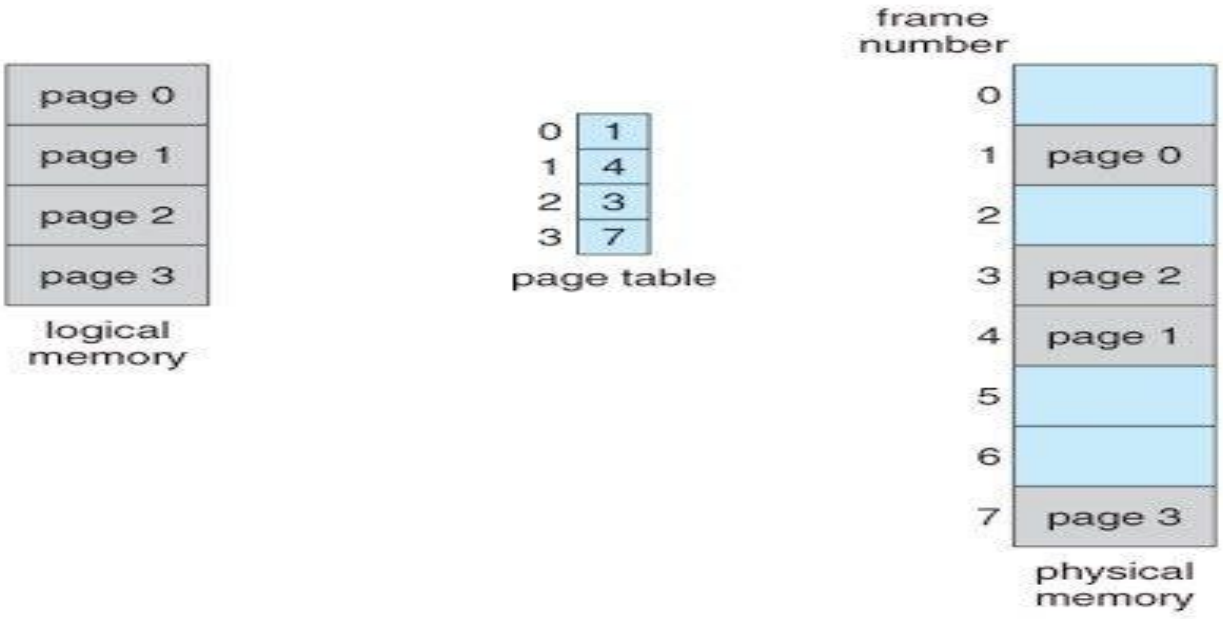
Difference between Overlays and Paging Memory Management:

Overlay is a pretty much antiquated technique. It requires the programmers to split their object code to into multiple completely-independent sections, and the overlay manager thats linked to the code will load the required overlay dynamically & will swap them when necessary.

Figure 1



This technique requires the programmers to specify which overlay to load at different circumstances. It's not very efficient & it causes delay while the overlays the loaded into the overlay address space.



Also, it's not possible to access overlays that are not loaded in the memory.

On the other hand, paging is a very efficient form of memory management, where the system is capable of simulating more memory than that is available for a given process. It's also called virtual memory.

The system uses page tables and address mapping to make it appear that each process is getting a chunk of contigious block of memory for itself.

In reality, it'll just keep swapping the active/inactive pages between the memory & the secondary storage in the background in a transparent manner. The efficiency of virtual memory far exceeds 100% in most of the cases. But on occasion it fails due to excessive disk trashing.

Both the memory management techniques have both good and bad. And each has its own advantages. For example, paging usually requires a microprocessor that is capable of memory mapping & exception handling. So, it can't run on very basic processors like 8085, 8086, etc. On the other hand, overlays is very much capable of working with these processors efficiently.

Unit II Complted.

==========

Unit III

Processor Management:

Overview:

What is a Process?

A process is a program in execution. Process is not as same as program code but a lot more than it. A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity. Attributes held by process include hardware state, memory, CPU etc.

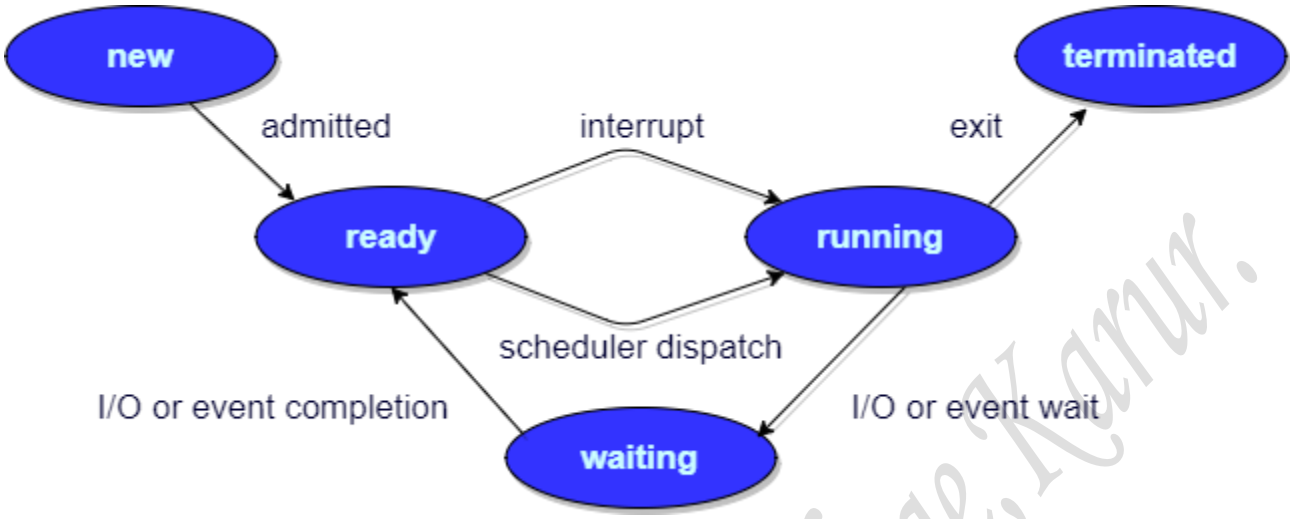
Process memory is divided into four sections for efficient working :

- The text section is made up of the compiled program code, read in from non-volatile storage when the program is launched.
- The data section is made up the global and static variables, allocated and initialized prior to executing the main.
- The heap is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- The stack is used for local variables. Space on the stack is reserved for local variables when they are declared.

PROCESS STATE

Processes can be any of the following states :

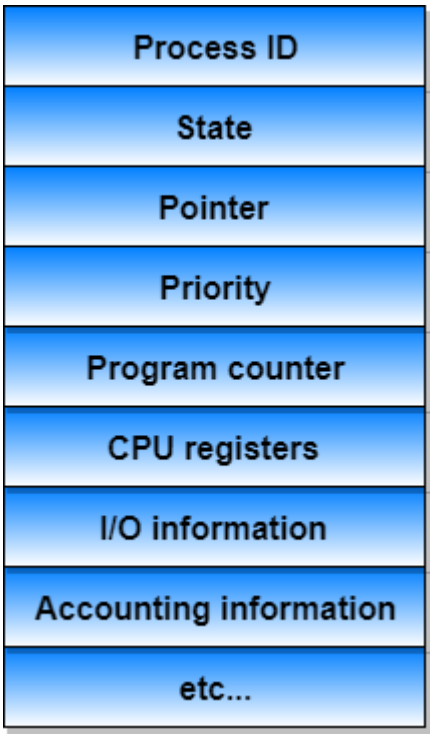
- **New** - The process is being created.
- **Ready** - The process is waiting to be assigned to a processor.
- **Running** - Instructions are being executed.
- **Waiting** - The process is waiting for some event to occur(such as an I/O completion or reception of a signal).
- **Terminated** - The process has finished execution.



PROCESS CONTROL BLOCK

There is a Process Control Block for each process, enclosing all the information about the process. It is a data structure, which contains the following :

- Process State - It can be running, waiting etc.
- Process ID and parent process ID.
- CPU registers and Program Counter. **Program Counter** holds the address of the next instruction to be executed for that process.
- CPU Scheduling information - Such as priority information and pointers to scheduling queues.
- Memory Management information - Eg. page tables or segment tables.
- Accounting information - user and kernel CPU time consumed, account numbers, limits, etc.
- I/O Status information - Devices allocated, open file tables, etc.



Process:

A process is a program in execution. Process is not as same as program code but a lot more than it. A process is an 'active' entity as opposed to program which is considered to be a 'passive' entity. Attributes held by process include hardware state, memory, CPU etc.

Process memory is divided into four sections for efficient working :

- The text section is made up of the compiled program code, read in from non-volatile storage when the program is launched.
- The data section is made up the global and static variables, allocated and initialized prior to executing the main.
- The heap is used for the dynamic memory allocation, and is managed via calls to new, delete, malloc, free, etc.
- The stack is used for local variables. Space on the stack is reserved for local variables when they are declared.

State Model:

I/O process has a similar state diagram-it became ready when my CPU process initiated i/o.When the i/o process finished, it signaled my blocked process that it was through. When the operating system received the signal that my i/o Process was finished, the traffic controller module changed my CPU Process from the blocked state to the ready state again.This time when the Process scheduler assigned a Processor to my Process, I finished all my work.

MODULE	FUNCTION
Spooler	1. Place all submitted jobs into a form for processing by the job scheduler
	2. Keep track of all jobs in system
	3. Select a job to run and create corresponding process
Job scheduler	Submit Hold
Process scheduling	4. Select a process to run and allocate a processor
	5.
Traffic controller	Keep track of status of all processes

6. Provide the mechanics for changing Process' states

Hold Ready

7. Coordinate interprocess synchronization and communication

Ready Running

Figure 3.1 Job Scheduling

Job Scheduling Versus Process Scheduling:

Job Scheduler:

Functions:

- Keep track of the status of all jobs. it must note which jobs are trying to get some service and the status of all jobs being serviced.
- Choose the policy by which jobs will "enter the system". This decision may be based on such characteristics as priority, resources requested, or system balance.
- Allocate the necessary resources for the scheduled job by use of memory, device, and Processor management.
- Deallocate these resources when the job is done.

Job and Process Synchronization:

On the job level there are usually mechanisms for passing conditions between job steps. For example, we might want to prevent job step 4 from being performed if job step 1 had failed.

On the process level there must be mechanisms to prevent race conditions. A race condition exists when the result of a computation varies, depending on the timing of other processes.

The p and v operators and semaphores are one set of mechanisms for coordinating the assignment of processor to processes.

In addtion, we must be aware of possible dead block situations in cases where there are two processes, each of which is waiting for resources that the other has and will not give up. In this situation, no processor can be assigned to either process.

Structure of Process Management:

Processor management operates on two levels-assigning processors to jobs and assigning processor to processes.

Processor management is concerned with such question as which jobs will be run and which will be run first. At this first level Processor management is not concerned with multiprogramming.

The job scheduler is like the coordinator of a contest; he decides who will enter the contest but not who will win it.

The process scheduler decides which participant will win. In a non multiprogramming environment only one contestant is allowed to enter.

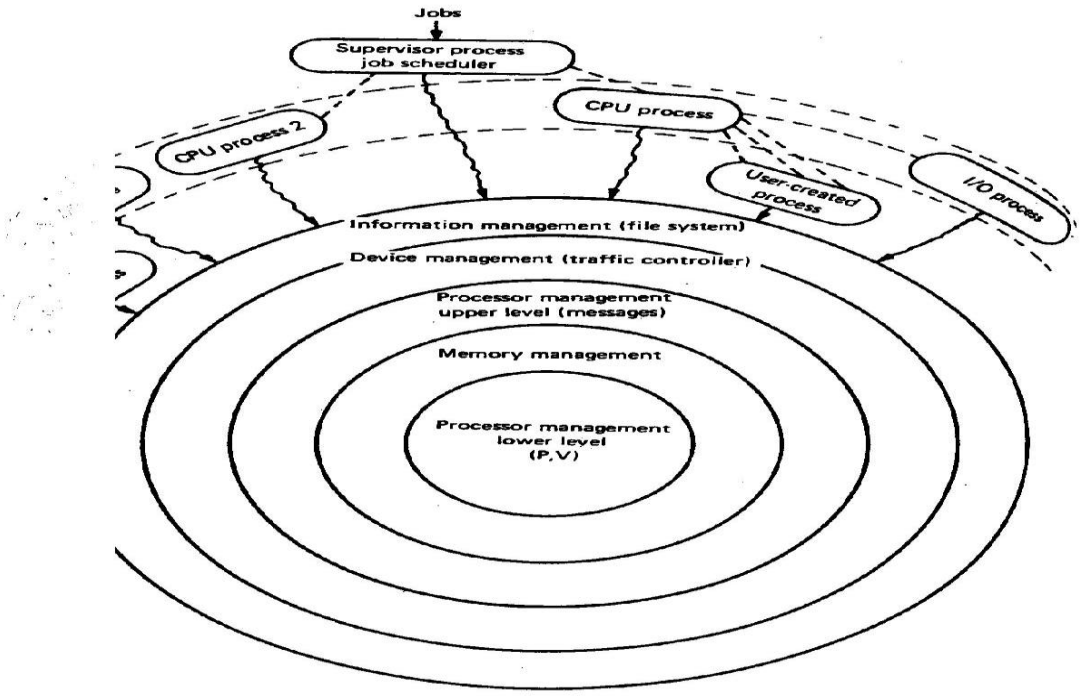


Figure.32 Hierarchical view of computer-based operating system levels for a real or extended machine

Job Scheduling:

Job scheduling is the process of allocating system resources to many different tasks by an operating system (OS). The system handles prioritized job queues that are waiting CPU time and it should determine which job to be taken from which queue and the amount of time to be allocated for the job. This type of scheduling makes sure that all jobs are carried out fairly and on time.

The functions of the job scheduler may be done quite simply , quite elaborately, or dynamically.

In a time sharing system such as the compatible time sharing system[CTSS]. The job scheduling policy may consist of admitting the first 30 users that log in.

A simple two-level priority algorithm allows a user with a higher priority to force the logout of a lower priority user.

The user to be logged out is chosen on the basis of the processor time he has used: The one who

has used the most processor time gets logged out.

Job scheduling is performed using job schedulers. Job schedulers are programs that enable scheduling and, at times, track computer "batch" jobs, or units of work like the operation of a payroll program. Job schedulers have the ability to start and control jobs automatically by running prepared job-control-language statements or by means of similar communication with a human operator. Generally, the present-day job schedulers include a graphical user interface (GUI) along with a single point of control.

Organizations wishing to automate unrelated IT workload could also use more sophisticated attributes from a job scheduler, for example:

- Real-time scheduling in accordance with external, unforeseen events
- Automated restart and recovery in case of failures
- Notifying the operations personnel
- Generating reports of incidents
- Audit trails meant for regulation compliance purposes

In-house developers can write these advanced capabilities; however, these are usually offered by providers who are experts in systems-management software.In scheduling, many different schemes are used to determine which specific job to run. Some parameters that may be considered are as follows:

- Job priority
- Availability of computing resource
- License key if the job is utilizing a licensed software
- Execution time assigned to the user
- Number of parallel jobs permitted for a user
- Projected execution time
- Elapsed execution time
- Presence of peripheral devices
- Number of cases of prescribed events

Functions:

The job scheduler can be viewed as an overall supervisor that assigns system resources to certain jobs. Therefore,it must keep track of the jobs, invoke policies for deciding which jobs

gets resources, and allocate and deallocate the resources.

