ARBAMINCH UNIVERSITY

ARBAMINCH INSTITUTE OF TECHNOLOGY

FACULTY OF COMPUTING AND SOFTWARE ENGINEERING

OPERATING SYTEM

Compiled by: Dawit Mekonnen Tekleyohannes
Dawit.mekonnen@amu.edu.et

Contents

Chapter 1: Introduction to Operating System

An Operating System (OS) is a collection of software that manages hardware resources and provides various service to the users. OS lies between hardware and user program. It acts as an intermediary between the user and the hardware.

Modes of operation of computer

Kernel Mode: has complete access to all the hardware. It can execute any instruction that a machine is capable of executing. It has high privileged (rights). The **User Mode** it can execute only subset (few) of the machine instructions. This mode has less privileged (rights).

Roles of modern operating system

1. **The OS as an Extended/Virtual Machine: OS provides a set of basic commands** or instructions to perform various operations such as read, write, modify, save or close. Operating system **hides the complexity of hardware** and **present a beautiful interface** to the users.

2. **The OS as a Resource Manager:** It is the job of OS to allocate these resources to the various applications so that Deadlock are detected, resolved and avoided. Resource manager – sharing resources in two different ways:
   - ✓ In time sharing/multiplexing (i.e CPU)
   - ✓ In space sharing/multiplexing. (i.e Memory)

OS Services / Task of OS

- ✓ Program development
- ✓ Program execution
- ✓ Access to I/O devices
- ✓ Memory management
- ✓ Controlled access to file
- ✓ Communication
- ✓ Error detection and response
- ✓ Accounting
- ✓ Protection & Security

Types of OS

- ✓ Mainframe operating systems
- ✓ Server operating systems
- ✓ Multiprocessor operating systems
- ✓ Personal computer operating systems
- ✓ Handhelds computer operating systems
- ✓ Embedded operating systems
- ✓ Sensor node operating systems
- ✓ Real time operating systems
- ✓ Smart card operating systems

Chapter 2: Process Management

Process is a program under execution. Process is an abstraction of a running program. Process is an instance of an executing program, including the current values of the program counter, registers & variables. Each process has its own virtual CPU. The real CPU **switches back and forth** from process to process. This **rapid switching back and forth** is called **multiprogramming**. The **number of processes loaded simultaneously in memory** is called **degree of multiprogramming.**

Process Creation

Processes can be created in either of four ways

- ✓ System initialization: At the time of system (OS) booting various processes are created. Foreground and background processes are created
    - Background process – that do not interact with user e.g. process to accept mail
    - Foreground Process – that interact with user
- ✓ Execution of a process creation system call (fork) by running process
- ✓ A user request to create a new process
- ✓ Initialization of batch process

Process Termination

- ✓ Normal exit (voluntary): Terminated because process has done its work.
- ✓ Error exit (voluntary): The process **discovers a fatal error** e.g. user types the command **ccfoo.c** to compile the program foo.c and no such file exists, the compiler simply exit.
- ✓ Fatal error (involuntary): An error caused by a process often **due to a program bug** e.g. executing an illegal instruction, referencing nonexistent memory or divided by zero.
- ✓ Killed by another process (involuntary): process **executes a system call** telling the OS **to kill some other process** using **kill system call**.

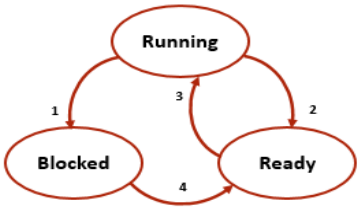
Process State

Processes are always **either executing (running) or waiting to execute (ready) or waiting for an event (blocked)** to occur.

1. Running – Process is actually using the CPU
2. Ready – Process is runnable, temporarily stopped to let another process to run
3. Blocked – process is unable to run until some external event happens

When and how these transitions occur (process moves from one state to another)?

1. Process blocks for input or waits for an event (i.e. printer is not available)
2. Scheduler picks another process: End of time-slice or pre-emption.
3. Scheduler picks this process
4. Input becomes available, event arrives (i.e. printer become available)



Process Control Block (PCB)

A Process Control Block (PCB) is a data structure maintained by the operating system for every process. PCB is used for storing the collection of information about the processes. The PCB is identified by an integer process ID (PID). A PCB keeps all the information needed to keep track of a process. The PCB is maintained for a process **throughout its lifetime** and is **deleted once** the **process terminates**. The **architecture** of a PCB is completely **dependent on operating system** and may contain different information in different operating systems. PCB **lies in kernel** memory space.

Process Control Block (PCB) contains

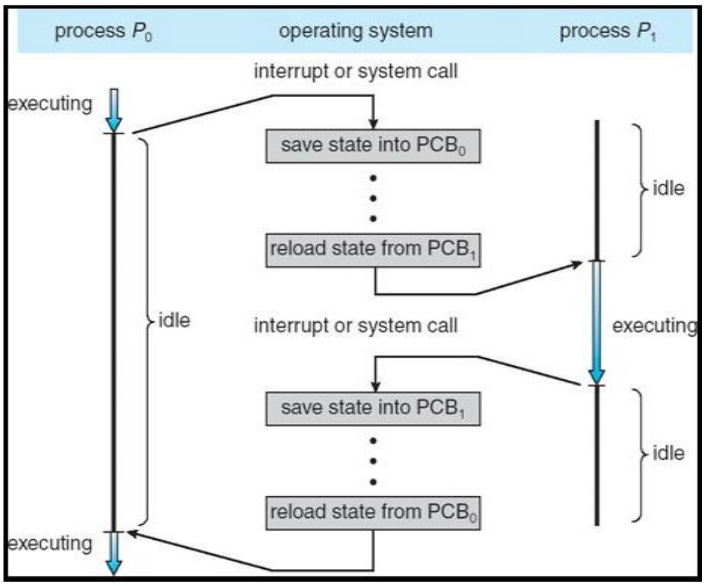
- ✓ Process ID - Unique identification for each of the process in the operating system.
- ✓ Process State - The current state of the process i.e., whether it is ready, running, waiting.
- ✓ Pointer - A pointer to parent process.
- ✓ Priority - Priority of a process.
- ✓ Program Counter - Program Counter is a pointer to the address of the next instruction to be executed for this process.
- ✓ CPU registers - Various CPU registers where process need to be stored for execution for running state.
- ✓ IO status information - This includes a list of I/O devices allocated to the process.

- ✓ Accounting information - This includes the amount of CPU used for process execution, time limits etc.

Context switching

Context switch means stopping one process and restarting another process. When an event occur, the OS saves the state of an active process and restore the state of new process. Context switching is purely overhead because system does not perform any useful work while context switch. Sequence of action:

1. OS takes control (through interrupt)
2. Saves context of running process in the process PCB
3. Reload context of new process from the new process PCB
4. Return control to new process



Thread

Thread is **light weight process** created by a process. Processes are used to execute large, 'heavyweight' jobs such as working in word, while threads are used to carry out smaller or 'lightweight' jobs such as auto saving a word document. **Thread is light weight process created by a process. Thread is a single sequence stream within a process. Thread has its own**

- ✓ Program counter that keeps track of which instruction to execute next.
- ✓ System registers which hold its current working variables.
- ✓ Stack which contains the execution history.

Per process items	**Per thread items**
Address space	Program counter
Global variables	Registers
Open files	Stack
Child processes	State
Pending alarms	
Signals and signal handlers	
Accounting information	

Inter process Communication

Processes in a system can be independent or cooperating. **Independent process cannot affect** or be **affected** by the execution of another process. **Cooperating process can affect** or be **affected** by the execution of another process. Cooperating processes need **inter process communication** mechanisms.

Reasons of process cooperation

Information sharing: Sharing of information between multiple processes can be accomplished using cooperating processes. This may include access to the same files. A mechanism is required so that the processes can access the files in parallel to each other.

Computation speed-up: Subtasks of a single task can be performed parallel using cooperating processes. This increases the computation speedup as the task can be executed faster.

Modularity: Modularity involves dividing complicated tasks into smaller subtasks. These subtasks can completed by different cooperating processes. This leads to faster and more efficient completion of the required tasks.

Convenience: There are many tasks that a user needs to do such as compiling, printing, editing etc. It is convenient if these tasks can be managed by cooperating processes.

Inter Process Communication: It is a communication between two or more processes. There are two models for IPC. Message Passing (Process A send the message to Kernel and then Kernel send that message to Process B). Shared Memory (Process A put the message into Shared Memory and then Process B read that message from Shared Memory).

Race Condition

A race condition is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time. But, because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly. Race Condition is Situations like this where processes access the same data concurrently and the outcome of execution depends on the particular order in which the access takes place is called a race condition. It is Situation where two or more processes are reading or writing some shared data and the final result depends on who runs precisely and when. Reasons for Race Condition can be Exact instruction execution order cannot be predicted and Resource (file, memory, data etc…) sharing.

Critical Section

The part of program where the shared resource is accessed is called critical section or critical region.

Mutual Exclusion

Way of making sure that if one process is using a shared variable or file; the other process will be excluded (stopped) from doing the same thing.