One mechanism for keeping track of jobs is to have a separate job control block (JCB) for each job in the system .

When a process is placed in hold state a jcb is created for it with entries regarding its status and position in a job queue.

- Type of information
- The job control cards submitted with job.
- Current state, is set by the operating system.

Job identification
Current state
Priority
Time estimate
Etc..,

Policies:

The "hold" jobs those will be made "ready" to run. In a small computing center this function may be done by an operator.

He may choosen arbitrarily ,choose his friends , or choose the shortest job. In larger system , eg..,OS/360, all submitted jobs may be first stored on a secondary storage device whereupon the job scheduler can exaime all such jobs.

- Most resources are finite(eg.., tapes ,memory)
- Many resources cannot be shared or easily reassigned to another process.

Job Scheduling In Nonmultiprogrammed Environment:

The section let us assume no multiprogramming;that is, once a processor has been assigneda processor,it does not release the processor until it is finished.

We will first examine a simple of scheduling jobs using a policy of trying to reduce the average turnaround time.

Since there is no multiprogramming, we assume one CPU process is created for each job.

Thus the terms job and process may be used interchangeably.

Job Scheduling Using FIFO

Job no	Arrival time	Run time
1	10.00	2.00hrs
2	10.10	1.00hrs
3	10.25	0.25hrs

Figure3.3 Sample job arrival time

Job 1 arrived at 10 AM, and the job control card estimated that it would run for two hours. we assume that the estimate is in fact how long the job runs.

If we use a First In First Out (FIFO) algorithm , the jobs will be run as depicted. Average

turnaround is computed.

Job no	Arrival Time	Start Time	finish Time	Turnaround Time
1	10.00	10.00	12.00	2.00hrs
2	10.10	12.00	13.00	2.90hrs
3	10.25	13.00	13.25	3.00hrs
Average turnaround=2.63hrs				7.90hrs

Figure 3.4 Jobs scheduled using FIFO

Job Scheduling Using Shortest Job First:

The average turnaround time using a different scheduling algorithm that runs the "hold" job with the shortest run time first.

Job no	Arrival Time	Start Time	finish Time	Turnaround Time
1	10.00	10.00	12.00	2.00hrs
2	10.10	12.25	13.25	3.15hrs
3	10.25	12.00	12.25	2.00hrs
Average turnaround=2.38hrs				7.15hrs

Figure 3.5 Scheduling jobs using shortest job first

Job Scheduling Using Future Knowledge:

Is there any way to further improve the average turnaround time? For example , would future knowledge be helpful?If at 10 AM we knew that two short jobs .

Job no	Arrival Time	Start Time	finish Time	Turnaround Time
1	10.00	11.50	13.50	3.50hrs
2	10.10	10.50	11.50	1.40hrs
3	10.25	10.25	10.50	0.25hrs
CPU ideal=.25hr				5.15hrs
Average turnaround=1.72hrs				

Figure 3.6 **Scheduling jobs using Future knowledge**

We reduce average turnaround time,but we "wasted" .25 hours of CPU time and probably made job1 an unhappy customer.

Problems in Job Scheduling Algorithms

1. When one system using an algorithm that did not run long jobs immediately closed down after five years of operation, it was rumored that jobs were found that had been lost for over three years!
2. Future knowledge is rare.
3. Run time are usually estimated approximations.
4. Other resources must be considered, such as memory requirements , i/o devices , etc..,

Measure Of Scheduling Performance:

On the other hand, if a 1-hour job is scheduled and run immediately,it has a 1-hour turnaround time. Does this mean that the 1-hour job was treated poorly? No, since it is impossible to have a turnaround time that is less than the run time.

We can define another measure,called the weigted turnaround time (W),Which is w=t/r.t is the turnaround time , as computed earlier,and R is the actual run time.

Job Scheduling In Multiprogrammed Environment:

If our policy was to minimize average Turnaround time , T ,we should always run the job with the shortest run time.

Job Scheduling With Multiprogramming but No I/O Overlap:

The results of an FIFO scheduling algorithm, assuming no multiprogramming.

Job no	Arrival time	Run time
1	10.0	0.3hr
2	10.2	0.5hr
3	10.4	0.1hr
4	10.5	0.4hr
5	10.8	0.1hr

Figure3.7: Job Arrival and Runtime

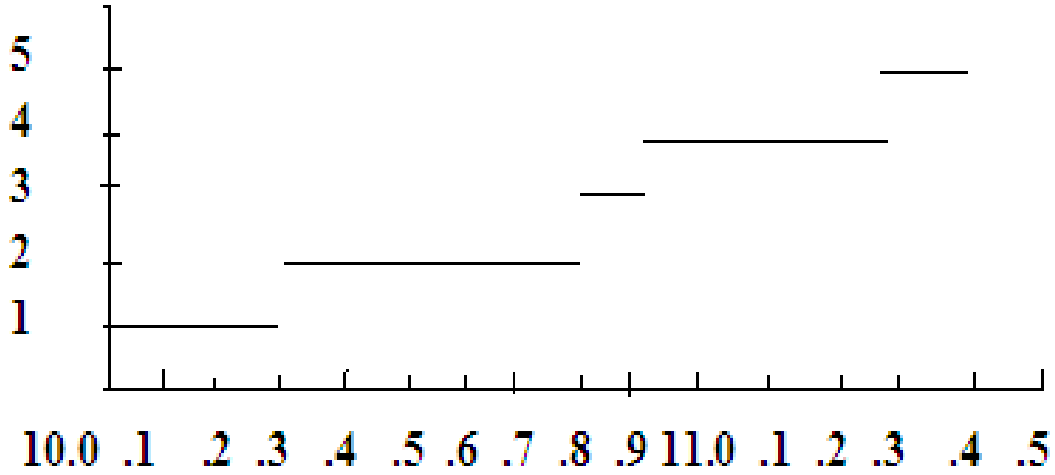


Figure:3.8 FIFO- no multiprogramming: graphic representation

Job no	Arrival Time	Start Time	finish Time	Turnaround Time	weighted Turnaround Time
1	10.0	10.0	10.3	0.3	1.00
2	10.2	10.3	10.8	0.6	1.20
3	10.4	10.8	10.9	0.5	5.00
4	10.5	10.9	11.3	0.8	2.00
5	10.8	11.3	11.4	0.6	6.00
				2.8hrs	15.20

Average turnaround, t=.56

Weighted average turn around ,w=3.04

Figure: 3.9 FIFO- NO multiprogramming: tabular representation

Headway of CPU: CPU headway is the amount of CPU time spent on a job.

If two jobs are being multiprogrammed, each job's CPU headway will be equal to half of the clock time elapsed. The availability of unlimited main memory and I/O devices; thus, a job may be started as soon as it arrives. Note that job 1 arrived at 10 AM and was to run for 3 hours. After job 1 had run for .2hours ,job2 arrived and was placed in memory

During the time segment 10.2 through 10.4, the processor was timesliced between the job 1 and job 2.

Even though job1 had only .1 hours of execution left , the processor was servicing two jobs ,and it took .2hours to complete job1.

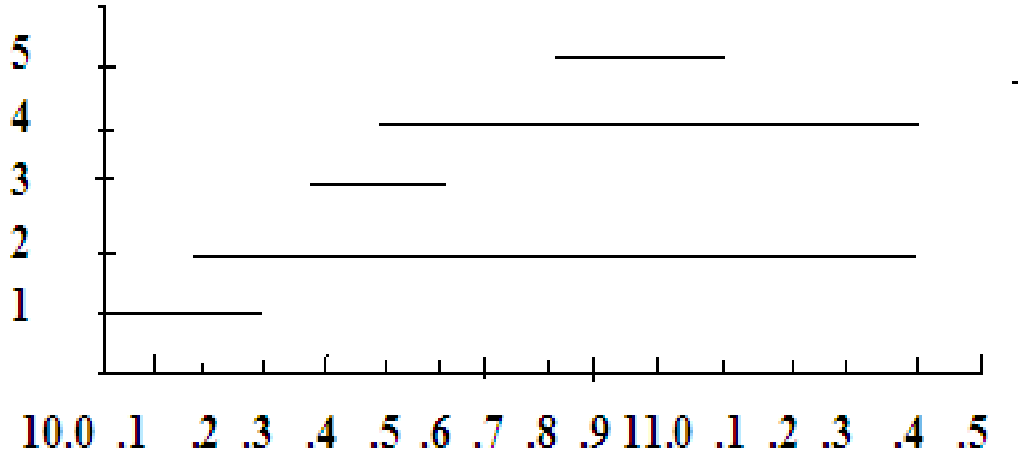


Figure 3.10: Five Jobs multiprogrammed- no I/O overlap

Time	Event	No. of Jobs	% cpu per job	Elapsed Time	Headway Per job	Job	Time Left
10.0	Job1 Arrives					1	.3
10.2	Job2 Arrives	1	1	.2	.2	1 2	.1 .5
10.4	Job3 Arrives Job 1 Terminate	2	1/2	.2	.1	1 2 3	 .4 .1
10.5	Job4 Arrives	2	1/2	.1	.05	2 3 4	.35 .05 .4
10.65	Job3 Terminate	3	1/3	.1 5	.05	2 3 4	.3 .35
10.8	Job5 Arrives	2	1/2	.15	.075	2 4 5	.225 .275 .1

Figure3.11: Calculation for figure 3.10

Job no	Run Time	Start Time	finish Time	Turnaround Time	weighted Turnaround Time
1	0.3	10.0	10.4	0.4	1.00
2	0.5	10.2	11.35	1.15	1.20
3	0.1	10.4	10.65	0.25	5.00
4	0.4	10.5	11.4	0.9	2.00
5	0.1	10.8	11.1	0.3	6.00
				3.00hrs	11.38

Average turnaround, t=0.6

Weighted average turnaround, w=2.276

Figure 3.12: FIFO with multiprogramming

Job Scheduling With Multiprogramming And I/O Overlap:

If a job were run by itself and i/o wait were 25 percent , then in 1 hour of elapsed time only .75 hours of cpu headway would take place. Thus , if the cpu time needed were 1.5 hours , it would take 2 hours of elapsed time for that job to complete .Now take the more complicated case of two jobs starting at the same time, both of which have i/o wait time of 25 percent and both of which need 1.5 hours of cpu time.

What is the total cpu headway possible in 1 hour for both jobs? It this is divided equally between the two jobs, each job gets only .48 hours of cpu headway per hour of elapsed time. Therefore , it will take 1.5 divided by .48 or 3.13 hours of total elapsed time to complete both of these jobs. This is a considered improvement over the 4 hours that would have elapsed if these jobs were run mono programmed and processed serially.

Job no	mono Program run Time	Start Time	finish Time	Turnaround Time	weighted Turnaround Time
1	0.3	10.0	10.356	0.356	1.187
2	0.5	10.2	11.05	0.850	1.700
3	0.1	10.4	10.581	0.181	1.810
4	0.4	10.5	11.158	0.658	1.645
5	0.1	10.8	11.025	0.225	2.250
				2.270hrs	8.592

Average turnaround, t=.454
Weighted turnaround, w=1.718

Figure3.13 Job Scheduling times for jobs With Multiprogramming and I/O Overlap: Summary analysis

Time Jobs	Event wait	No. of per job	cpu Time	% cpu	Elapsed Per job	job Left	Headway	Time
10.0	Job1 Arrives					1		.225
10.2	Job2 Arrives	1	25	75	.2	1 2	.15	.075 .375
10.356	Job1 finishes	2	4	48	.156	1 2	.075 .075	.3
10.4	Job3 Arrives	1	25	75	.04375	2 3	.033	.267 .0 75
10.5	Job4 Arrives	2	4	48	.1	2 3 4	.048 .048	.219 .027 .3
10.581	Job3 finish	3	0	33.3	.081	2 3 4	.027 .027 .027	.192 .273
10.8	Job5 Arrives	2	4	48	.219	2 4 5	.105 .105	.087 .168 .075

Figure:3.14JobScheduling times for jobs with multiprogramming and I/O overlap:tabular analysis

Job Scheduling With Memory Requirements and No I/O Overlap:

Time	Event	No. of Jobs	% cpu per job	Elapsed Time	Headway Per job	core available	Job	Time Left
10.0	Job1 Arrives Job 1 started					90	1	.3
10.2	Job2 Arrives Job 2 started	1	1	.2	.2	30	1 2	1 .5
10.4	Job3 Arrives Job 3 started	2	1/2	.2	.1	40	1 2	 .4
10.5	Job4 Arrives	1	1	.1	.1	30	2 4	.3 .4
10.8	Job5 Arrives	2	1/2	.3	.15	0	2 4 5	.15 .25 .1

Figure3.15 Multiprogramming core restrictions only: Tabular

Job no	Run Time	Start Time	finish Time	Turnaround Time	weighted Turnaround Time
1	0.3	10.0	10.4	0.4	1.33
2	0.5	10.2	11.2	1.0	2.0
3	0.1	10.4	11.4	1.0	10.0
4	0.4	10.5	11.4	0.9	2.25
5	0.1	10.8	11.1	0.3	3.0
				3.6hrs	18.58

Average turnaround, t=.72

Weighted turnaround, w=3.716

Figure3.16: Multiprogramming core restrictions only: Summary results

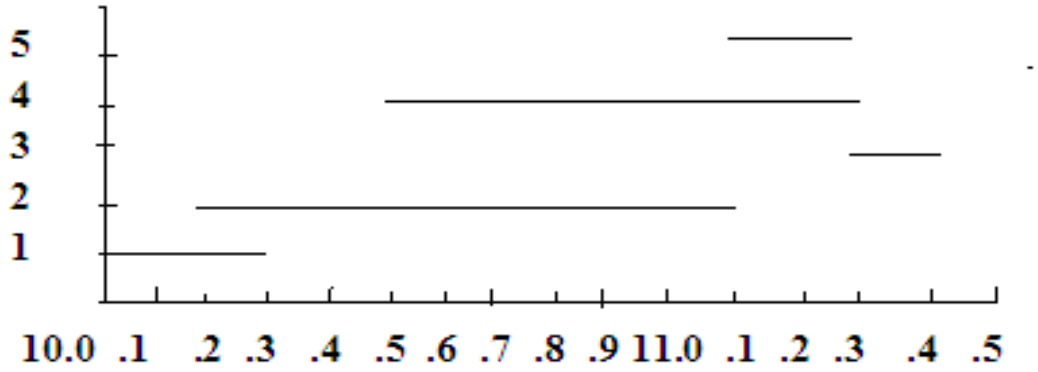
JOB SCHEDULING WITH MEMORY AND TAPE CONSTRAINTS AND NO I/O OVERLAP:

Let us now assume that the jobs also require a number of tape drives in addition to memory and cpu time . what is the impact on turnaround time now?

The job sample requests are indicated that the system has 100k memory and five tape drives.

Job no	Time Submitted	Run Time	Memory Needs	Tapes Needed
1	10.0	0.3hrs	10k	2
2	10.2	0.5hrs	60k	1
3	10.4	0.1hrs	50k	4
4	10.5	0.4hrs	10k	2
5	10.8	0.1hrs	30k	3

Figure3.17: Sample jobs with memory and tape needs



Time

x-Arrival time of job

Figure3.18: Multiprogramming with memory and tape needs: Graphical

Time	Event	No. of Jobs	% cpu per job	Elapsed Time	Headway Per job	core available	tape available	Job	Time Left
10.0	Job1 Arrives Job 1 started					90	3	1	.3
10.2	Job2 Arrives Job 2 started	1	1	.2	.2	30	2	1 2	1 .5
10.4	Job3 Arrives Job 3 started	2	1/2	.2	.1	40	4	1 2	 .4
10.5	Job4 Arrives	1	1	.1	.1	30	2	2 4	.3 .4
10.8	Job5 Arrives	2	1/2	.3	.15	0	2	2 4	.15 .25

Figure3.18: Multiprogramming with memory and tape needs: tabular

Job no	Run Time	Start Time	finish Time	Turnaround Time	weighted Turnaround Time
1	0.3	10.0	10.4	0.4	1.33
2	0.5	10.2	11.1	0.9	1.80
3	0.1	10.4	11.4	1.0	10.0 0
4	0.4	10.5	11.3	0.8	2.00
5	0.1	10.8	11.3	0.5	5.0
				3.6hrs	20.13

Average turnaround, t=.72

Weighted turnaround, w=4.026

Figure3.19: Multiprogramming with memory and tape needs: Summary needs

Process Scheduling:

Once the job scheduler has selected a collection of jobs to run, the process scheduler attempts to handle the microscopic scheduling(i.e., the dynamic assignment of processor b to process).

The process scheduler is also called the dispatcher or low-level scheduler in the literature.

In a multiprogrammed environment, a process typically uses the processor for only 100 ms or less before becoming blocked to wait for I/O completion or some other event.

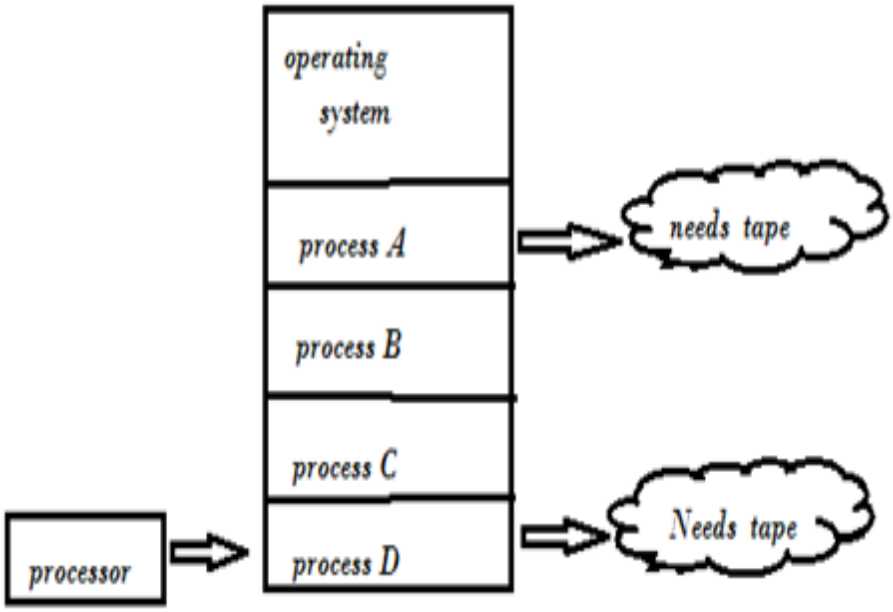


Figure3.20single processor interlock

The job and process scheduler may interact .

The process scheduler may choose to postpone, or roll out ,a process and require that it go through the macro-level job scheduling agai n in order to complete.

This is especially true in a timesharing system.

Functions:

The process scheduler must perform the following functions:

1) Keep track of the state of processes.
2) decide which j process gets a processor ,when ,and for how long.
3) allocate processors to processes.
4) deallocate processors from processes.

Policies:

- The scheduling policy must decide which process is to be assigned a aprocessor and for how long.
- The process is complete.
- The process becomes blocked.
- A higher priority process needs the processor.
- A time quantum has elapsed.
- An error occurs.

Typical process scheduling policies include the following:

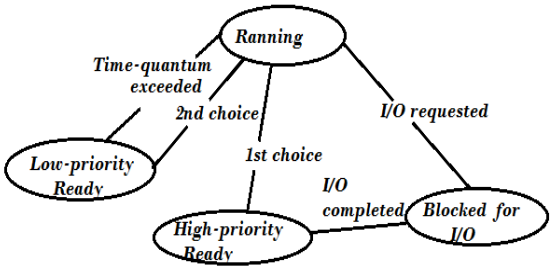
1) Round robin
2) Inverse of remainder of quantum
3) Multiple-level feedback variant on round robin
4) Priority
5) Limited round robin
6) System balance
7) Preferred treatment to "interactive" jobs
8) Merits of the job

Process States Diagrams for Scheduling:

A possible scheduler policy may be:

1) To select a process to run from the high-priority ready list
2) If there are no processes in high-priority ready, to select a process from the low-priority ready list.

Figure3.21: A set of scheduling state transitions:



A set of scheduling state transitions

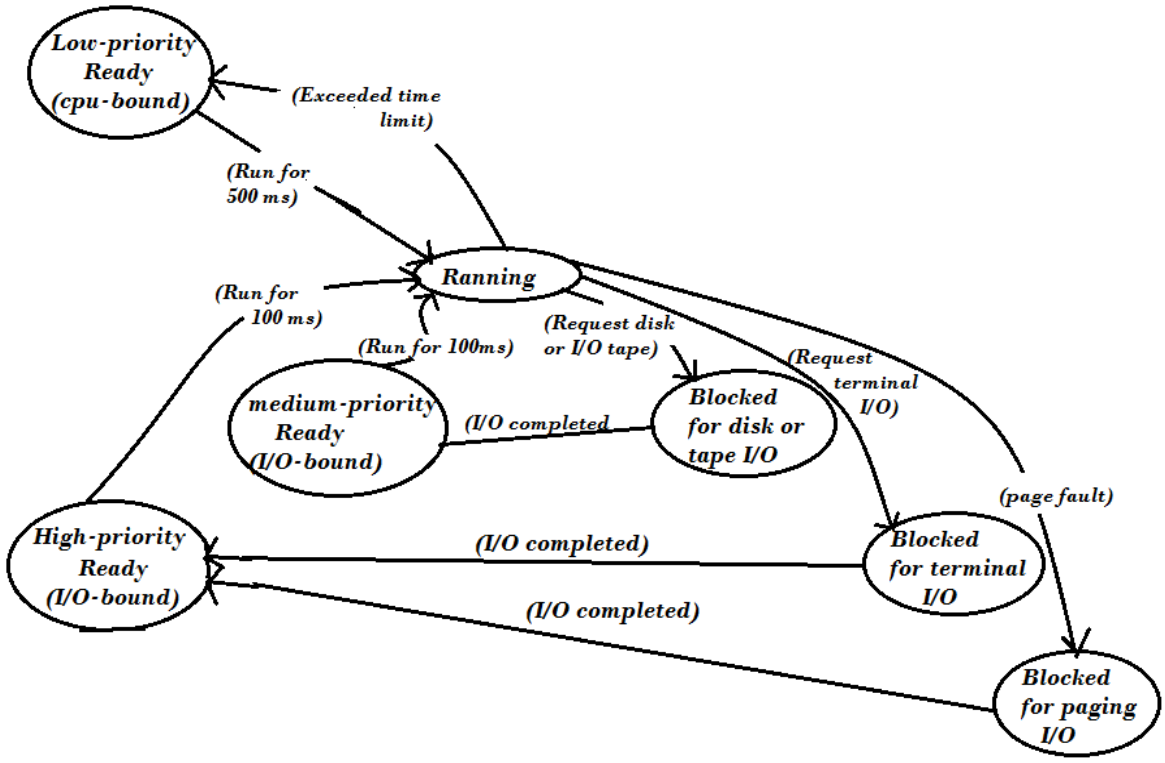


Figure3.22: More elaborate scheduling state transitions

Evaluation of Round Robin Method:

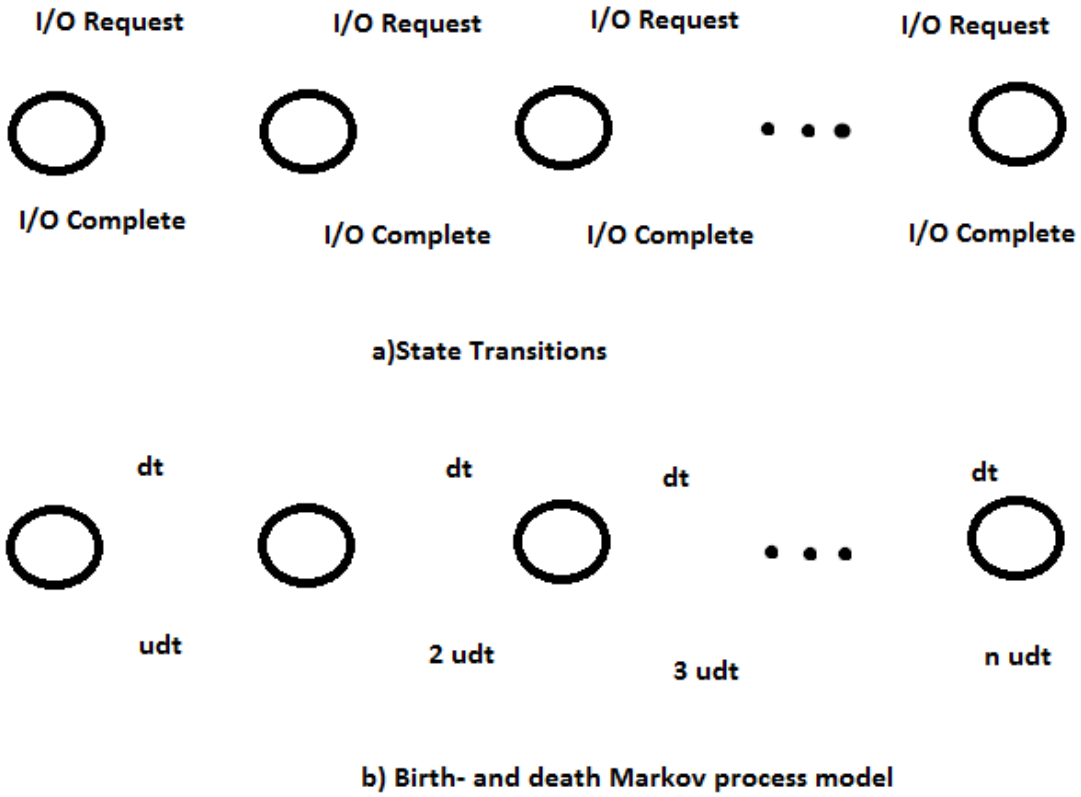


Figure 3.22 Model of System State

Process Synchronization:

The problem of process synchronization arises from the need to share resources in a computing system.

This sharing requires coordination and cooperation to ensure correct operation.

The cashier processes these bags, one at a time, receiving money from you and giving you your order.

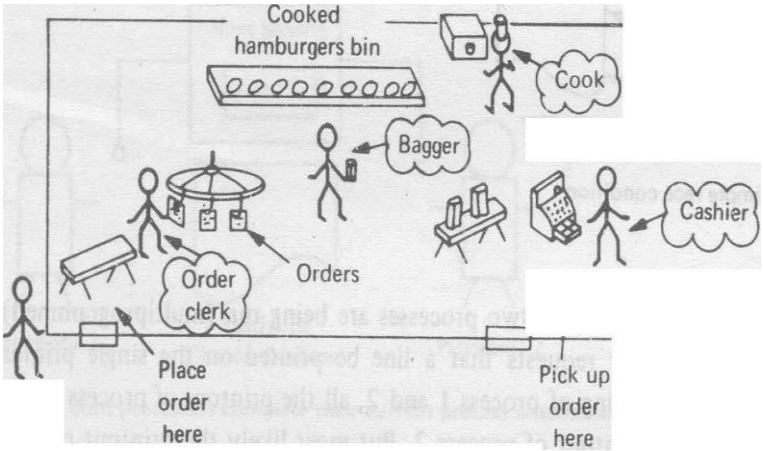


Figure3.24operation of a hamburger stand

The only communication is via certain shared areas: the order list turntable, the cooked hamburger bin, and the checkout counter.

Race condition:

The operations request and release are usually part of the operating system facilities handled by the traffic controller.

If a process requests a resource that is already in use, the process automatically becomes blocked.

When the resource becomes available as a result of a later release ,the blocked process may be assigned the resource and made ready.

Alternate solutions(e.g., spooling).

A race condition occurs when the scheduling of two processes is so critical that the various orders of scheduling them result in different computations. Race conditions result from the explicit or implicit sharing of data or resources among two or more processes. **Figure 3.23** illustrates a simple form of race.

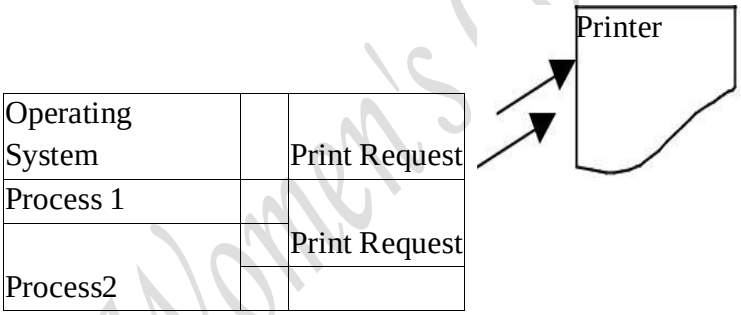


Figure 3.24 Simple race condition

Synchronization Mechanisms

Various synchronization mechanisms are available to provide interprocess co-ordination and communication. In this section several of the most common tech-niques will be briefly presented; the references provide more detail and additional alternatives.

Next Serving

When current process is completed or postponed:

1. current process in ready list

2. Mark it "complete" or "postponed"
3. Note "next serving" number (17)
4. Find processes 17 in ready list
5. Make note of process to be done
6. Adjust ready list for "next serving" number (18)

Figure 3.24 Scheduling (multiple processors)

1. Processor 1 finds its current process in ready list
2. Processor 2 finds its current process in ready list
3. Processor 1 marks its current process
4. Processor 2 marks its current process
5. Processor 1 notes "next serving" (17)
6. Processor 2 notes "next serving" (17)
7. Processor 1 finds process 17 in ready list and notes it
8. Processor 2 finds process 17 in ready list and notes it
9. Processor 1 adjusts ready list "next serving" (18)
10. Processor 2 adjusts ready list "next serving" (19)

Now both processors are simultaneously working on the same process and process 18 has been skipped.

Set (TS) instruction that performs both steps 1 and 2 as an indivisible operation. There are similar instructions available on most contemporary computers. The reader should convince

himself that the lock and unlock operations do, in fact, prevent race conditions. For example, consider the three processes has been skipped.