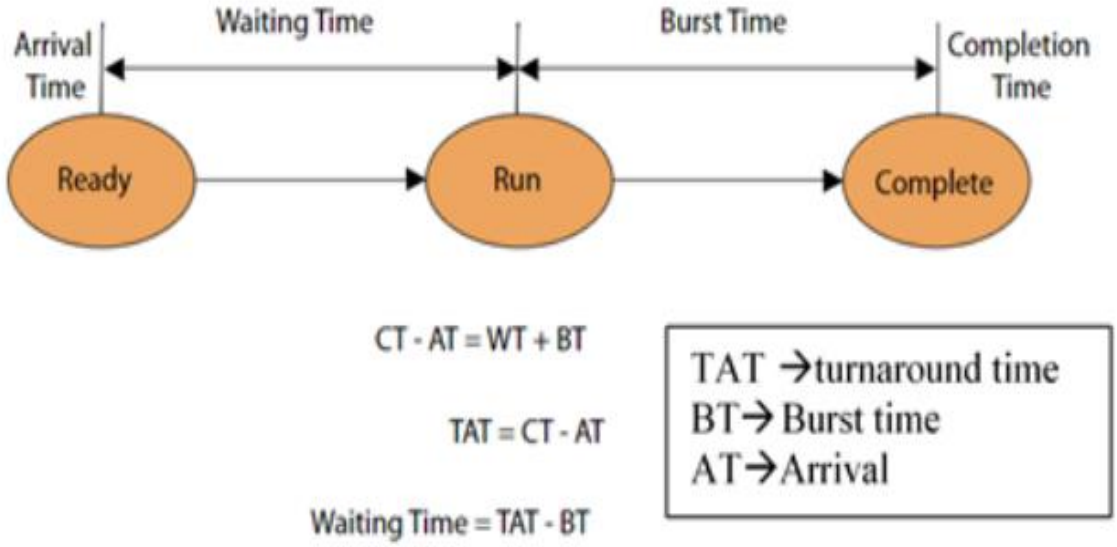
Process Scheduling

Process scheduling is the activity of the process manager that **handles suspension of running process** from CPU and **selection of another process** on the basis of a particular ***strategy***. The **part of operating system** that **makes the choice** is called **scheduler**. The **algorithm used** by this scheduler is called **scheduling algorithm.** Process scheduling is an essential part of a multiprogramming operating systems.

Objectives (goals) of scheduling

- ✓ Fairness: giving each process a fair share of the CPU.
- ✓ Balance: keeping all the parts of the system busy (Maximize).
- ✓ Throughput: no of processes that are completed per time unit (Maximize).
- ✓ Turnaround time: time to execute a process from submission to completion (Minimize).
    - **Turnaround time** = Process finish time – Process arrival time

- ✓ CPU utilization: It is percent of time that the CPU is busy in executing a process.
  - keep CPU as busy as possible (Maximized).
- ✓ Response time: time between issuing a command and getting the result (Minimized).
- ✓ Waiting time: amount of time a process has been waiting in the ready queue (Minimize).
  - Waiting time = Turnaround time – Actual execution time



Scheduling algorithms Modes

- ✓ Preemptive algorithm
  - If a process is still running at the end of its time interval, it is suspended and another process is picked up
- ✓ Non-preemptive
  - Picks a process to run and then lets it run till it blocks or it voluntarily releases the CPU
  - once the CPU has been allocated to a process, the process keeps CPU until it releases the CPU either by terminating or by switching to the waiting state.

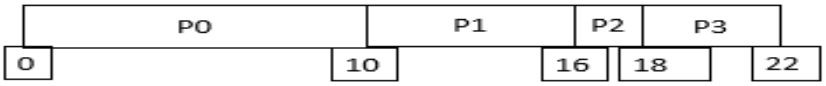
First Come First Served (FCFS)

The process that request first is served first. It means that processes are served in the exact order of their arrival. Decision Mode is Non preemptive: Once a process is selected, it runs until it is blocked for an I/O or some other event or it is terminated. This strategy can be easily implemented by using FIFO (First In First Out) queue. When CPU becomes free, a process from the first position in a queue is selected to run. Disadvantages of this algorithm is that **Convoy effect is possible**. All small I/O bound processes wait for one big CPU bound process to acquire CPU.

- Example

Process	Arrival Time (T0)	Burst Time (ΔT)	Finish Time (T1)	Turnaround Time (TAT = T1-T0)	Waiting Time (WT = TAT-ΔT)
P0	0	10	10	10	0
P1	1	6	16	15	9
P2	3	2	18	15	13
P3	5	4	22	17	13

- **Gantt Chart**



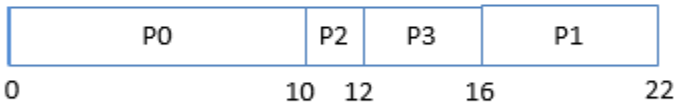
- Average Turnaround Time: (10+15+15+17)/4 = 14.25 ms.
- Average Waiting Time: (0+9+13+13)/4 = 8.75 ms.

Shortest Job First (SJF)

The process, that requires shortest time to complete execution, is served first.. the Decision Mode is Non preemptive: Once a process is selected, it runs until either it is blocked for an I/O or some other event or it is terminated. This strategy can be easily implemented by using FIFO (First In First Out) queue. All processes in a queue are sorted in ascending order based on their required CPU bursts. When CPU becomes free, a process from the first position in a queue is selected to run. It is difficult to estimate time required to complete execution. Starvation is possible for long process. Long process may wait forever.

Process	Arrival Time (T0)	Burst Time (ΔT)	Finish Time (T1)	Turnaround Time (TAT = T1-T0)	Waiting Time (WT = TAT-ΔT)
P0	0	10	10	10	0
P1	1	6	22	21	15
P2	3	2	12	9	7
P3	5	4	16	11	7

- Gantt Chart



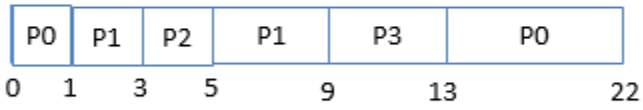
- Average Turnaround Time: (10+21+9+11)/4 = 12.75 ms.
- Average Waiting Time: (0+15+7+7)/4 = 7.25 ms.

Shortest Remaining Time Next (SRTN)

The process, whose remaining run time is shortest, is served first. This is a preemptive version of SJF scheduling. The Decision Mode used is Preemptive: When a new process arrives, its total time is compared to the current process remaining run time. If the new process needs less time to finish than the current process, the current process is suspended and the new job is started. This strategy can also be implemented by using sorted FIFO queue. All processes in a queue are sorted in ascending order on their remaining run time. When CPU becomes free, a process from the first position in a queue is selected to run. Here **Starvation is possible** for **long process**. Long process may wait forever and again **Context switch overhead is there**.

Process	Arrival Time (T0)	Burst Time (ΔT)	Finish Time (T1)	Turnaround Time (TAT = T1-T0)	Waiting Time (WT = TAT-ΔT)
P0	0	10	22	22	12
P1	1	6	9	8	2
P2	3	2	5	2	0
P3	5	4	13	8	4

- Gantt Chart



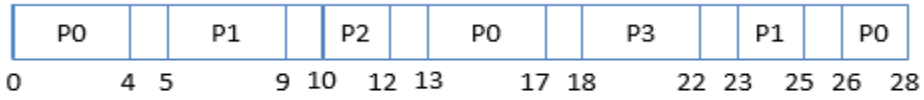
- Average Turnaround Time: (22+8+2+8) / 4 = 10 ms.
- Average Waiting Time: (12+2+0+4)/4 = 4.5 ms.

Round Robin (RR)

Each selected process is assigned a time interval, called time quantum or time slice. Process is allowed to run only for this time interval. Here, two things are possible. First, process is either blocked or terminated before the quantum has elapsed. In this case the CPU switching is done and another process is scheduled to run. Second, process needs CPU burst longer than time quantum. In this case, process is running at the end of the time quantum. Now, it will be preempted and moved to the end of the queue. CPU will be allocated to another process. Here, length of time quantum is critical to determine. The Decision Mode is **Preemptive**: When quantum time is over or process completes its execution (whichever is earlier), it starts new job. **Selection** of new job is **as per FCFS** scheduling algorithm. This strategy can be implemented by using circular FIFO queue. If any process comes, or process releases CPU, or process is preempted. It is moved to the end of the queue. When CPU becomes free, a process from the first position in a queue is selected to run. Here **Context switch overhead** is there. **Determination of time quantum is too critical**. If it is too **short**, it **causes frequent context switches** and **lowers CPU efficiency**. If it is too **long**, it **causes poor response** for **short interactive process**.

Process	Arrival Time (T0)	Burst Time (ΔT)	Finish Time (T1)	Turnaround Time (TAT = T1-T0)	Waiting Time (WT = TAT-ΔT)
P0	0	10	28	28	18
P1	1	6	25	24	18
P2	3	2	12	9	7
P3	5	4	22	17	13

- Gantt Chart (Quantum time is 4 ms & context switch overhead is 1 ms)



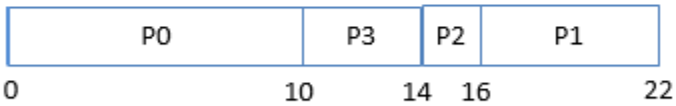
- Average Turnaround Time: (28+24+9+17)/4 = 19.5 ms.
- Average Waiting Time: (18+18+7+13)/4 = 14 ms.

Non Preemptive Priority Scheduling

The process, that has highest priority, is served first. The Decision Mode: Non Preemptive: Once a process is selected, it runs until it blocks for an I/O or some event or it terminates. This strategy can be implemented by using sorted FIFO queue. All processes in a queue are sorted based on their priority with highest priority process at front end. When CPU becomes free, a process from the first position in a queue is selected to run. Here **Starvation is possible for low priority processes**. It can be overcome by using technique called 'Aging'. **Aging**: **gradually increases the priority of processes** that wait in the system for a long time.