Lock (X)	UnLock (X)
TS X	MVI X,00
	BNZ *- 4

Processor Management / 37

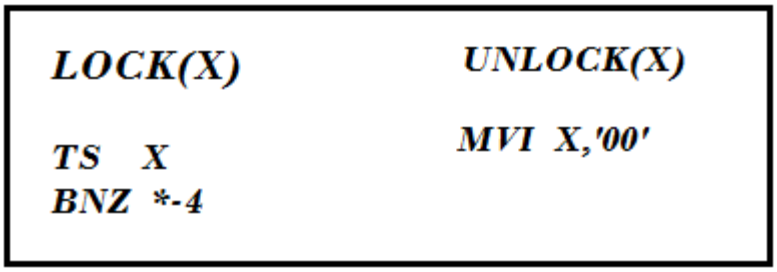
PROCESS 1	PROCESS 2	PROCESS 3
Lock (XI	LOCK (XI	LOC (XI
(critical		
: database	: 1...1	: (...)
manipuLationsl		
UNLOCK (XI	UNLOCK (XI	UNLOCK (XI

Synchronization Mechanisms:

Various synchronization mechanisms are available to provide interprocess co-ordination and communication.

Test-And-Set Instruction:

In most synchronization schemes ,a physical entity must be used to represent the resource.
This entity is often called a lock byte or semaphore.
Thus, for each shared database or device there should be a separate lock byte.
We will use the convention that lock byte=0 means the resource is available, whereas lock byte=1 means the resource is already in use.
Before operating on such a shared resource ,a process must perform the following actions:
1) Examine the value of the lock byte (either it is 0 or 1).
2) Set the lock byte to 1.
3) If the original value was 1, go back to step1.



LOCK(X) :

1. Examine the value of the lock byte (either it is 0 or 1).
2. Set the lock byte to 1.
3. If the original value was 1, call WAIT(X).

UNLOCK(X) :

1. Set the lock byte to 0.
2. Call SIGNAL(X).

WAIT and SIGNAL are primitives of the traffic controller component processor management.

A WAIT (X) sets the process PCB to the blocked state and links it to the lock byte X.

The WAIT and SIGNAL mechanisms can be used for other purposes, such as waiting for I/O completion.

After an I/O request (i.e., SIO instruction) has been issued, the process can be blocked by a WAIT(X) when the I/O completion interrupt occurs, it is converted into a SIGNAL (X).

P AND V OPERATIONS ON COUNTING SEMAPHORES:

- Depending upon the initial value of the semaphores and the number of semaphores used, P and V can be used for many purposes.
- If one semaphore is used and its initial value is 1,P(S) and V(S) are identical to **WAIT(S) and SIGNAL(S).**
- The producer must be prevented from placing an item in a full buffer and the consumer must be prevented from removing an item from an empty buffer.
- By using two semaphores, initially set as S1=n and S2=o, we can synchronize the processes.

Message Communication:

- The synchronization mechanisms we have discussed provide communication between processes indirectly by means of a shared buffer area and a shared a lock byte or

semaphore.

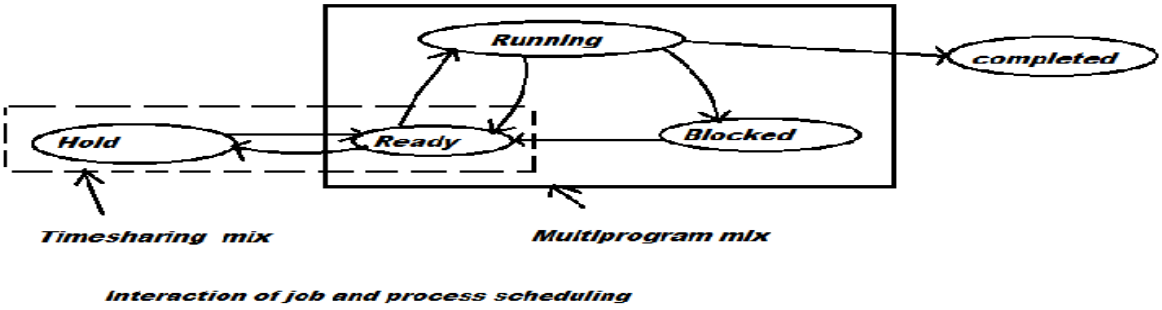
- Situations such as the producer /consumer problem can also be solved by providing direct process-to-process communication by means of the primitives.
- SEND (Pr , M) and RECEIVE (Ps ,M) where Pr and Ps are the names of processes and M is a message (i.e., a k- byte character string).
- SEND (Pr ,M) saves the message M for the receiver process Pr .RECEIVE (Ps ,M) returns a message M, if there is one, to the requestor process; it also returns the name of the process Ps that sent the message. if there are no message for the RECEIVE requestor, it becomes blocked until a message is sent to it.
- The buffering of messages and synchronization are handled automatically by the SEND and RECEIVE primitives.

Combined Job and Process Scheduling:

- In many instances, the job scheduling and process scheduling algorithms must interact closely.
- If we attempt multiprogramming too many processes at the same time, especially in a demand paging system, all the processes will suffer from poor performance due to an overloading of available system resources(this phenomenon, called thrashing, is analyzed in Chapter 8).On the other hand, in a conversational timesharing system, every user wants a "crack at the machine" as soon as possible__1-hour waits will not be tolerated.
- As a compromise, we can adopt the philosophy that it is far better to give a few jobs at a time a chance than to starve them all! For example, we may allow two to five users to be ready at a time.

The basic concepts here are:

(1) Multi program "in the small" for two to five jobs, process- scheduling every few ms; and
(2) Timeshare "in the large," where every few 100 ms or seconds one ready job is placed in hold status and one of the hold status jobs is made ready.



Interaction of job and process scheduling

Figure 3.26 Interaction of job and process scheduling

- Thus, over a period of a minute or so, every job has had its turn, and furthermore, the system runs fairly efficiently.

Unit III Completed

Unit IV

Device Management Types of Devices-Sequential Access Storage Media-Direct Access Storage DevicesMagnetic Disk Drive Access Times- Components of the I/O SubsystemCommunication among Devices-Management of I/O Requests

4.1 Introduction:

This chapter focuses on the management of I/0 devices. These include actual 1/0 devices, such as printers, card readers, tapes, disks, and drums, and supporting devices, such as control units or control channels. Some readers may already be familiar with some of these devices; our purpose in these sections is to merely give a feel for the relative speeds and characteristics of the important devices. A large installation may devote over half the system cost to I/O devices. Therefore, it is desirable to use these devices in an efficient manner.

The basic functions of device management are:

1. Keeping track of the status of all devices, which requires special mechanisms. One commonly used mechanism is to have a database such as a Unit Control Block (UCB) associated with each device.
2. Deciding on policy to determine who gets a device, for how long, and when. A wide range of techniques is available for implementing these policies. For example, a policy of high device utilization attempts to match the non uniform requests by processes to the uniform speed of many I/0 devices. There are three basic techniques for implementing the policies of device management:

   A. Dedicated-a technique whereby a device is assigned to a single process.

   B. Shared-a technique whereby a device is shared by many processes.

   C. Virtual-a technique whereby one physical device is simulated on another physical device.

   3. Allocation-physically assigning a device to process. Likewise the corresponding control units and channels must be assigned.

4. Deallocation policy and techniques. De allocation may be done on either a process or a job level. On a job level, a device is assigned for as long as the job exists in the system. On a process level, a device may be assigned for as long as the process needs it.

This chapter will focus mainly on the policies of device management, since the techniques are heavily device-dependent and are changing rapidly with hardware advances.

The module that keeps track of the status of devices 'is called the I/O traffic controller. We have called all modules associated with the operation of a single device the I/O device handlers.

The I/O device handler's function is to create the channel program for performing the desired function, initiate i/0 to that device, handle the interrupts from it, and optimize its performance. In short, the device handler performs the physical 1/0. The I/O scheduler decides when an I/O processor is assigned to a request and sets up the path to the device.

4.2 Techniques for Device Management

Three major techniques are used for managing and allocating devices: (I) dedicated, (2) shared, and (3) virtual.

Dedicated Devices:

A dedicated device is allocated to a job for the job's entire duration. 30me devices lend themselves to this form of allocation. It is difficult, for example, to share a card reader, tape, or printer. If several users were to use the printer at the same time, would they cut up their appropriate output and paste it together? Unfortu-nately, dedicated assignment may be inefficient if the job does not fully and con-tinually utilize the device. The other techniques, shared and virtual, are usually preferred whenever they are applicable.

Shared Devices

Some devices such as disks, drums, and most other Direct Access Storage Devices (DASD) may be shared concurrently by several processes. Several processes can read from a single disk at essentially the same time. The management of a shared device can become quite complicated, particularly if utmost efficiency is desired. For example, if two processes simultaneously request a Read from Disk A, some mechanism must be employed. ` To determine which request should be handled first. This may be done partially by software (the I/O scheduler and traffic controller) or entirely by hardware (as in some computers with very sophisticated channels and control units).

Policy for establishing which process' request is to be satisfied first might be based on (1) a priority list or (2) the objective of achieving improved system out-put (for example, by choosing whichever request is nearest to the current position of the read heads of the disk).

Virtual Devices

Some devices that would normally have to be dedicated (e.g., card readers) may be converted into shared devices through techniques such as Spooling. For example, a Spooling program can read and copy all card input onto a disk at high speed. Later, when a process tries to read a card, the Spooling program intercepts the request and converts it to a read from the disk. Since several users may easily share a disk, we have converted a dedicated device to a shared device, changing one card reader into many "virtual" card readers. This technique is equally applicable to a large number of peripheral devices, such as teletypes, printers, and most dedicated slow input/output devices.

Generalized Strategies

Some professionals have attempted to generalize device management even more than is done here. For example, the call side of Spooling is similar to the file sys-tem and buffering bears striking similarity to Spooling. We could generalize even more and view memory, processors, and I/O devices as simple devices to which the same theory should apply. At present

we realize that there are practical limitations to these generalizations, although at some time in the future more general theories may emerge that are applicable to all devices and that have direct practical applications.

Device Characteristics-Hardware Considerations

Almost everything imaginable can be, and probably has been, used as a peripheral device to a computer, ranging from steel mills to laser beams, from radar to mechanical potato pickers, from thermometers to space ships. Fortunately, most computer installations utilize only a relatively small set of peripheral devices. Peripheral devices can be generally categorized into two major groups: (1) input or output devices and (2) storage devices.

Input or Output Devices

An input device is one by which the computer "senses" or "feels" the outside world. These devices may be mechanisms such as thermometers or radar devices, but, more conventionally. They are devices to read punched cards, punched paper tape, or messages typed on typewriter like terminals. An output device is one by which the computer "affects" or "controls" the outside world. It may be a mechanism such as a temperature control knob or a radar direction control, but more commonly it is a device to punch holes in cards or paper tape, print letters and numbers on paper, or control the typing of typewriter like terminals.

Storage Devices

A storage device is a mechanism by which the computer may store information (a procedure commonly called writing) in such a way that this information may be retrieved at a later time (reading). Conceptually, storage devices are analogous to human storage mechanisms that use pieces of paper, pencils, and erasers.

It is helpful to differentiate among three types of devices. Our differentiation is based on the variation of access times (T_{ij}) where:

T_{ij} = time to access item j, given last access was to item i (or current position is item i)

We differentiate the following types of storage devices:

1. Serial Access' where T_{ij} has a large variance (e.g., tape)
2. Completely Direct Access where T_{ij} is constant (e.g., core)
3. Direct Access where T_{ij} has only a small variance (e.g., drum)

Serial Access Device

A serial access storage device can be characterized as one that relies on strictly physical sequential positioning and accessing of information. Access to an arbitrary, stored item requires a "linear search" so that an average access takes the time required to read half the information stored. A Magnetic Tape Unit (MTU) is the most common example of a serial access storage

device. The MTU is based upon the same principles as the audio tape deck or tape cassette, but instead of music or voices, binary information is stored. A typical serial access device is depicted in Figure 4.1.

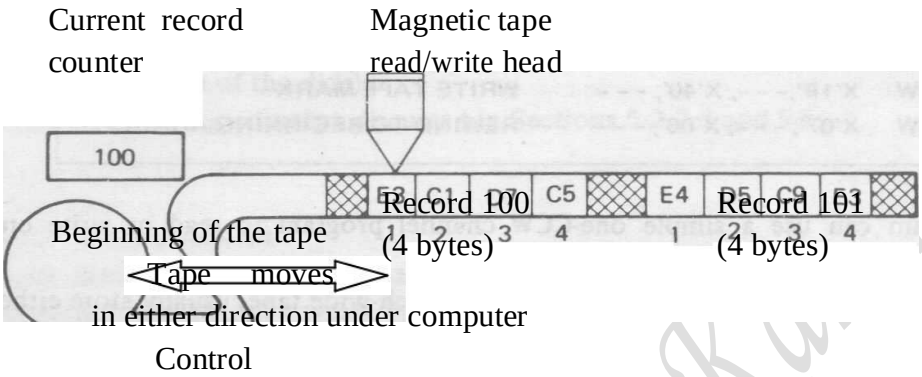


Figure 4.1 Magnetic tape unit.

Information is usually stored as groups of bytes, called records, rather than single bytes. In general, a record can be of arbitrary length, e.g., from 1 to 32,768 bytes long. In Figure 3.1 all the records are four bytes long. Records may be viewed as being analogous to spoken words on a standard audio tape deck. Each record can be identified by its physical position on the tape; the first record is 1, the second, 2, and so on. The current record counter of Figure 3.1 serves the same function as the "time" or "number-of feet" meter of audio tape decks. If the tape is positioned at it's beginning (i.e. completely rewound), and we wish to read record number 101, it is necessary to skip over all interviewing records (e.g., I, 2, . . .99, 100) in order to reach record number 101.

In addition to obvious I/O commands, such as read the Next Record or Write the Next Record, a tape unit usually has commands such as read the Last Record (read backward), Rewind to Beginning of Tape at High Speed, Skip Forward (or Backward) One Record without Actually Reading or Writing Data. In order to help establish groupings of records on a tape, there is a special record type called a tape mark. This record type can be written only by using the Write Tape Mark command. If the tape unit encounters a tape mark while reading or skipping, a special interrupt is generated. Tape marks are somewhat analogous to putting bookmarks in a book to help find your place.

Completely Direct Access Devices

A completely direct access device is one in which the access time Tij is a constant. Magnetic core memory, semiconductor memory, read-only wired memories, and diode matrix memories are all examples of completely direct access memories. One property of a magnetic core is that when it changes magnetic state it induces a current would decompose the address:

Most contemporary computers use semiconductor (transistor like) memories. The basic ideas are similar to the core memories described above, but semi-conductor memories offer advantages in cost and speed. Typical semiconductor memories operate between 50 to 500 nanoseconds access time.

Although we say this type of storage device may be shared on the nanosecond level, it is

limited in the sense that only one processor (CPU or I/O channel) can read a specific location (or block of locations) at a given time. Whenever more than one processor is accessing memory, especially if there is one CPU and several I/O channels, we frequently say that the channels are memory cycle stealing. Memory cycle stealing can degrade the performance of a system and is a factor with which the I/O traffic controller must contend.

Direct Access Storage Devices (DASD)

A direct access device is one that is characterized by small variances in the access time Tij' these have been called Direct Access Storage Devices. Although they do not offer completely direct access, the name persists in the field.

In this section we give two examples of DASD, fixed head and moving-head devices.

Fixed-Head Drums and Disks

A magnetic drum can be simplistically viewed as several adjacent strips of mag-netic tape wrapped around a drum so that the ends of each tape strip join. Each tape strip, called a track, has a separate read/write head. The drum continuously revolves at high speed so that the records repeatedly pass under the read/write heads (e.g., record I, record 2, then record I again, record 2, . . .etc.). A track number and then a record number identify each individual record. Typical magnetic drums spin very fast and have several hundred read/write heads; thus a random access to read or write can be accomplished in 5 or 10 ms in contrast to the minute required for random access to a record on a full magnetic tape.

Moving-head Disks and Drums

The magnetic disk is similar to the magnetic drum but is based upon the use of one or more flat disks, each with a series of concentric circles, one per read/write head. This is analogous to a phonograph disk. Since read/write heads are expensive, some devices do not have a unique head for each data track. Instead, the heads are physically moved from track to track; such units are called moving-arm or moving-head DASDs. Each arm position is called a cylinder.

In order to identify a particular record stored on the moving-head DASD shown in Figure 3.4, it is necessary to specify the arm position, track number, and record number. Notice that arm position is based upon radial movement whereas record number is based upon circumferential movement. Thus, to access a record, it is necessary to move to the correct radial position (if not already there) and then to wait for the correct record to rotate under the read/write head.

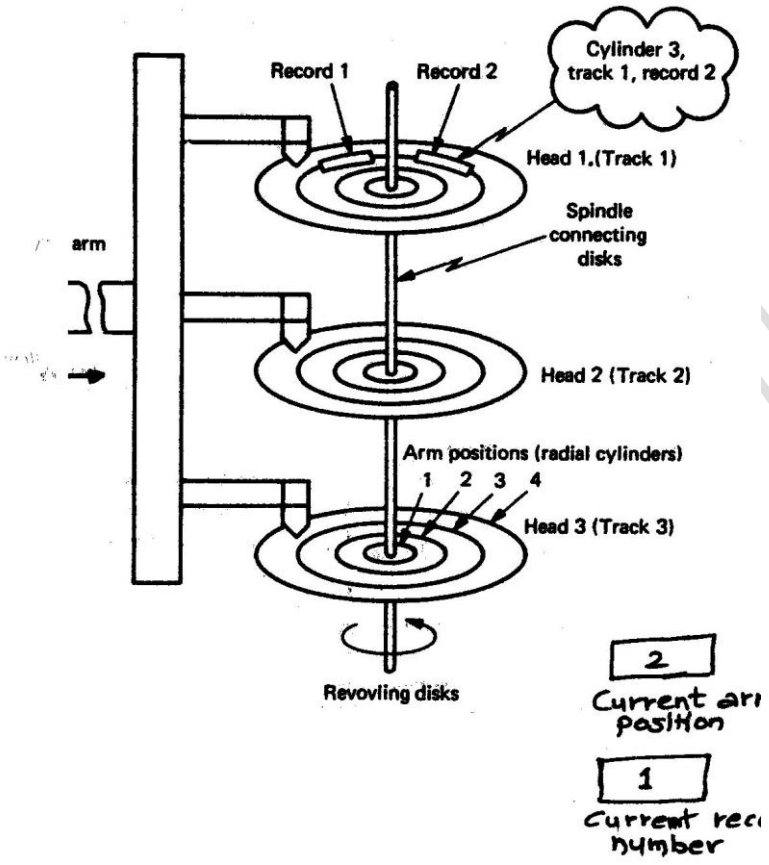


Figure 4.2 Moving-head disks DASD

General Characteristics

Some DASD devices allow keys to be associated with each record. The keys are assigned by the user and can denote the contents of a record. The controller for such devices allows commands to read or write a record addressed not by a track, arm, or cylinder, but rather by its key.

There are often some constraints on DASDs that allow this form of addressing, e.g., the read head must already be positioned on the right cylinder.

Given the present trend toward reductions in the price of CPUs, it is probably better to keep a table in memory and not to use such searching on the disks, thus keeping the cost of the channels down.

The preceding examples are representative of storage devices, since in spite of numerous variations, all have similar characteristics.

Channels And Control Units

Device management must also take responsibility for the channels and control units. The channels are the special processors that execute the channel programs (i.e., the CCWs). Due to the high cost of channels, there are usually fewer channels than devices. The channels must, therefore, be switched from one device to another. Further cost savings can be obtained by taking advantage of the fact that relatively few I/O devices are in use at one time. Thus, it is wise to separate the device's electronics from its mechanical components. The electronics for several devices can be replaced by a single control unit, which can be switched from one device to another. Figure 3.6 illustrates the relationship between channels, control units, and devices. Typically, there are up to eight control units per channel and up to eight devices of the same type per control unit.

In order to initiate an I/O operation, a path is required between memory and the device. For example, to access drum J requires the use of channel A and drum con-trol unit D. If an I/O operation to drum G were in progress, it would not be possi-ble to initiate I/O on drum J because channel A would not be available (the I/O on drum G requires use of channel A and control unit CU).The device management routines must keep track of the status of,

(1) channels,
(2) control units, and
(3) devices.

Since the channels and control units are usually scarce, they frequently create a serious bottleneck. Several hardware techniques can be used to lessen this problem: independent device operation, device buffering, multiple paths, and block multiplexing.

Independent Device Operation

In many cases there is sufficient capability in the device to complete an I/O operation without further assistance from the channel or control unit. A disk seek operation, which might require 50 ms, only needs the channel long enough to transmit the seek address from memory to the disk-10 or 20 ms. The seek can be completed by the device alone. Likewise, there are operations that may require both the device and the control unit but not the channel for completion.

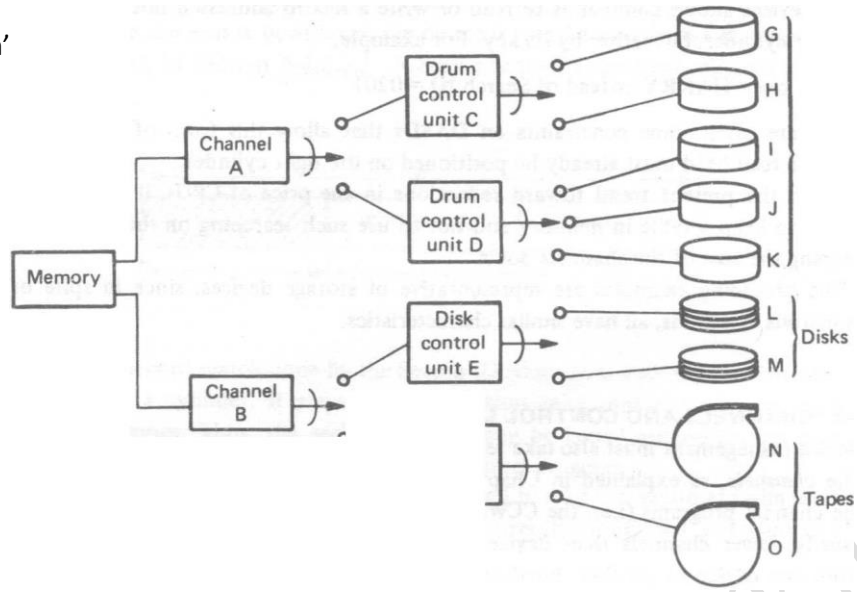


Figure 4.3 I/O Configuration of a computer System

When an I/O operation is initiated, the entire path must be available. In order to indicate when parts of the path become available, there may be separate I/O com-pletion interrupts: channel end, control unit end, and device end. The status bits in the Channel Status Word designate the type of I/O completion interrupt. If more than one part of the path becomes available at the same time, there will be only one interrupt but the status bits will indicate all available parts (e.g., both the channel end and control unit end bits will be on in the CSW).

If a channel end occurs, the channel is no longer busy, although the device may still be busy. An I/O operation can then be initiated on some other device using that channel.

Buffering

By providing a data buffer in the device or control unit, devices that would ordinarily require the channel for a long period can be made to perform independently of the channel. For example, a card reader may physically require about 60 ms to read a card and transfer the data to memory through the channel.

However, a buffered card reader always reads one card before it is needed and saves the 80 bytes in a buffer in the card reader or control unit. When the channel requests that a card be read, the contents of the SO-byte buffer are transferred to memory at high speed (e.g., 100 / ms) and the channel is released. The device then proceeds to read the next card and buffer it independently. Card readers, cardpunches, and printers are frequently buffered.

Multiple Paths

We could reduce the channel and control unit bottlenecks by buying more channels and control units. A more economical alternative, however, is to use the channels and control units in a more flexible manner and allow multiple paths to each device. Figure 3.7 illustrates such an I/O configuration. In this example there are four different paths to Device E:

1. Channel A - Control Unit C - Device E

2. Channel A - Control Unit D - Device E

3. Channel B -Control Unit, C - Devices E

4. Channel B -Control Unit D - Devices E

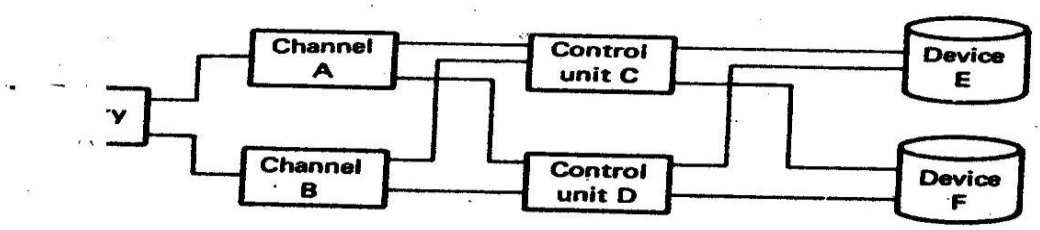


Figure 4.4 I/O configuration with multiple paths

If channel A and control unit C were both busy due to I/O for device F, a path could be found through channel B and control D to service device E. An actual I/O configuration may have eight control units per channel and eight devices per control unit rather than the sparse configuration. A completely flexi-ble configuration would make all control units accessible to all channels and all devices accessible to all control units, thus providing possibly 256 alternate paths between memory and each device. For economic reasons, this extreme case, though feasible, is seldom found.

In addition to providing flexibility, a multiple path I/O configuration is desirable for reliability. For example, if channel a malfunction and must be repaired, all the control units and devices can still be accessed through channel B. It is the responsi-bility of the device management routines to maintain status information on all paths in the I/O configuration and find an available' route to access the desired device.

Block Multiplexing

On conventional 370 selector channels, the techniques of independent device opera-tion and buffering are applicable only to the last CCW of a channel program. The channel and control unit remain connected to the device for the duration of the channel program. A 370 block multiplexor channel can be servicing multiple channel programs at the same time (e.g., eight channel programs). When the channel, encounters an I/O operation such as a buffered or independent device operation that does not need the channel for a while, it automatically switches to another channel pro-gram that needs immediate servicing. In essence, a block multiplexor channels represents a hardware implementation of multiprogramming for channel programs.

This type of channel multiprogramming could be simulated on a selector channel by making all channel programs only one command long. The device management software routines could then reassign the channel as necessary. However, this approach is not very attractive unless the central processor is extremely fast, since the channel switching must be done

very frequently and very quickly.

I/O Traffic Controller, I/O Scheduler, I/O Device Handlers
Functions:

The functions of device management can be conveniently divided into three parts:

1. I/O traffic controller-keeps track of the status of all devices, control units, and channels.

2. I/O scheduler-implements the policy algorithms used to allocate channel, control unit, and device access to I/O requests from jobs.

3. I/O devices handlers-perform the actual dynamic allocations, once the I/O scheduler has made the decision, by constructing the channel program, issuing the start I/O instruction, and processing the device interrupts. There is usually a separate device handler for each type of device.

I/O Traffic Controller:

Whereas the I/O scheduler is mainly concerned with policies (e.g., who gets the device and when), the I/O traffic controller is primarily concerned with mechan-ics (e.g., can the device be assigned?). The traffic controller maintains all status information.

The I/O traffic controller becomes complicated due to the interdependencies introduced by the scarcity of channels and control units, and the multiple paths The traffic controller attempts to answer at least three key questions:

1. Is there a path available to service an I/O request?

2. Is more than one path available?

3. If no path is currently available, when will one be free?

In order to answer these questions, the traffic controller maintains a database reflecting status and connections. This can be accomplished by means of Unit Control Blocks (UCB),

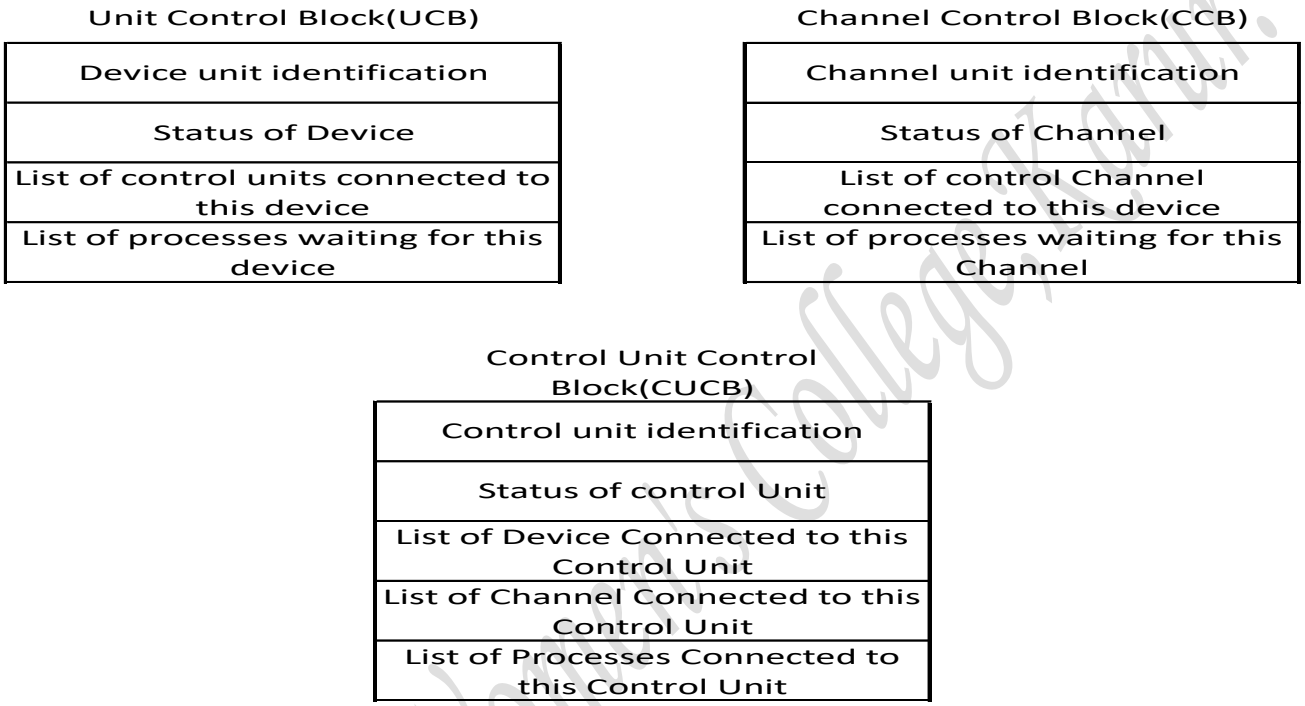


Figure 4.5 control Block

Control Unit Control Blocks (CUCB), and Channel Control Blocks (CCB) as depicted in Figure 4.5. The first question can be answered by starting at the desired UCB and working back through the connected control units and channels trying to find a combination that is available. In a similar manner it is possible to determine all available paths. This second question may be important, especially when the I/O configuration is not symmetric, as shown in Figure 3-9. Since choosing one path may block out other I/O requests. Thus, one path may conflict with fewer I/O requests than another path. In Figure 4.6, there may be up to five different paths to device H, yet only one possible path for either device G or I.