Process	Arrival Time (T0)	Burst Time (ΔT)	Finish Time (T1)	Turnaround Time (TAT = T1-T0)	Waiting Time (WT = TAT-ΔT)
P0	**0**	**10**	**10**	**10**	**0**
P1	**1**	**6**	**22**	**21**	**15**
P2	**3**	**2**	**16**	**13**	**11**
P3	**5**	**4**	**14**	**9**	**5**

- Gantt Chart (small values for priority means higher priority of a process)



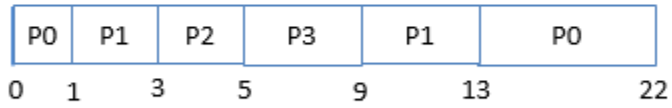
- Average Turnaround Time: (10+21+13+9) / 4 = 13.25 ms
- Average Waiting Time: (0+15+11+5) / 4 = 7.75 ms

Preemptive Priority Scheduling

The process, that has highest priority, is served first. The Decision Mode is Preemptive: When a new process arrives, its priority is compared with current process priority. If the new process has higher priority than the current, the current process is suspended and new job is started. This strategy can be implemented by using sorted FIFO queue. All processes in a queue are sorted based on priority with highest priority process at front end. When CPU becomes free, a process from the first position in a queue is selected to run.Here again **Starvation is possible for low priority processes**. It can be overcome by using technique called 'Aging'. **Aging**: **gradually increases the priority of processes** that wait in the system for a long time. **Context switch overhead is there**.

Process	Arrival Time (T0)	Burst Time (ΔT)	Finish Time (T1)	Turnaround Time (TAT = T1-T0)	Waiting Time (WT = TAT-ΔT)
P0	0	10	22	22	12
P1	1	6	13	12	6
P2	3	2	5	2	0
P3	5	4	9	4	0

- Gantt Chart (small values for priority means higher priority of a process)



- Average Turnaround Time: (22+12+2+4) / 4 = 10 ms
- Average Waiting Time: (12+6+0+0) / 4 = 4.5 ms

Deadlocks

Deadlock is a set of processes is deadlocked **if each process in the set is waiting for an event** that **only another process in the set can cause**. Deadlocks are a **set of blocked processes** each **holding a resource and waiting to acquire a resource held by another process.**

Conditions that lead to deadlock

1. Mutual exclusion
   - Each resource is either currently assigned to exactly one process or is available.
2. Hold and wait
   - Process currently holding resources granted earlier can request more resources.
3. No preemption
   - Previously granted resources cannot be forcibly taken away from process.
4. Circular wait
   - There must be a circular chain of 2 or more processes. Each process is waiting for resource that is held by next member of the chain.

Strategies for dealing with deadlock

1. Just **ignore** the problem.- Ostrich Algorithm
2. **Detection** and **recovery**.
    - Let deadlocks occur, detect them and take action.
3. Dynamic **avoidance** by careful resource allocation. Banker's algorithm.
4. **Prevention**, by structurally negating (killing) one of the four required conditions.

Chapter 3- Memory Management

Computer memory is any physical device capable of storing information temporarily or permanently. Random Access Memory (RAM), is a volatile memory that loses its contents when the computer or hardware device loses power. Read Only Memory (ROM), is a non-volatile memory, sometimes abbreviated as NVRAM, is a memory that keeps its contents even if the power is lost. Computer uses special ROM called BIOS (Basic Input Output System) which permanently stores the software needed to access computer hardware such as hard disk and then load an operating system into RAM and start to execute it. Programmable Read-Only Memory (PROM), is a memory chip on which you can store a program. But once the PROM has been used, you cannot wipe it clean and use it to store something else. Like ROMs, PROMs are non-volatile. E.g CD-R. Erasable Programmable Read-Only Memory (EPROM), is a special type of PROM that can be erased by exposing it to ultraviolet light. E.g CD-RW Electrically Erasable Programmable Read-Only Memory (EEPROM), is a special type of PROM that can be erased by exposing it to an electrical charge. E.g Pendrive. Memory is an important resource that must be carefully managed. A part of operating system that manages memory hierarchy is called **memory manager.** Memory manger is responsible for **Tracking which parts are in use and which parts are not in use. Allocating and deallocates** memory to processes and Managing **swapping** between main memory and disk.

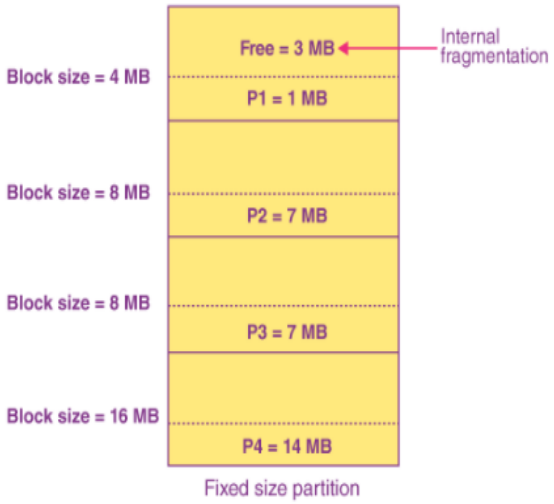
Swapping

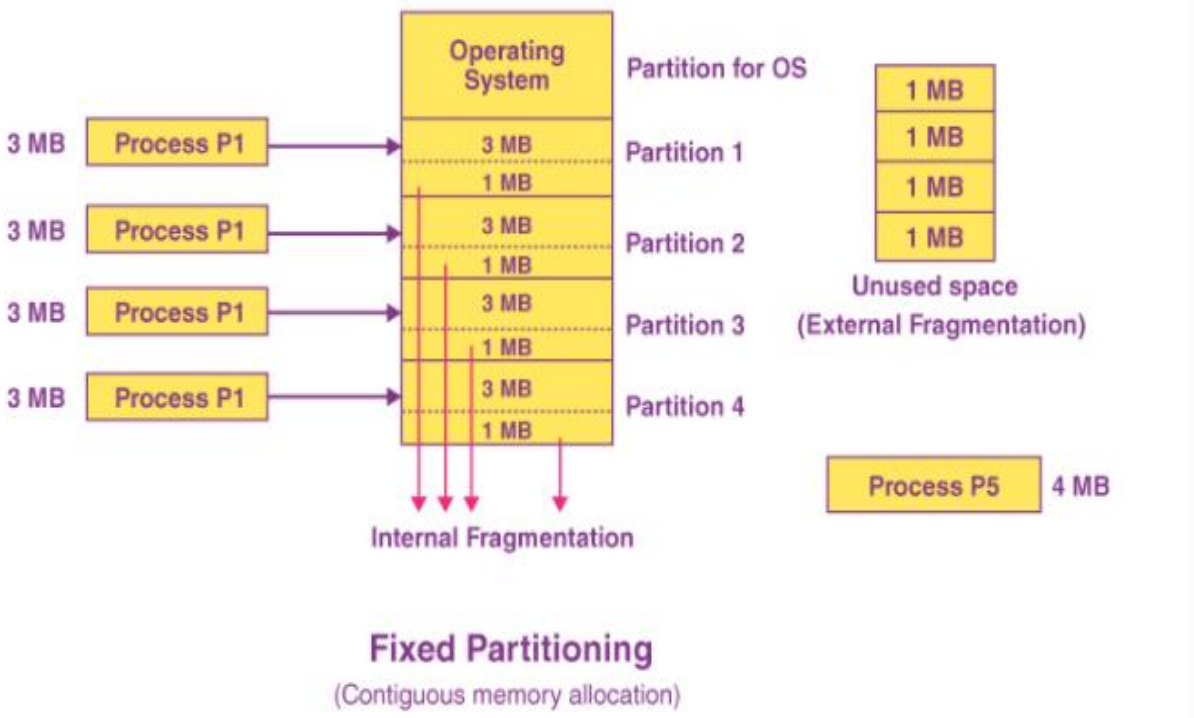
The process of bringing in each process in its entirely in to memory, running it for a while and then putting it back on the disk is called swapping.

Multiprogramming with fixed partitions

Here memory is divided into fixed sized partitions. Size can be equal or unequal for different partitions. Generally unequal partitions are used for better utilizations. Each partition can accommodate exactly one process, means only single process can be placed in one partition. The **partition boundaries** are **not movable.** Whenever any program needs to be loaded in memory, **a free partition big enough to hold the program is found**. This partition will be allocated to that program or process. If there is **no free partition available of required size**, then the process **needs to wait**. Such process will be put in a queue. **There are two ways to maintain queue.**