Under heavy I/O loads, it is quite likely that there will not be a path available at the time an I/O request is issued by a process. When an I/O completion interrupt occurs, one or more components (i.e., device, control units, and/or channel) become available again. We could try all

I/O requests that are waiting to see if any can now be satisfied; if, however, hundreds of I/O requests were waiting, this could be a time- consuming approach. A more efficient approach would be to list all I/O requests that are "interested" in the component in the control block, as suggested in Figure 4.5. Only those particular I/O requests would then have to be examined.

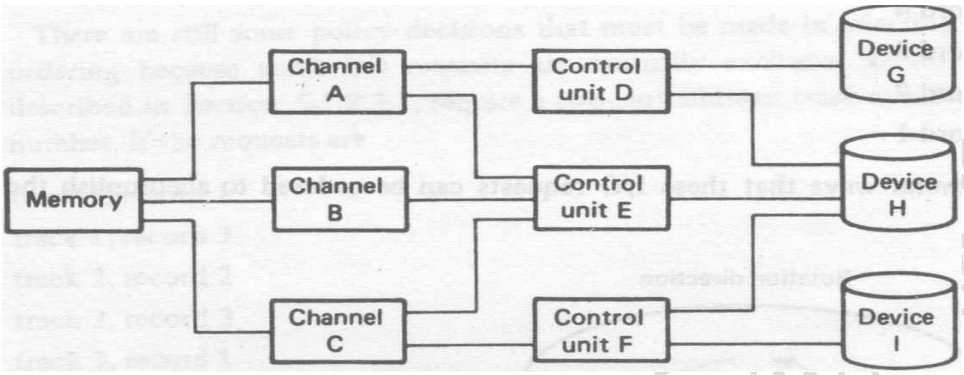


Figure 4.6 Asymmetric I/O Configurations

I/ O Scheduler:

If there are more I/O requests pending than available paths, it is necessary to choose which I/O requests to satisfy first. Many of the same concerns discussed in process scheduling are applicable here. One important difference is that I/O requests are not normally timesliced, that is, once a channel program 'has been started it is not usually interrupted until it has been completed. Most channel programs are quite' short and terminate within 50-100 ms; it is, however, possible to encounter channel programs that take seconds or even minutes before termination. Many different policies may be incorporated into the I/0 scheduler. For example, if a process has a high priority assigned by the process scheduler, it is reasonable also to assign a high priority to its I/O requests. This would help complete that job as fast as possible. Once the I/O scheduler has determined the relative orderings of the I/O requests, the I/O traffic controller must determine which, if any, of them can be satisfied. The I/O device handlers then provide another level of device- dependent I/O scheduling and optimization.

I/ O Device Handlers

In addition to setting up the channel command words, handling error conditions, and processing I/O interrupts, the I/O device handlers provide Operating Systems / detailed scheduling algorithms that are dependent upon the peculiarities of the device type. There is usually a different device handler algorithm for each type of I/O device.

Rotational Ordering

Under heavy I/O loads, several I/O requests may be waiting for the same device. As noted earlier, there may be significant variations in the Tij access time for a device. Certain orderings of I/O requests may reduce the total time required to. Service the I/O requests. Consider the drum like device that has only four records stored in it (see Figure 4.7). The

following four I/O requests have been received and a path to the device is now available:

1. Read record 4
2. Read record 3
3. Read record 2
4. Read record 1

There are several ways that these I/O requests can be ordered to accomplish the same result

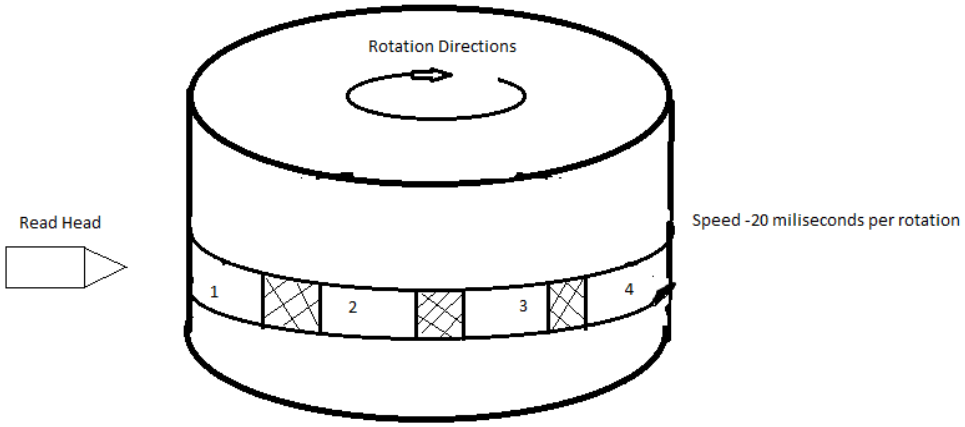


Figure 4.7 Simple drum like device illustrating rotational order

Ordering A:

If the I/O scheduler satisfies the request by reading record 4, 3, 2, I, then the total processing time equals T?4 + T 43 + T 32 + T 21 "=" 3 revolutions (3/4 + 3 X 3/4 rotations), which equals approximately 60 ms. For the reading of record 4, since we do not know the current position of the drum, we assume ½ revolution (10 ms) to position it plus 1/4 revolution (5 ms) to read it.

Ordering B

If the I/O scheduler satisfies the request by reading record 1, 2, 3, 4, then the total processing time equals T1l + Tn + T23 + T34 "=" 1-1/2 revolutions (3/4 + 3 X 1/4 rotations), which equals approximately 30 ms.

The results of ordering A and ordering B are the same, but there is a difference in speed of a factor of 2. Speed = 20 millisecond per rotation Rotation direction

Ordering C

If we knew that "1" equalled record 3 (i.e., current read position is record 3), then the ordering 4, 1, 2, 3 (T34+T4+ T+ T23) would be even better exactly 1 revolution, which equals 20 ms.

In order to accomplish ordering C, it is necessary for the device handler to know the current position for the rotating drum. This hardware facility is called rotational position sensing.

Under heavy I/O loads, rotational ordering can increase throughput there are still some policy decisions that must be made in selecting a rotational ordering because some I/O requests are mutually exclusive. Drum requests, require a two-part address: track number and record number. If the requests are,

track 1, record 1
track 1, record 3
track 2, record 2
track 2, record 3
track 3, record I

then, requests 1 and 5 and requests 2 and 4 are mutually exclusive since they reference the same record number. Thus, either request 1 can be serviced on the first rotation and request 5 on the second rotation, or vice versa. Sorting the requests on the basis of record number is often called slotting since there is one I/O request "slot" per record number per revolution. All I/O requests for the same record number position must "fight" for the same slot. This congestion can be decreased if the hardware allows reading and writing from more than one track at a time, but this capability usually requires the use of an additional channel and control unit as well as more electronics in the device.

Some devices, such as the IBM 2305 Fixed Head Disk, can perform some of the rotational ordering and slotting automatically in the hardware. The control unit for the 2305 can accept up to 16 channel programs at the same time, via a block multiplexor-channel, and it attempts to process these I/O requests in an "optimal" order.

Alternate Addresses

The access time to read an arbitrary record on a drum like device is largely determined by the rotational speed. For a given device this speed is constant.

Recording each record at multiple locations on the device can substantially reduce the effective access time. Thus, there are several alternate addresses for reading the same data record. This technique has also been called ***folding.***

Let us consider a device that has eight records per track and a rotation speed of 20 ms. If record A is stored on track 1, record 1, on the average it would take half a revolution-to ms-to access record A. If a copy of record A is stored at both track 1, record 1 and track 1, record 5,

then by always accessing the "closest" copy, using rotational position sensing, the effective access time can be reduced to 5 ms. Similarly, .the effective access time can be reduced to 2.5 ms or even 1.25 ms by storing even more copies of the same data record.

Of course, by storing multiple copies of the same data record the effective capacity, in terms of unique data records, is reduced by the number of copies. If there are n copies per record, the device has been "folded" n times.

Seek Ordering:

Moving-head storage devices have the problem of seek position in addition to rotational position. I/O requests require a three-part address: cylinder number, track number, and record number. If the requests
cylinder 1, track 2, record 1
cylinder 40, track 3, record 3
cylinder 5, track 6, record 5
cylinder 1, track 5, record 7 were processed in the order shown, a considerable amount of time would be spent to move the seek arm back and forth. A more efficient ordering would be:
cylinder 1, track 2, record 1
cylinder 1, track 5, record 7
cylinder 5, track 6, record 5
cylinder 40, track 3, record 3

These requests are in Minimum seek time order, that is, the distances between seeks have been minimized.

There are many other possible seek orderings When rotational position is important, it is sometimes better to seek a further distance to get a record that will be at a closer rotational position. For an extreme example, if seek times were very small in comparison to the rotation time, the first ordering shown might be best since it accesses records in the order 1, 3, 5, 7. Determining this minimum service time order can be quite complex. It is further complicated by the fact that seek times, though monotonic, are not usually linear as a function of seek distance, i.e., the time to move the seek arm 20 cylinders is usually less than double the time to move it 10 cylinders.

In both the minimum seek time and minimum service time cases, the ordering was determined by the current I/0 requests for that device. New I/O requests are likely to occur while the original ones are being satisfied. In retrospect, the ordering chosen on the basis of the original I/O requests may be poor when the new I/O requests are considered. Unless the system has a "crystal ball" feature, it is impossible to know what future requests will be. Several schemes have been used to minimize the negative effects of future I/O requests.

Usually the ordering is revaluated after each individual request has been processed so as to allow consideration of any new requests. **Unidirectional or linear sweep service time**

ordering is a technique that moves the arm monotonically in one direction. When the last cylinder is reached or there are no further requests in that direction, the arm is reset back to the beginning and another sweep is started. This technique attempts to decrease the amount of back and forward arm thrashing caused by the continued arrival of new I/O requests.

Device Management Software Overheads

Each of the device management algorithms discussed in the preceding sections pro-vides additional device efficiency but at the cost of increased processor time (unless the algorithms are incorporated in the hardware, in which case there is increased hardware expense). The value of these algorithms depends upon the speed of the processor and the amount of I/O requests. If there are few I/O requests, multiprogramming of the processor alone may be sufficient to attain high throughput. If the processor is quite slow, the processor overhead may swamp the I/O advantages.

For example, a System/360 Model 30, with an average instruction time of 30 ms would take 30 ms to execute a 1000 instruction reordering algorithm. If the I/O requests take less than 30 ms to service without reordering, there is no savings attained. However, reordering techniques do become very important on large-scale computers.

Virtual storage memory management increases the load on device management in two ways. First, the page swapping often introduces a large amount of I/O requests. Second; the device handler must handle translation from virtual memory addresses to physical memory addresses in the construction of CCWs, since most channels require physical memory addresses rather than virtual memory addresses.

Finally, device management must ensure the protection of each job's areas in main memory and secondary storage. In the extreme, this would require very care-ful analysis of all I/O requests and addresses referenced. This can be done because all I/0 instructions are privileged and, thus, can be initiated only by the operating system. On the 360 it is possible to use the memory lock
hardware to safeguard memory areas. Similar hardware features are often provided to protect secondary storage areas.

In total, device man-agement consumes a substantial amount of processor capacity. This overhead is usually worthwhile since device management can sig-nificantly increase system throughput by decreasing system I/O wait time.

Virtual Devices

Motivation:

Devices such as card readers, punches, and printers present two serious problems that hamper effective device utilization.

1) Each of these devices performs best when there is a steady, continuous stream of requests at a specified rate that matches. its physical characteristics (e.g., 1,000 cards

per minute read rate, 1,000 lines per minute print rate, etc.). If a job attempts to generate requests faster than the device's performance rate, the job must wait a significant amount of time. On the other hand, if a job generates requests at a much lower rate, the device is idle much of the time and is substantially underutilized.

2) These devices must be dedicated to a single job at a time. Thus, since most jobs perform some input and output, we would require as many card readers and printers as we have jobs being multiprogrammed. This would normally be uneconomical because most jobs use only a fraction of the device's capacity.

To summarize, these devices should have a steady request rate, whereas actual programs generate highly irregular requests. A small input/output-intensive program (e.g., a check-printing program) might generate thousands of print requests in a few seconds of CPU time. On the other hand, a compute-intensive program (e.g., de-termine the first 1,000 prime numbers) may print only a single line of output for every hour of CPU time. Buffering and multiprogramming help reduce these problems, are not capable of solving them completely. Furthermore, many other devices, such as plotters, graphic displays, and type-writer terminals, have similar problems.

Historical Solutions

These problems would disappear, or at least substantially decrease, if it were possible to use Direct Access Storage Devices for all input and output. A single DASD can be efficiently shared and simultaneously used for reading and/or writing data by many jobs, as shown in Figure 4.7. Furthermore, DASDs provide very high performance rates, especially if the data are locked, thereby decreasing the amount of wait time for jobs that require substantial amounts of input/output.

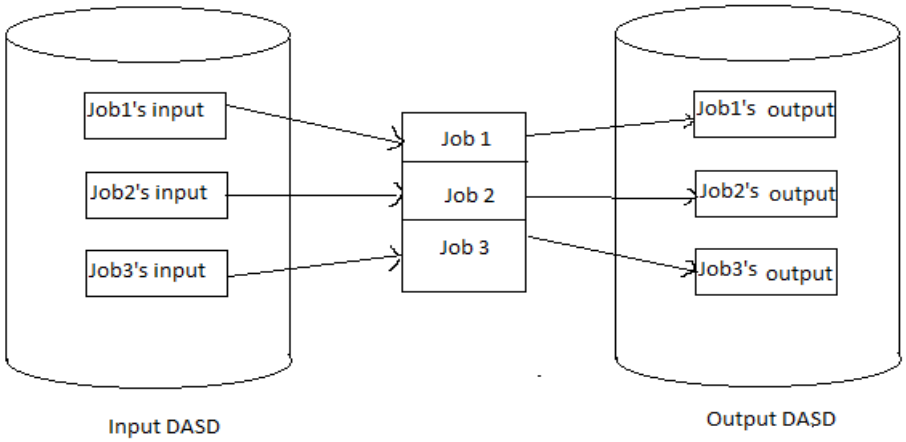


Figure 4.7 Use of DASDs input and output

Although theoretically appealing, the use of DASDs for all input and output appears impractical. How do we get our input data onto the DASD in the first place? What do we do with the output data recorded on the output DASD? Consider for a moment walking into a drugstore with a disk pack and trying to convince the clerk that recorded on a specific part of the disk is our payroll check and you wish to cash it!

Unit IV Completed

Unit: V

File Management: The File Manager -Interacting with the File Manager - File Organization - Physical Storage Allocation -Access Methods-Levels in a File Management System - Access Control Verification Module

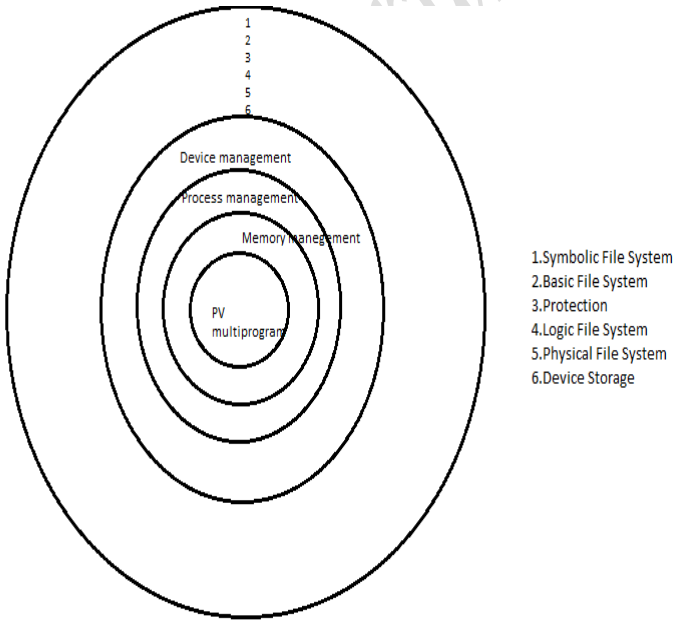
Introduction:

The Modules of information management are collectively referred to File System.

Information management is concerned with the storage and retrieval of information entrusted to the system or in a library.

Functions of Information management:

1. Keeping track of all information in the system through various tables.
2. Deciding policy for determining where and how information is stored and who gets access to the information.
3. Allocation the information resources.
4. Deallocating of the resource.



File

A file is a named collection of related information that is recorded on secondary storage such as magnetic disks, magnetic tapes and optical disks. In general, a file is a sequence of bits, bytes, lines or records whose meaning is defined by the files creator and user.

File Structure

A File Structure should be according to a required format that the operating system can understand.

- A file has a certain defined structure according to its type.
- A text file is a sequence of characters organized into lines.
- A source file is a sequence of procedures and functions.
- An object file is a sequence of bytes organized into blocks that are understandable by the machine.
- When operating system defines different file structures, it also contains the code to support these file structure. UNIX, MS-DOS support minimum number of file structure.

File Types

File type refers to the ability of the operating system to distinguish different types of file such as text files source files and binary files etc. Many operating systems support many types of files. Operating system like MS-DOS and UNIX have the following types of files −

Ordinary files

- These are the files that contain user information.
- These may have text, databases or executable program.
- The user can apply various operations on such files like add, modify, delete or even remove the entire file.

Directory files

- These files contain list of file names and other information related to these files.

Special files

- These files are also known as device files.
- These files represent physical device like disks, terminals, printers, networks, tape drive etc.

These files are of two types −

- **Character special files** − data is handled character by character as in case of terminals or printers.
- **Block special files** − data is handled in blocks as in the case of disks and tapes.

A Simple File System:

We consider the steps needed to process the following PL/I-like statement:

READ FILE (RAJ) RECORD (4) INTO LOCATION (1200)

This request to read the fourth record of the file named *RAJ* into a memory location 1200. Such a request invokes elements of the file system.

Let us assume the file RAJ is stored on disk, shown in figure 5.1.

- The file RAJ is the shaded portion and consists of seven logical records.
- Each logical record is 500 bytes long.
- The disk consists of 16 physical blocks of 1000 bytes each.

The physical units of the storage device are called **Blocks.** The logical units of the storage device are called **Records.**

We can store two logical records of the file RAJ into each physical block. Instead of using the physical two-component (track number, record number) address, we will assign each physical block a unique number, as indicate in Figure 5.1.

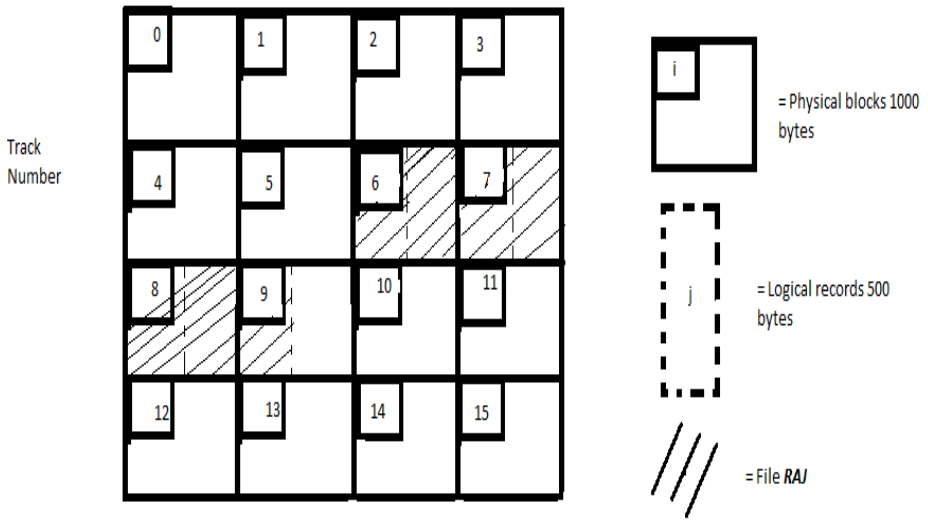


Figure 5.1 Physical file Storage

File Directory Database:

Information on each existing file's status, (name and location) must be maintained in a table. This table is called **File Directory**.

File Directories:

Collection of files is a file directory. The directory contains information about the files, including attributes, location and ownership. Much of this information especially that is concerned with storage is managed by the operating system. The directory is itself a file, accessible by various file management routines

Information contained in a device directory are:

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed
- Date last updated
- Owner id
- Protection information

Operations performed on directory are:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

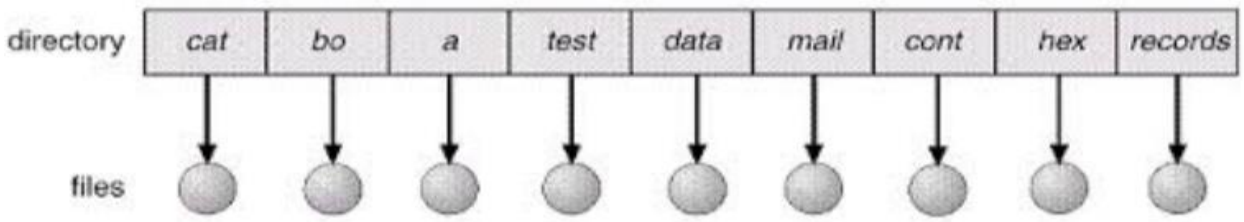
Advantages of maintaining directories are:

- **Efficiency:** A file can be located more quickly.
- **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
- **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.

Single-Level Directory

In this a single directory is maintained for all the users.

- **Naming problem:** Users cannot have same name for two files.
- **Grouping problem:** Users cannot group files according to their need**.**



Physical Files and Logical Files

Physical files contain the actual data that is stored on the system, and a description of how data is to be presented to or received from a program. They contain only one record format, and one or more members. Records in database files can be externally or program-described.

A physical file can have a keyed sequence access path. This means that data is presented to a program in a sequence based on one or more key fields in the file.

Logical files do not contain data. They contain a description of records found in one or more physical files. A logical file is a view or representation of one or more physical files. Logical files that contain more than one format are referred to as **multi-format** logical files.

If your program processes a logical file which contains more than one record format, you can use a read by record format to set the format you wish to use.

Logical File System Overview

The *logical file system* is the level of the file system at which users can request file operations by system call. This level of the file system provides the kernel with a consistent view of what might be multiple physical file systems and multiple file system implementations. As far as the logical file system is concerned, file system types, whether local, remote, or strictly logical, and regardless of implementation, are indistinguishable.

A consistent view of file system implementations is made possible by the virtual file system abstraction. This abstraction specifies the set of file system operations that an implementation must include in order to carry out logical file system requests. Physical file systems can differ in how they implement these predefined operations, but they must present a uniform interface to the logical file system. A list of file system operators can be found at Requirements for a File System Implementation.

Each set of predefined operations implemented constitutes a virtual file system. As such, a single physical file system can appear to the logical file system as one or more separate virtual file systems.

Virtual file system operations are available at the logical file system level through the *virtual file system switch*. This array contains one entry for each virtual file system, with each entry holding entry point addresses for separate operations. Each file system type has a set of entries in the virtual file system switch.

The logical file system and the virtual file system switch support other operating system file-system access semantics. This does not mean that only other operating system file systems can be supported. It does mean, however, that a file system implementation must be designed to fit into the logical file system model. Operations or information requested from a file system implementation need be performed only to the extent possible.

Logical file system can also refer to the tree of known path names in force while the system is running. A virtual file system that is mounted onto the logical file system tree itself becomes part of that tree. In fact, a single virtual file system can be mounted onto the logical file system tree at multiple points, so that nodes in the virtual sub tree have multiple names. Multiple mount points allow maximum flexibility when constructing the logical file system view.

FILE DIRECTORIES:

Collection of files is a file directory. The directory contains information about the files, including attributes, location and ownership. Much of this information, especially that is concerned with storage, is managed by the operating system. The directory is itself a file, accessible by various file management routines.

Information contained in a device directory are:

File type	Usual extension	Function
Executable	exe,com,bin	Read to run machine language program
Object	obj,o	Compiled,machine language not linked
Source code	C,java,pas,asm,a	Source code in various languages
Batch	bat,sh	Commands to the command interpreter
Text	txt,doc	Textual data,documents
Word processor	Wp,tex,rrf,doc	Various word processor formats
Archive	arc,zip,tar	Related files grouped into one file compressed
		for storage
Multimedia	mpeg,mov,rm	File containing audio or a/v information

- Name
- Type
- Address
- Current length
- Maximum length
- Date last accessed
- Date last updated
- Owner id
- Protection information

Operation performed on directory are:

- Search for a file
- Create a file
- Delete a file
- List a directory
- Rename a file
- Traverse the file system

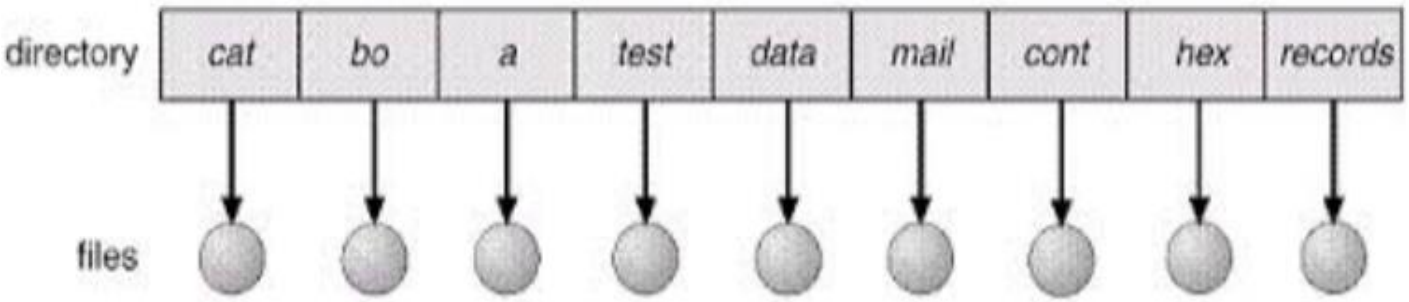
Advantages of maintaining directories are:

- **Efficiency:** A file can be located more quickly.
- **Naming:** It becomes convenient for users as two users can have same name for different files or may have different name for same file.
- **Grouping:** Logical grouping of files can be done by properties e.g. all java programs, all games etc.
-

SINGLE-LEVEL DIRECTORY

In this a single directory is maintained for all the users.

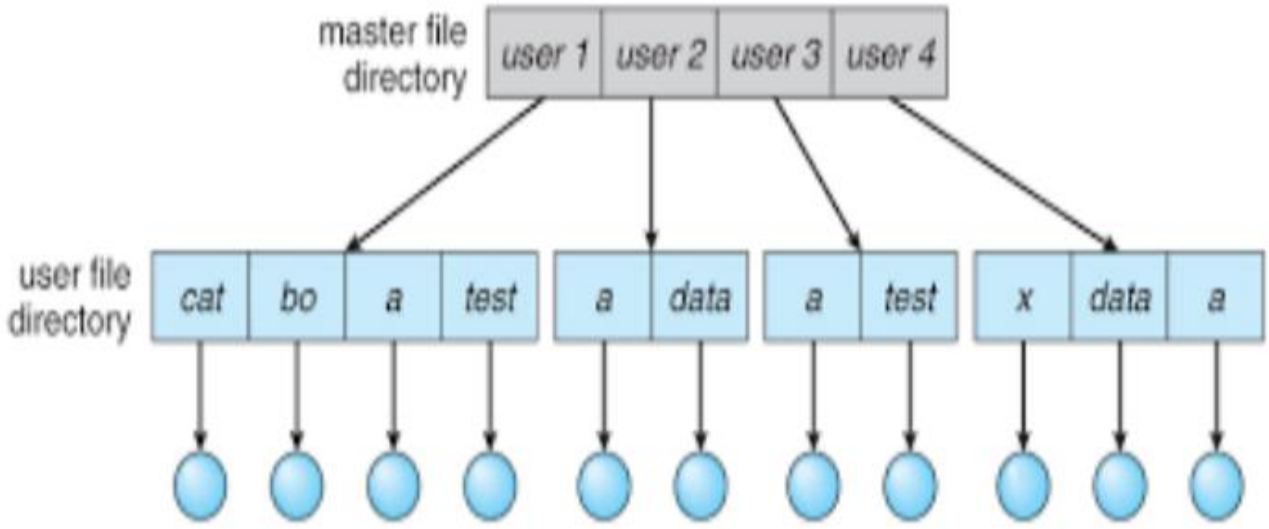
- **Naming problem:** Users cannot have same name for two files.
- **Grouping problem:** Users cannot group files according to their need.