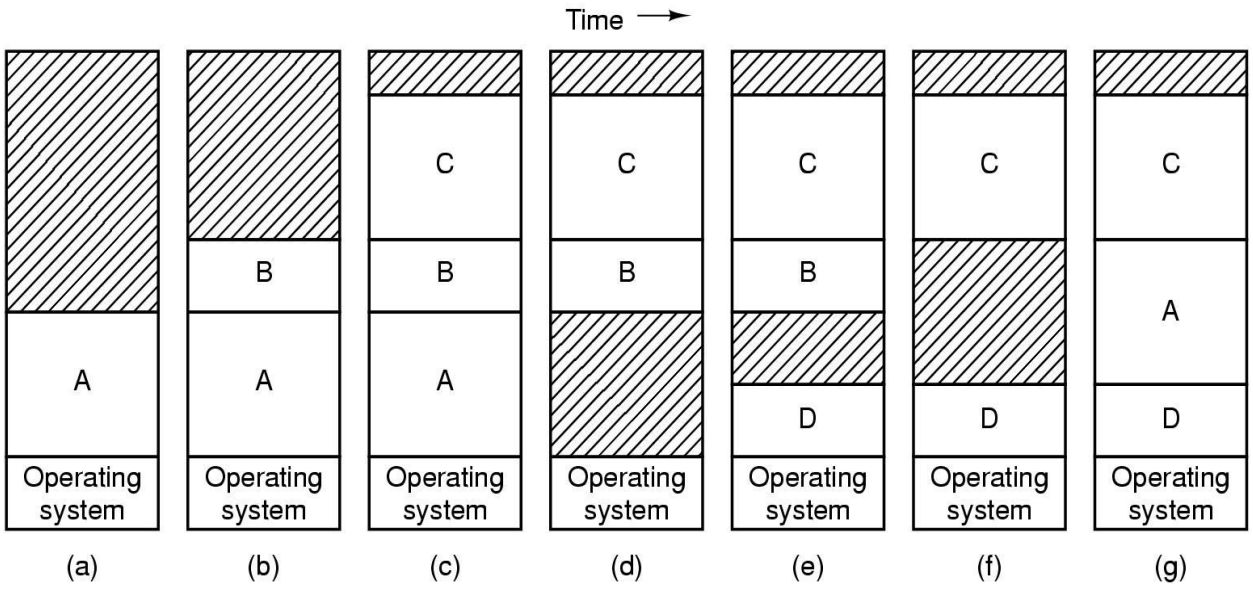
- ✓ **Using Multiple Input Queues.**
- ✓ **Using Single Input Queue.**





Multiprogramming with dynamic partitions

Here, memory is **shared among operating system** and various **simultaneously running processes**. Process is **allocated exactly as much memory as it requires**. Initially, the **entire available memory is treated as a single free partition**. Whenever any **process enters** in a system, a **chunk of free memory big enough to fit the process is found and allocated**. The **remaining unoccupied space is treated as another free partition**. If **enough free memory is not available** to fit the process, **process needs to wait** until required memory becomes available. Whenever any **process gets terminate**, it **releases the space occupied**. If the released free space is contiguous to another free partition, both the free partitions are merged together in to single free partition. **Better utilization** of memory than fixed sized size partition.



Memory compaction

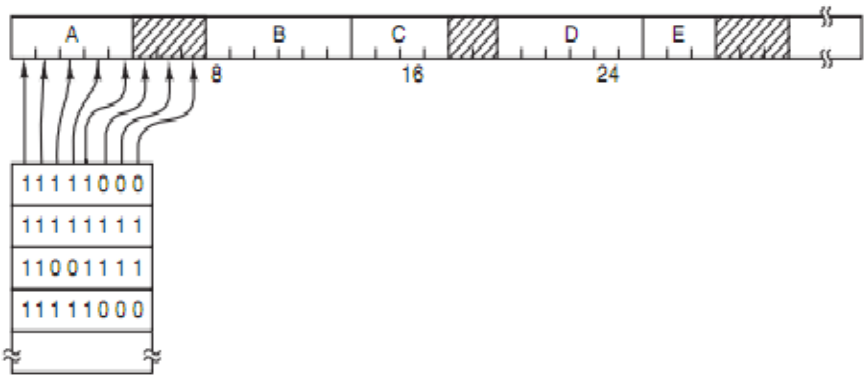
When swapping creates multiple holes in memory, it is possible to combine them all in one big hole by moving all the processes downward as far as possible. This techniques is known as memory compaction. It requires lot of CPU time.

Managing free memory

Two ways to keep track of memory usage (free memory)

Memory management with Bitmaps

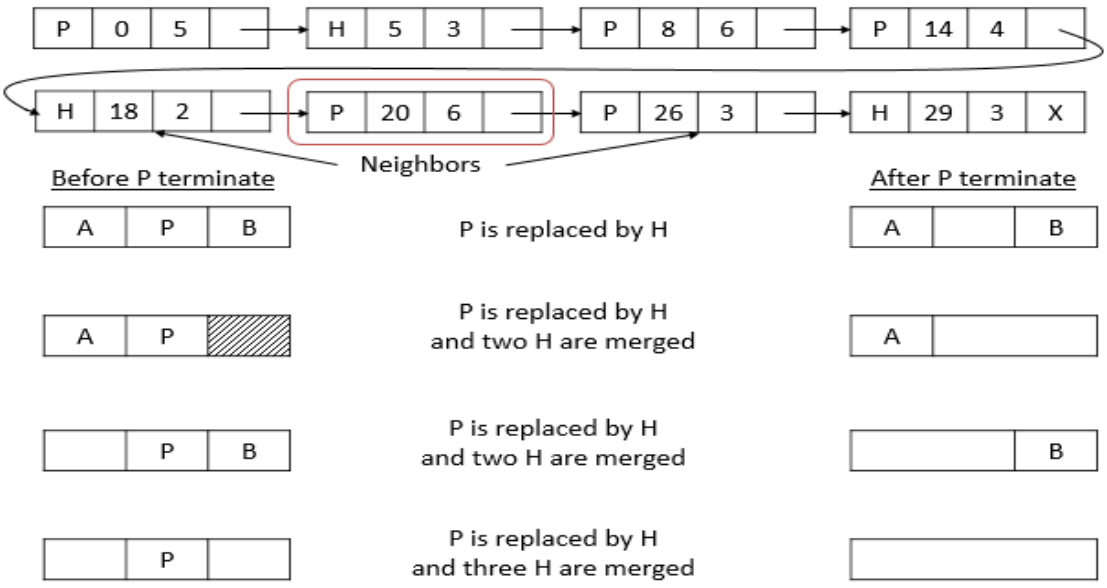
With bitmap, memory is divided into allocation units. Corresponding to each allocation unit there is a bit in a bitmap. Bit is 0 if the unit is free and 1 if unit is occupied. The size of allocation unit is an important design issue, the smaller the size, the larger the bitmap.



The main problem is that when it has been decided to bring a k unit process, the memory manager must search the bitmap to find a run of k consecutive 0 bits in the map. Searching a bitmap for a run of a given length is a slow operation.

Memory management with Linked List

Another way to keep track of memory is to maintain a linked list of allocated and free memory segments, where segment either contains a process or is an empty hole between two processes. Each entry in the list specifies a hole (H) or process (P), the address at which it starts the length and a pointer to the next entry. The segment list is kept sorted by address. Sorting this way has the advantage that when a process terminates or is swapped out, updating the list is straightforward. A terminating process normally has two neighbors (except when it is at the very top or bottom of memory).

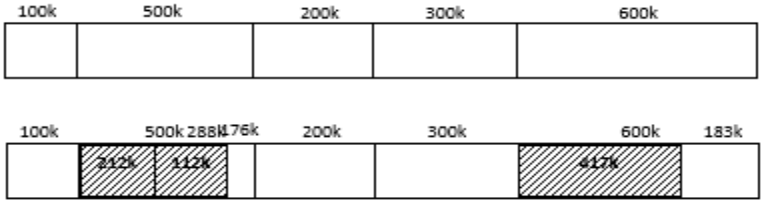


Memory allocation algorithms

First fit

Search **starts from the starting location** of the memory. **First available hole** that is large enough to hold the process is selected for allocation. The hole is then **broken up into two pieces**, **one for process** and **another for unused memory**. Example: Processes of *212K, 417K, 112K and 426K* arrives in order.

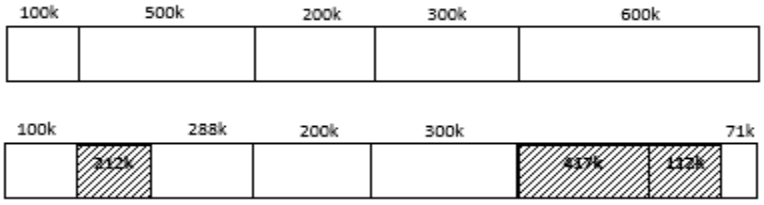
Here process of size **426k will not get any partition** for allocation.



Fastest algorithm because it **searches as little as possible**. **Memory loss is higher**, as very large hole may be selected for small process. Here process of size 426k will not get any partition for allocation.

Next fit

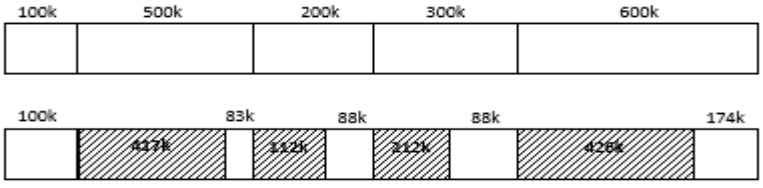
It works in the same way as first fit, except that it keeps the track of where it is whenever it finds a suitable hole. The next time when it is called to find a hole, it starts searching the list from the place where it left off last time. Processes of 212K, 417K, 112K and 426K arrives in order.



Here process of size **426k will not get any partition** for allocation. **Search time is smaller**. Memory manager must have to **keep track of last allotted hole** to process. It gives **slightly worse performance** than first fit.

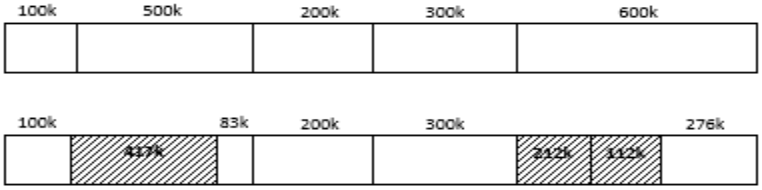
Best fit

Entire memory is searched here. The **smallest hole**, which is **large enough to hold the process**, is **selected for allocation**. Processes of *212K, 417K, 112K and 426K* arrives in order. **Search time is high**, as it searches entire memory every time. **Memory loss is less**.



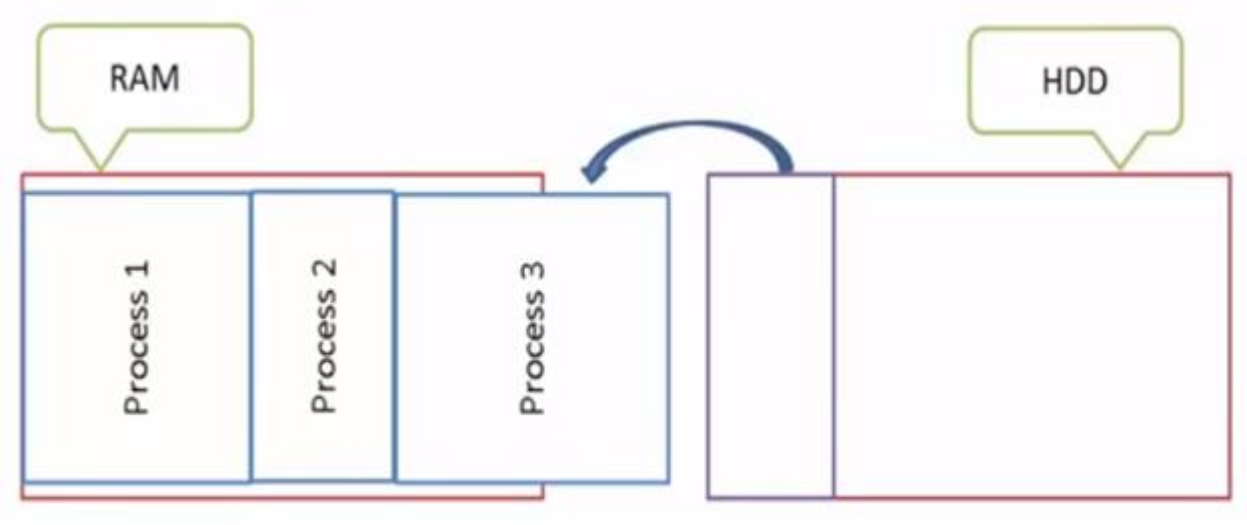
Worst fit

Entire memory is searched here also. The largest hole, which is largest enough to hold the process, is selected for allocation. **Processes of 212K, 417K, 112K and 426K arrives in order. Search time is high**, as it searches entire memory every time. This algorithm can be **used only with dynamic partitioning**. Here process of size **426k will not get any partition** for allocation.

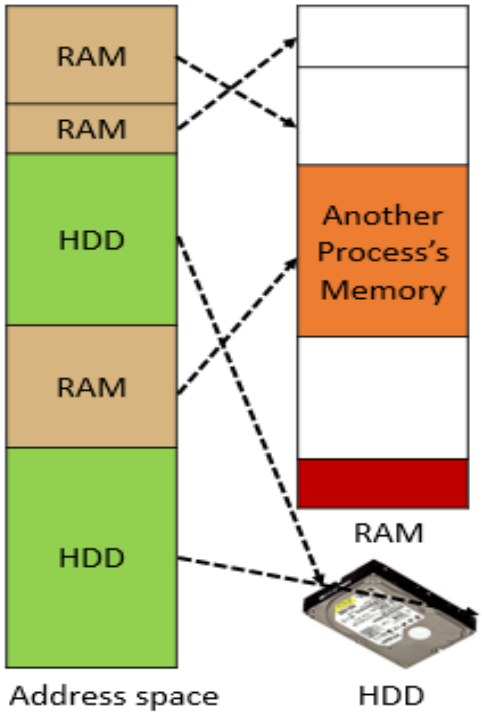


Virtual Memory and Paging

Memory is hardware that your computer uses to **load the operating system and run programs**. Computer consists of **one or more RAM chips** that each have several memory modules. The **amount of real memory in a computer is limited** to the amount of RAM installed. Common memory sizes are 1GB, 2GB, and 4GB. Because your **computer has a finite amount of RAM**, it is possible to **run out of memory when too many programs are running** at one time. Virtual memory **increases the available memory your computer has by enlarging the "address space," or places in memory where data can be stored**. It does this **by using hard disk space** for additional memory allocation. However, since the **hard drive is much slower than the RAM**, data stored in **virtual memory must be mapped back to real memory** in order to be used.



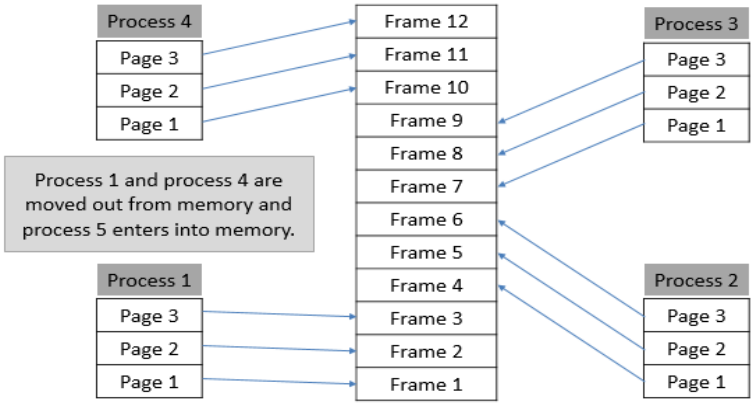
Each program has its own address space, which is broken up into pages. Each **page is a contiguous range of addresses**. These **pages are mapped onto the physical memory** but, to **run the program**, **all pages are not required to be present** in the physical memory. The operating system **keeps those parts of the program currently in use in main memory**, and the **rest on the disk**. In a system using virtual memory, the **physical memory is divided into page frames** and the **virtual address space is divided in into equally-sized partitions called pages**. Virtual memory works fine in a multiprogramming system, with bits and pieces of many programs in memory at once.



Paging

Paging is a storage **mechanism used to retrieve processes from the secondary storage (Hard disk) into the main memory (RAM)** in the form of pages. The main idea behind the paging is to **divide each process in the form of pages**. The main memory will also be divided in the form of frames. One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.

Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage. The **sizes of each frame must be equal**. Considering the fact that the pages are mapped to the frames in Paging, **page size needs to be as same as frame size**. Different operating system defines different frame sizes.



Size of Virtual Address Space is greater than that of main memory, so instead of loading entire address space in to memory to run the process, MMU copies only required pages into main memory. In order to keep the track of pages and page frames, OS maintains a data structure called page table.

Logical Address vs Physical Address

Logical Address is generated by CPU while a program is running. The logical address is virtual address as it does not exist physically therefore it is also known as Virtual Address. Physical Address identifies a physical location of required data in a memory. The user never directly deals with the physical address but can access by its corresponding logical address. The hardware device called Memory-Management Unit is used for mapping (converting) logical address to its corresponding physical address. When virtual memory is used, the **virtual address is presented to an MMU** (Memory Management Unit) that **maps the virtual addresses onto the physical memory addresses**. The virtual address is split into a virtual page number (high order bits) and an offset (low-order bits). With a 16-bit address and a 4KB page size, the upper 4 bits could specify one of the 11 virtual pages and the lower 12 bits would then specify the byte offset (0 to 4095) within the selected page. The virtual page number is used as an index into the Page table. If the present/absent bit is 0, it is page-fault; a trap to the operating system is caused to bring required page into main memory. If the present/absent bit is 1, required page is there with main memory and page frame number found in the page table is copied to the higher order bit of the output register along with the offset. Together Page frame number and offset creates physical address.

Physical Address = Page frame Number + offset of virtual address.

Page table

Page table is a data structure which translates virtual address into equivalent physical address. The virtual page number is used as an index into the Page table to find the entry for that virtual page and from the Page table physical page frame number is found. Thus the purpose of page table is to map virtual pages onto page frames.

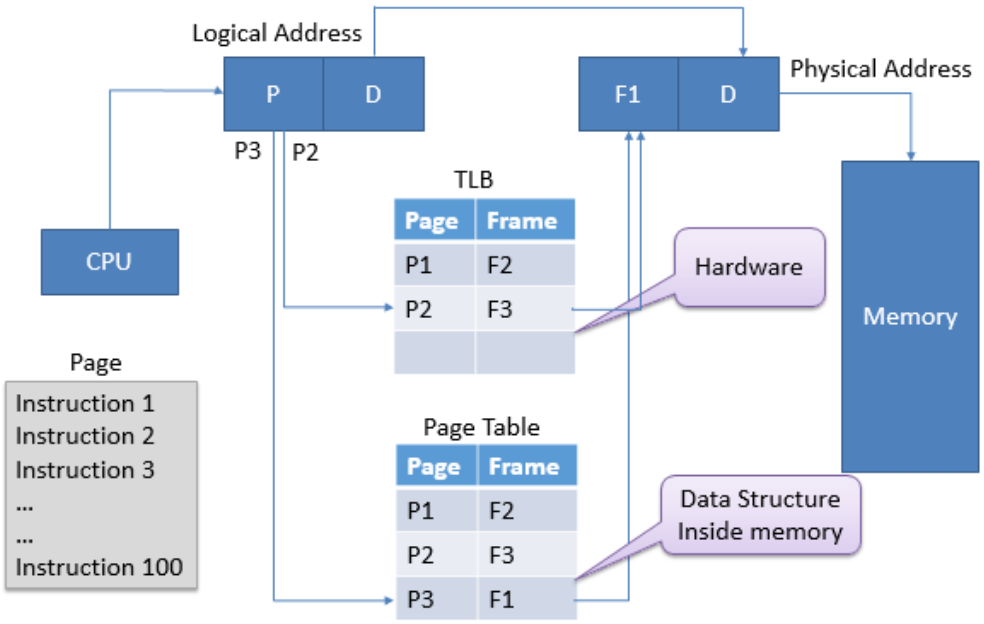
Demand Paging

In paging system, processes are started up with none of their pages in memory. When CPU tries to fetch the first instruction, it gets page fault, other page faults for global variables and stack usually follow quickly. After a while, the process has most of the pages it needs in main memory and it has few page faults. This strategy is called demand paging because pages are loaded only on demand, not in advance.