TWO-LEVEL DIRECTORY

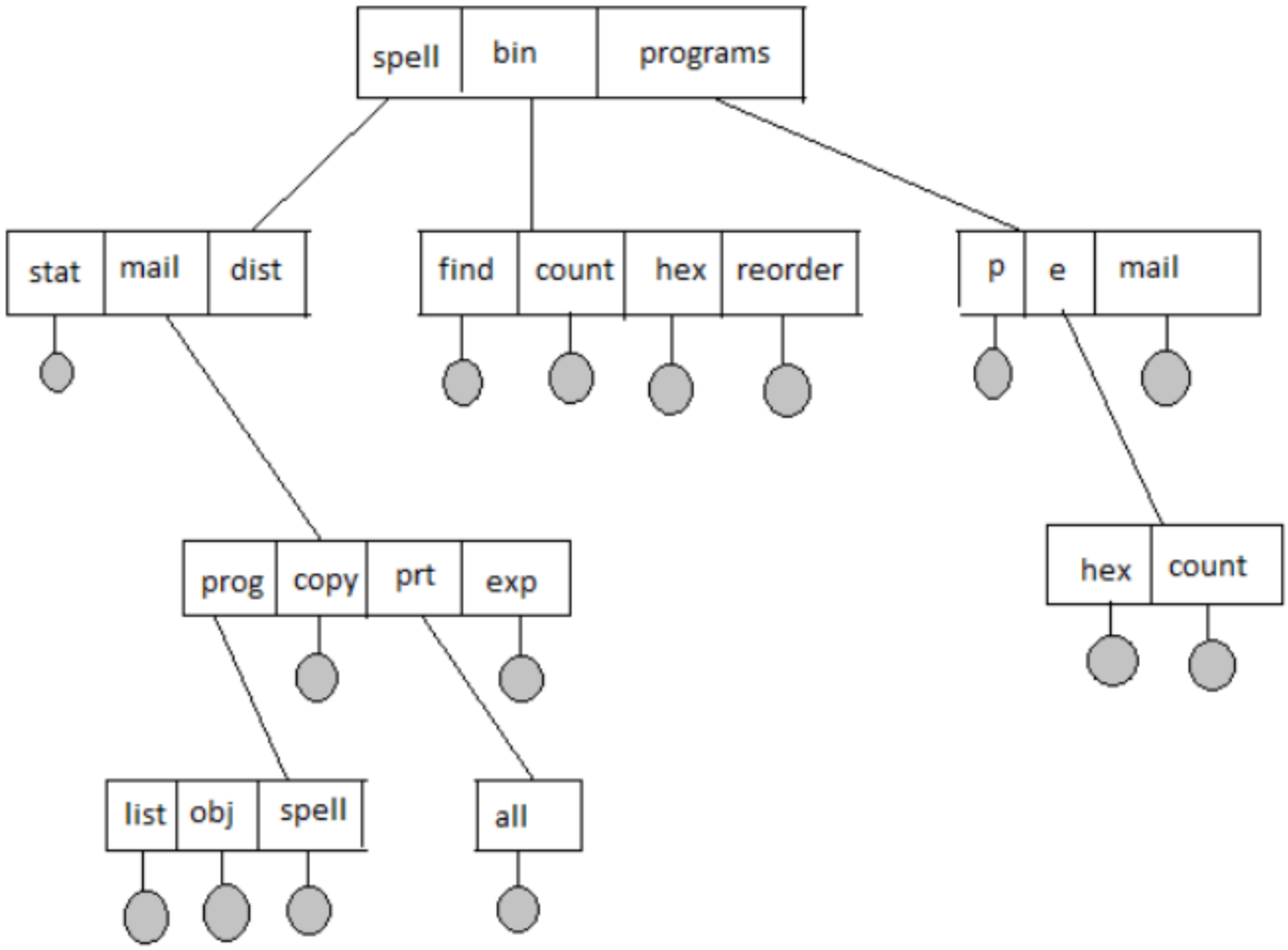
In this separate directories for each user is maintained.

- Path name:Due to two levels there is a path name for every file to locate that file.
- Now,we can have same file name for different user.
- Searching is efficient in this method.



TREE-STRUCTURED DIRECTORY:

Directory is maintained in the form of a tree. Searching is efficient and also there is grouping capability. We have absolute or relative path name for a file.

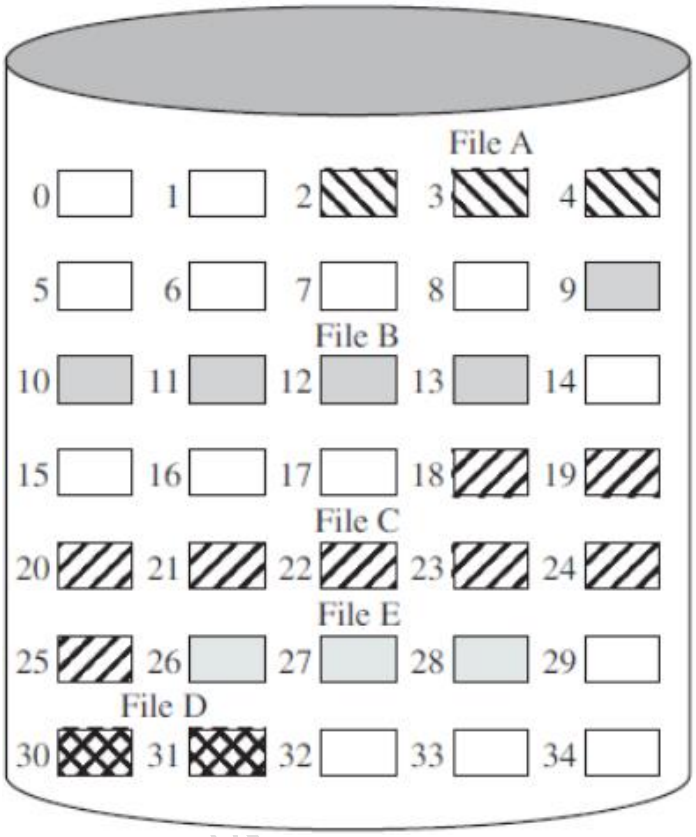


FILE ALLOCATION METHODS

1. Continuous Allocation:

A single continuous set of blocks is allocated to a file at the time of file creation. Thus, this is a pre-allocation strategy, using variable size portions. The file allocation table needs just a single entry for each file, showing the starting block and the length of the file. This method is best from the point of view of the individual sequential file. Multiple blocks can be read in at a time to improve I/O performance for sequential

processing. It is also easy to retrieve a single block. For example, if a file starts at block b, and the ith block of the file is wanted, its location on secondary storage is simply b+i-1.



File allocation table

File name	Start block	Length
File A	2	3
File B	9	5
File C	18	8
File D	30	2
File E	26	3

Disadvantage

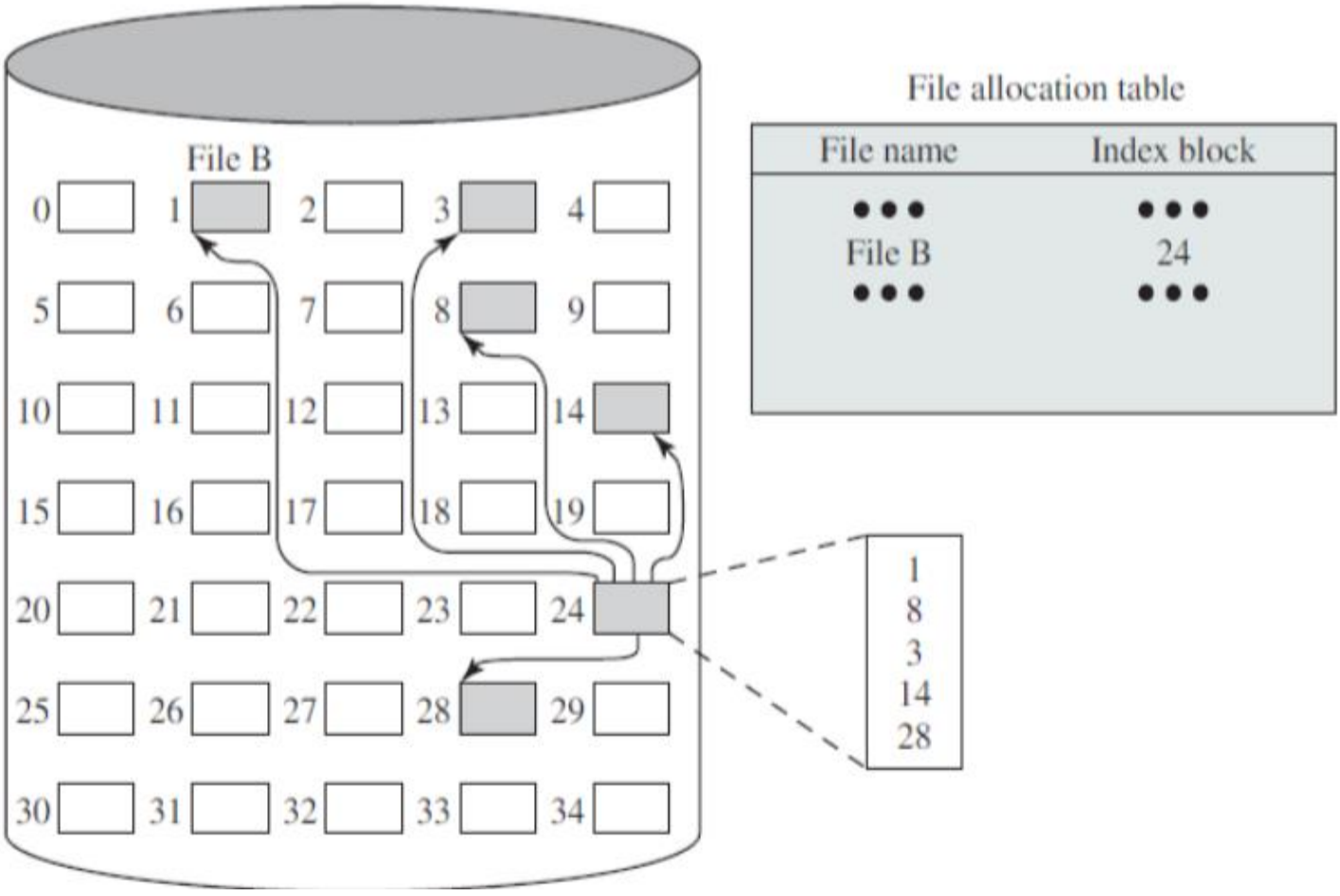
- External fragmentation will occur, making it difficult to find contiguous blocks of space of sufficient length. Compaction algorithm will be necessary to free up additional space on disk.
- Also, with pre-allocation, it is necessary to declare the size of the file at the time of creation.

2. Linked Allocation(Non-contiguous allocation) : Allocation is on an individual block basis. Each block contains a pointer to the next block in the chain. Again the file table needs just a single entry for each file,

showing the starting block and the length of the file. Although pre-allocation is possible, it is more common simply to allocate blocks as needed. Any free block can be added to the chain. The blocks need not be continuous. Increase in file size is always possible if free disk block is available. There is no external fragmentation because only one block at a time is needed but there can be internal fragmentation but it exists only in the last disk block of file.

Disadvantage:

- Internal fragmentation exists in last disk block of file.
- There is an overhead of maintaining the pointer in every disk block.
- If the pointer of any disk block is lost, the file will be truncated.
- It supports only the sequencial access of files.



3. Indexed Allocation:

It addresses many of the problems of contiguous and chained allocation. In this case, the file allocation table contains a separate one-level index for each file: The index has one entry for each block allocated to the file. Allocation may be on the basis of fixed-size blocks or variable-sized blocks. Allocation by blocks eliminates external fragmentation, whereas allocation by variable-size blocks improves locality. This allocation technique supports both sequential and direct access to the file and thus is the most popular form of file allocation.

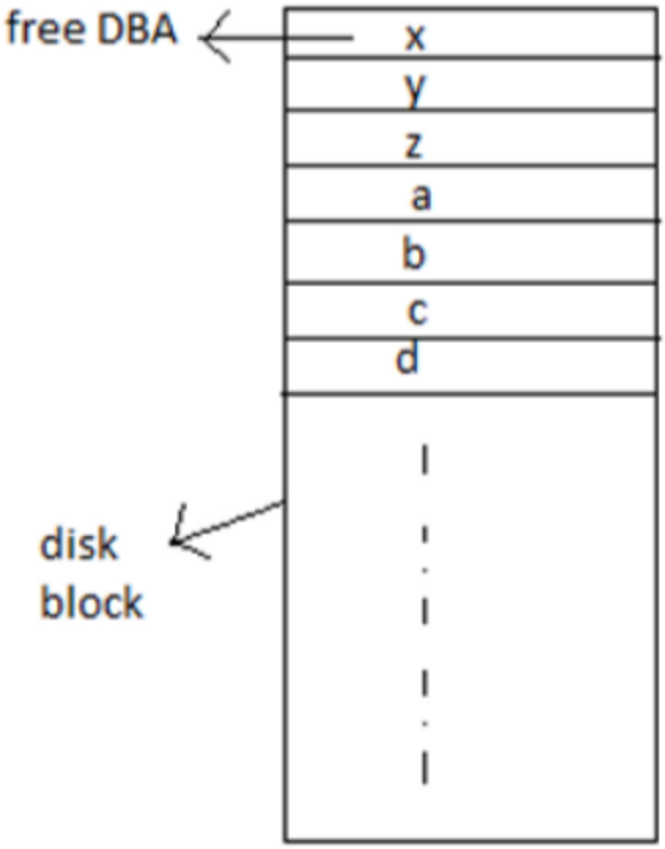
Disk Free Space Management

Just as the space that is allocated to files must be managed ,so the space that is not currently allocated to any file must be managed. To perform any of the file allocation techniques,it is necessary to know what blocks on the disk are available. Thus we need a disk allocation table in addition to a file allocation table.The following are the approaches used for free space management.

1. **Bit Tables** : This method uses a vector containing one bit for each block on the disk. Each entry for a 0 corresponds to a free block and each 1 corresponds to a block in use.
   For example: 00011010111100110001

In this vector every bit correspond to a particular vector and 0 implies that, that particular block is free and 1 implies that the block is already occupied. A bit table has the advantage that it is relatively easy to find one or a contiguous group of free blocks. Thus, a bit table works well with any of the file allocation methods. Another advantage is that it is as small as possible.

2. **Free Block List** : In this method, each block is assigned a number sequentially and the list of the numbers of all free blocks is maintained in a reserved block of the disk.



File Access Methods

When a file is used, information is read and accessed into computer memory and there are several ways to access this information of the file. Some systems provide only one access method for files. Other systems, such as those of IBM, support many access methods, and choosing the right one for a particular application is a major design problem.

There are three ways to access a file into a computer system: Sequential-Access, Direct Access, Index sequential Method.

1. **Sequential Access –**

   It is the simplest access method. Information in the file is processed in order, one record after the other. This mode of access

either read or write in one by one

- Data is accessed one record right after another record in an order.
- When we use read command, it move ahead pointer by one
- When we use write command, it will allocate memory and move the pointer to the end of the file
- Such a method is reasonable for tape.

2. **Direct Access –**

   Another method is *direct access method* also known as *relative access method*. A filed-length logical record that allows the program to read and write record rapidly. in no particular order.

   The direct access is based on the disk model of a file since disk allows random access to any file block.

   Thus, we may read block 14 then block 59 and then we can write block 17. There is no restriction on the order of reading and writing for a direct access file.

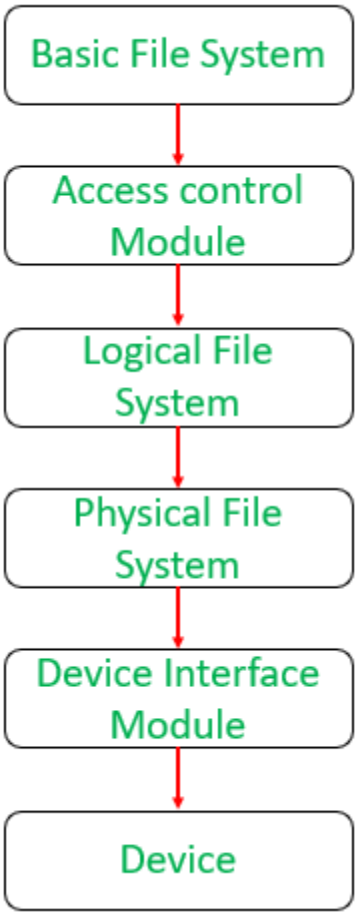
A block number provided by the user to the operating system is normally a *relative block number*, the first relative block of the file is 0 and then 1 and so on.

3. **Index sequential method –**

   It is the other method of accessing a file which is built on the top of the direct access method. These methods construct an index for the file. The index, like an index in the back of a book, contains the pointer to the various blocks. To find a record in the file, we first search the index and then by the help of pointer we access the file directly.

Levels in a File Management System

The management of files and the management of device are interlinked with each other. Given below is an hierarchy used to perform the required functions of an I/O system efficiently.



The highest level module called *Basic File System* passes the information given to it to *Logical File System*, which in turn, notifies the *Physical File System*, that works with Device Manager.