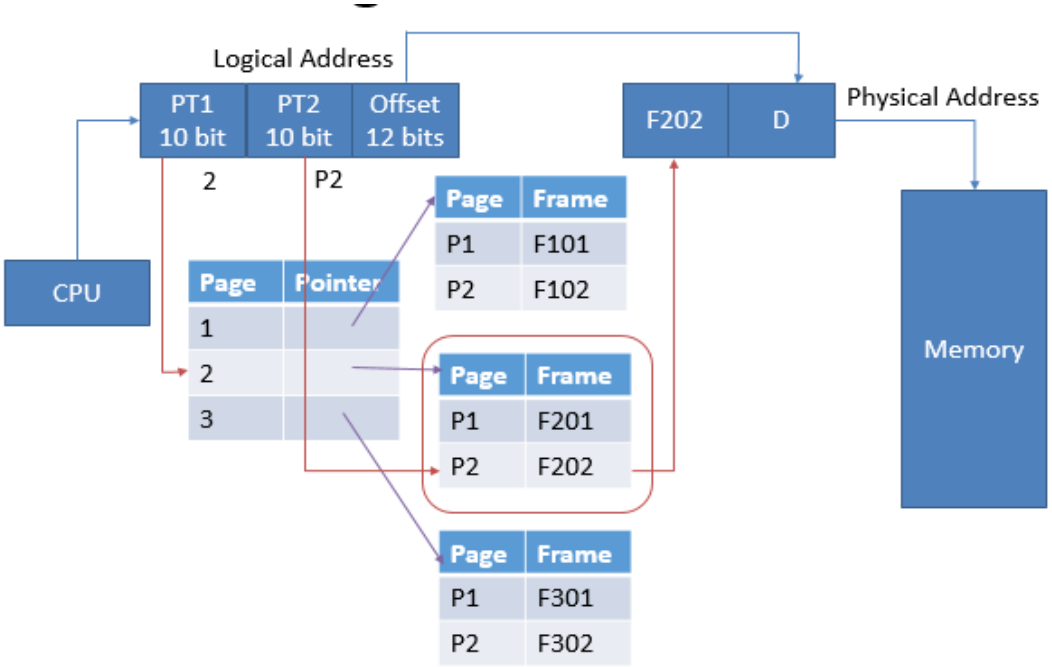
Issues in Paging

In any paging system, two major issues must be faced:

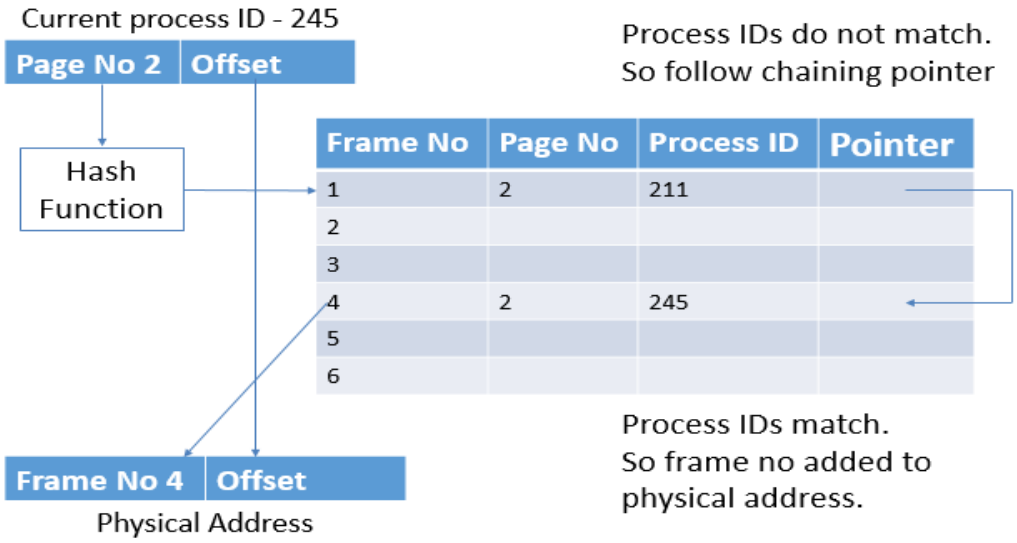
- ✓ The mapping from virtual address to physical address must be fast.
  Solution: using TLB



- ✓ If the virtual address space is large, the page table will be large.
  Solution 1: using Multilevel Page Table



Solution 2: using Inverted Page Table



Page replacement algorithms

Optimal Page Replacement Algorithm

The moment a page fault occurs, some set of pages will be in the memory. One of these pages will be referenced on the very next instruction. Other pages may not be referenced until 10, 100, or perhaps 1000 instructions later. Each page can be labeled with the number of instructions that will be executed before that page is first referenced. The optimal page algorithm simply says that the page with the highest label should be removed. The only problem with this algorithm is that it is unrealizable. At the time of the page fault, the operating system has no way of knowing when each of the pages will be referenced next.

Example

- Page Reference String:
  - 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 0, 2, 0, 1, 7, 0, 1
  - Three frames

Page Requests	7	0	1	2	0	3	0	4	2	3	0	3	2	0	2	0	1	7	0	1
Frame 1	7	7	7	2	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1	1
Frame 2		0	0	0	0	0	0	4	4	4	0	0	0	0	0	0	0	0	0	0
Frame 3			1	1	1	3	3	3	3	3	3	3	3	3	3	3	3	7	7	7
Page Faults (9)	F	F	F	F		F		F			F						F	F		

FIFO Page Replacement Algorithm

The first in first out page replacement algorithm is the simplest page replacement algorithm. The operating system **maintains a list of all pages currently in memory**, with the **most recently arrived page at the tail and least recent at the head**. On a page fault, the **page at head is removed** and the **new page is added to the tail**. When a page replacement is required the **oldest page in memory needs to be replaced**. The performance of the FIFO algorithm is not always good because **it may happen that the page which is the oldest is frequently referred by OS**. Hence removing the oldest page may create page fault again.

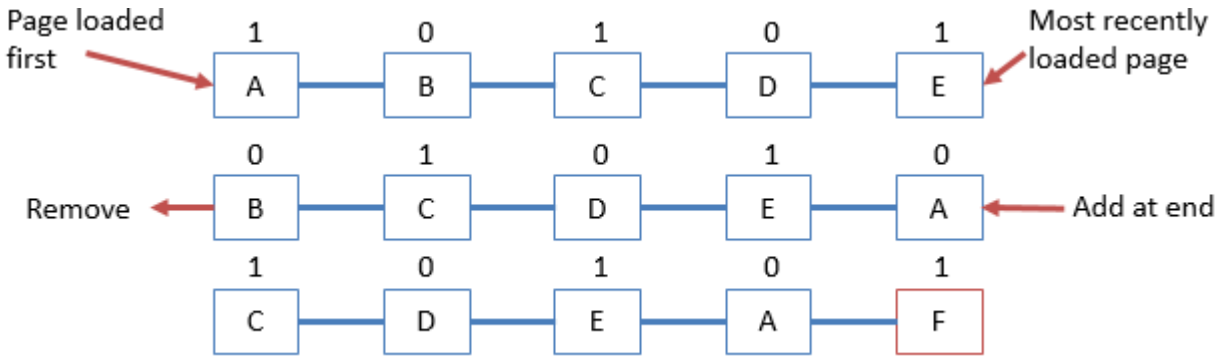
Example

- Page Reference String:
  - 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
  - Three frames

Page Requests	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Frame 1	7	7	7	2	2	2	2	4	4	4	0	0	0	0	0	0	0	7	7	7
Frame 2		0	0	0	0	3	3	3	2	2	2	2	2	1	1	1	1	1	0	0
Frame 3			1	1	1	1	0	0	0	3	3	3	3	3	2	2	2	2	2	1
Page Faults (15)	F	F	F	F		F	F	F	F	F	F			F	F			F	F	F

Second Chance Page Replacement Algorithm

It is modified form of the FIFO page replacement algorithm. It looks at the front of the queue as FIFO does, but instead of immediately paging out that page, it checks to see if its referenced bit is set. If it is not set (zero), the page is swapped out. Otherwise, the referenced bit is cleared, the page is inserted at the back of the queue (as if it were a new page) and this process is repeated.



LRU (Least Recently Used) Page Replacement Algorithm

A good approximation to the optimal algorithm is based on the observation that pages that have been heavily used in last few instructions will probably be heavily used again in next few instructions. When page fault occurs, throw out the page that has been used for the longest time. This strategy is called LRU (Least Recently Used) paging. To fully implement LRU, it is necessary to maintain a linked list of all pages in memory, with the most recently used page at the front and the least recently used page at the rear. The list must be updated on every memory reference. Finding a page in the list, deleting it, and then moving it to the front is a very time consuming operations.

Page Reference String:

- 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1
- Three frames

Page Requests	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
Frame 1	7	7	7	2	2	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
Frame 2		0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
Frame 3			1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7
Page Faults (12)	F	F	F	F		F		F	F	F	F			F		F		F		

Belady's Anomaly

Belady's anomaly is the phenomenon in which increasing the number of page frames results in an increase in the number of page faults for certain memory access patterns. This phenomenon is commonly experienced when using the first-in first-out (FIFO) page replacement algorithm. In FIFO, the page fault may or may not increase as the page frames increase, but in Optimal and LRU Algorithms, as the page frames increase the page fault decreases.

- Page Reference String:
  - 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

Page Faults of 3 Frame < Page Faults of 4 Frame

Belady's Anomaly

Three Frames

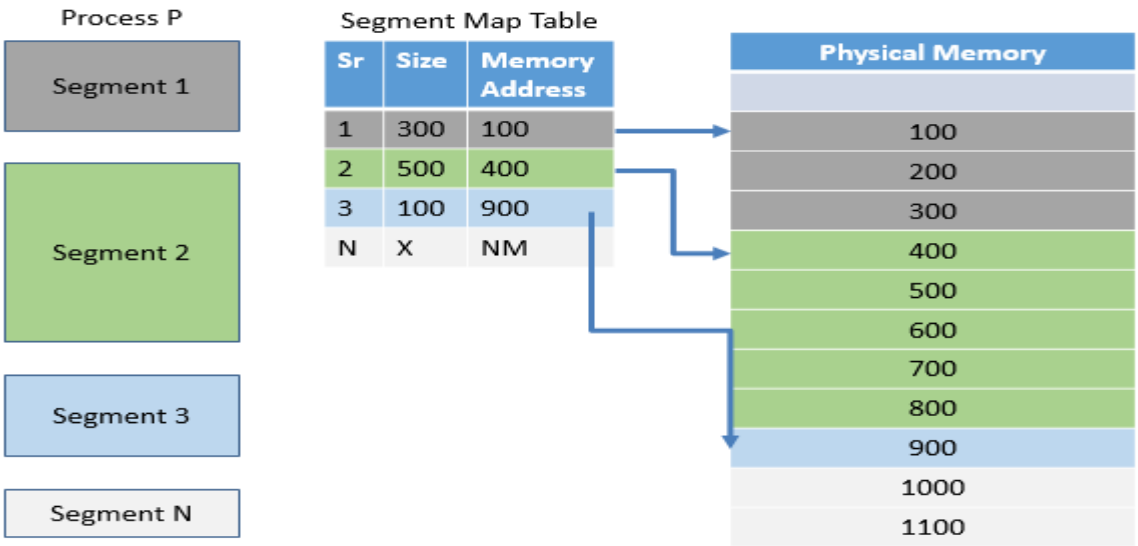
Page Requests	1	2	3	4	1	2	5	1	2	3	4	5
Frame 1	1	1	1	4	4	4	5	5	5	5	5	5
Frame 2		2	2	2	1	1	1	1	1	3	3	3
Frame 3			3	3	3	2	2	2	2	2	4	4
Page Faults **(9)**	F	F	F	F	F	F	F			F	F	

Four Frames

Page Requests	1	2	3	4	1	2	5	1	2	3	4	5
Frame 1	1	1	1	1	1	1	5	5	5	5	4	4
Frame 2		2	2	2	2	2	2	1	1	1	1	5
Frame 3			3	3	3	3	3	3	2	2	2	2
Frame 4				4	4	4	4	4	4	3	3	3
Page Faults **(10)**	F	F	F	F			F	F	F	F	F	F

Segmentation

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. Segment map table contains 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.

Paging	Segmentation
Paging was invented to get large address space without having to buy more physical memory.	Segmentation was invented to allow programs and data to be broken up into logically independent address space and to add sharing and protection.
The programmer does not aware that paging is used.	The programmer is aware that segmentation is used.
Procedure and data cannot be distinguished and protected separately.	Procedure and data be distinguished and protected separately.
Change in data or procedure requires compiling entire program.	Change in data or procedure requires compiling only affected segment not entire program.
Sharing of different procedures not available.	Sharing of different procedures available.

Chapter 4: I/O Management

I/O Devices

BLOCK device can be defined as one that stores information in fixed-size blocks, each one with its own address. e.g. Hard disks, CD-ROMs, and USB
CHARACTER device delivers or accepts a stream of characters, without regard to any block structure. e.g. Printers, NIC, and mice

Components of I/O devices

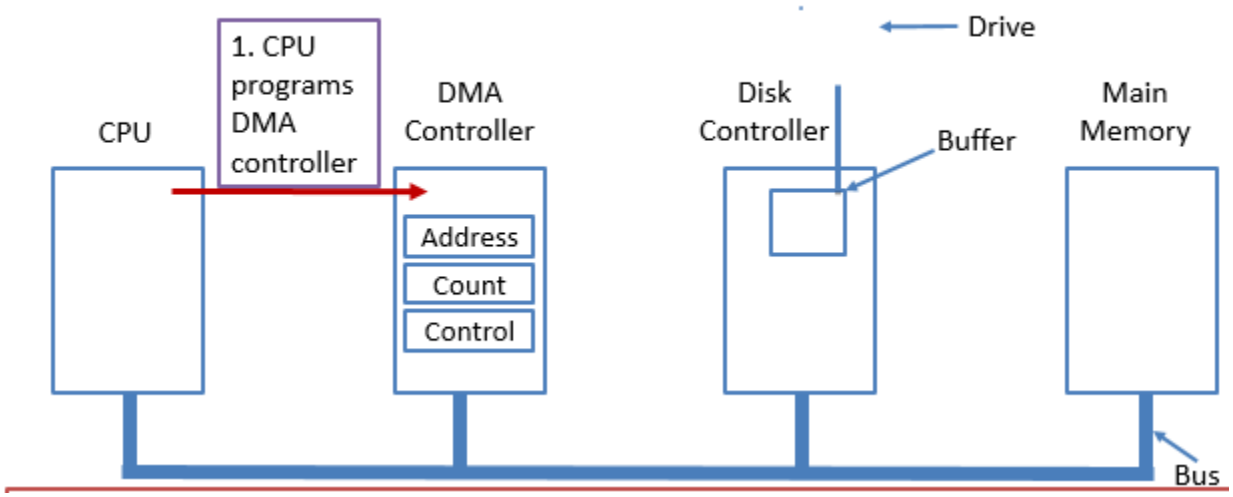
I/O devices have two components. Mechanical component and Electronic component. The mechanical component is device itself. The Electronic component of devices is called the Device Controller. Device Controller is Electronic component which controls the device. It may handle multiple devices. There may be more than one controller per mechanical component (example: hard drive). Controller's tasks are: It converts serial bit stream to block of bytes, perform error correction if necessary, Block of bytes is first assembled bit by bit in buffer inside the controller and after verification, the block has been declared to be error free, and then it can be copied to main memory.

Ways to perform I/O

- ✓ **Programmed I/O :** In programmed I/O, the data transfer is accomplished through an I/O port and are controlled by software.
- ✓ **Interrupt driven I/O** : In interrupt driven I/O, the I/O device will interrupt the processor, and initiate data transfer.
- ✓ **Direct memory access (DMA)** : In DMA, the data transfer between memory and I/O can be performed by bypassing the microprocessor.

Direct Memory Access

Feature of computer systems that allows certain hardware subsystems to access main memory (RAM), independent of the central processing unit (CPU). Without DMA, when the CPU is using programmed input/output, it is typically fully occupied for the entire duration of the read or write operation, and is thus unavailable to perform other work. With DMA, the **CPU first initiates the transfer, then it does other operations while the transfer is in progress**, and it finally receives an interrupt from the DMA controller when the operation is done. This feature is **useful when the CPU needs to perform useful work while waiting** for a relatively slow I/O data transfer. Many hardware systems such as disk drive controllers, graphics cards, network cards and sound cards use DMA.



Step 1: First the CPU programs the DMA controller by setting its registers so it knows what to transfer where. It also issues a command to the disk controller telling it to read data from the disk into its internal buffer and verify the checksum. When valid data are in the disk controller's buffer, DMA can begin.

Step 2: The DMA controller initiates the transfer by issuing a read request over the bus to the disk controller. This read request looks like any other read request, and the disk controller does not know (or care) whether it came from the CPU or from a DMA controller. Typically, the memory address to write to is on the bus' address lines, so when the disk controller fetches the next word from its internal buffer, it knows where to write it.

Step 3: The write to memory is another standard bus cycle.

Step 4: When the write is complete, the disk controller sends an acknowledgement signal to the DMA controller, also over the bus. The DMA controller then increments the memory address to use and decrements the byte count. If the byte count is still greater than 0, steps 2 to 4 are repeated until it reaches 0.

Chapter 5: File Management

File

A file is a **unit of storing data on a secondary storage device** such as a hard disk or other external media. Every **file has a name** and its **data**. Operating system associates various information with files. *For example the date and time of the last modified file and the size of file etc.* This information is called the **file's attributes** or **metadata**. The attributes varies considerably from system to system.

File attributes

1. **Protection** - **Who can access** the file and in **what way**.
2. **Password** - **Password needed to access** the file.
3. **Creator** - **ID** of the person **who created the file**.
4. **Owner** - Current **owner**.
5. **Read only flag** - 0 for read/write, 1 for read only.
6. **Hidden flag** - 0 for normal, 1 for do not display the listings.
7. **System flag** - 0 for normal, 1 for system file.
8. **Archive flag** - 0 for has been backed up, 1 for needs to be backed up.
9. **Random access flag** - 0 for sequential access only, 1 for random access.
10. **Temporary flag** - 0 for normal, 1 for delete file on process exit.
11. **Lock flag** - 0 for unlocked, 1 for locked.
12. **Record length** - Number of bytes in a record.
13. **Key position** - Offset of the key within each record.
14. **Key length** - Number of bytes in key field.
15. **Creation time** - Date and time the file was created.

16. **Time of last access** - Date and time of file was last accessed.
17. **Time of last change** - Date and time of file was last changed.
18. **Current size** - Number of bytes in the file.
19. **Maximum size** - Number of bytes the file may grow to.

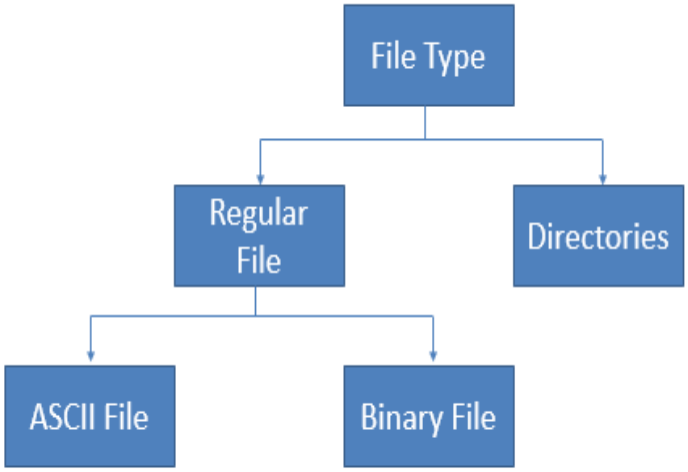
Types of files

Unix have regular files, directories, character special files and block special files. Windows have regular files and directories.

Regular File: Regular files are the ones 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.

Directories: Directories are system files for maintaining the structure of the file system. To keep track of files, file systems normally have directories or folder.

ASCII files: ASCII file consists of line of text. Advantage of ASCII files is that they can be displayed & printed as it is & they can be edited with ordinary text editor. If number of programs use ASCII files for input and output, it is easy to connect the output of one program to the input of another. C/C++/Perl/HTML files are all examples of ASCII files.



Binary Files: Binary files contain formatted information that only certain applications or processors can understand. Binary files must be run on the appropriate software or processor before humans can read them. Executable files, compiled programs, spreadsheets, compressed files, and graphic (image) files are all examples of binary files.

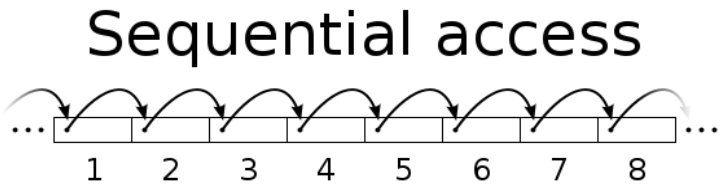
Device Files: Under Linux and UNIX each and every hardware device is treated as a file. A device file allows to accesses hardware devices so that end users do not need to get technical details about hardware. In short, a device file (also called as a special file) is an interface for a device driver that appears in a file system as if it were an ordinary file. This allows software to interact with the device driver using standard input/output system calls, which simplifies many tasks.

Character Special Files: It is a type of device file which **talks to devices in a character by character** (1 byte at a time). Character Special files are related to input/output and use to model serial I/O devices, such as terminals, printers, and networks.

Block Special Files: It is a type of device file which **talks to devices 1 block at a time** (1 block = 512 bytes to 32KB). Block special files are used to model disks, DVD/CD ROM, and memory regions etc.

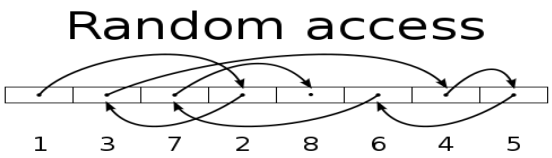
Files access methods

Sequential File Access: In sequential access, process could read all the bytes or records from a file in order, starting at the beginning, but could not skip around and read them out of order. Sequential files could be rewound, however, so they could be read as often as needed. These files were convenient when the storage medium was magnetic tape.



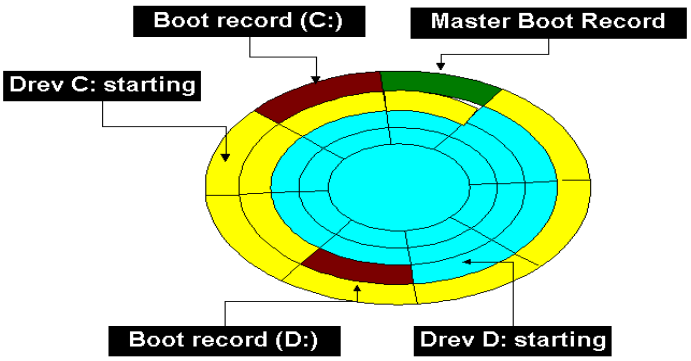
Random File Access

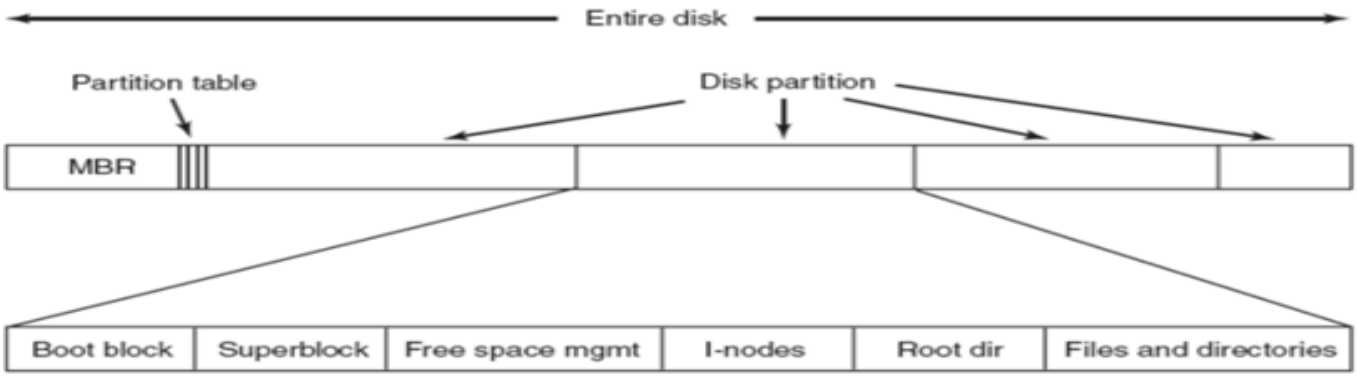
Files **whose bytes or records can be read in any order** are called random access files. Random access files are essentials for many applications, for example, data base systems. Example: If an airline customer calls up and wants to reserve a seat on a particular flight, the reservation program must be able to access the record for that flight without having to read the records for thousands of other flights.



MBR (Master Boot Record)

The Master Boot Record (MBR) is the **information in the first sector** (sector 0) of hard disk. It identifies **how and where an operating system is located** so that it can be boot (loaded) into the computer's main storage or random access memory (RAM). It is **used to boot** the computer. The end of the MBR contains the partition table which gives the starting and ending addresses of each partition





Chapter 6: Security and Protection

Protection

Control access by **limiting file types** accessed by different users. **Only authorized processes** can operate on memory segments, CPU and other resources.

Security

Protect information integrity by ensuring authentication of system users. Prevent unauthorized access Prevent malicious destruction of data. Prevent accidental introduction of inconsistency. Security takes into consideration the **protection system** which is strictly **internal**, as well as the **external** environment in which the **system operates**. Security violations can be **malicious** or **accidental.**

Access Control

Role-Based Access Control, RBAC, assigns privileges to users, programs, or roles as appropriate, where "privileges" refer to the right to call certain system calls, or to use certain parameters with those calls. RBAC supports the principle of least privilege, and reduces the susceptibility to abuse as opposed to SUID or SGID programs.

Security problem

- ✓ Breach of Confidentiality - Theft of private or confidential information, such as credit-card numbers, trade secrets, patents, secret formulas, manufacturing procedures, medical information, financial information, etc.
- ✓ Breach of Integrity - Unauthorized modification of data, which may have serious indirect consequences. For example a popular game or other program's source code could be modified to open up security holes on users systems before being released to the public.
- ✓ Breach of Availability - Unauthorized destruction of data, often just for the "fun" of causing havoc and for bragging rites. Vandalism of web sites is a common form of this violation.
- ✓ Theft of Service - Unauthorized use of resources, such as theft of CPU cycles, installation of daemons running an unauthorized file server, or tapping into the target's telephone or networking services.
- ✓ Denial of Service, DOS - Preventing legitimate users from using the system, often by overloading and overwhelming the system with an excess of requests for service.
- ✓ One common attack is masquerading, in which the attacker pretends to be a trusted third party. A variation of this is the man-in-the-middle, in which the attacker masquerades as both ends of the conversation to two targets.
- ✓ A replay attack involves repeating a valid transmission. Sometimes this can be the entire attack, (such as repeating a request for a money transfer), or other times the content of the original message is replaced with malicious content.

Operating System Security

User authentication Based on

- User possession (of key or card)
- User knowledge (user identifier + password)
- User attribute (fingerprint, retina pattern, signature)

Program Threats

Trojan horse: A ***Trojan horse*** is a program that secretly performs some maliciousness in addition to its visible actions. Some Trojan horses are deliberately written as such, and others are the result of legitimate programs that have become infected with ***viruses,*** One dangerous opening for Trojan horses is **long search paths**, and in particular paths which include the current directory (".") as part of the path. If a dangerous program having the same name as a legitimate program (or a common miss-spelling, such as "sl" instead of "ls") is placed anywhere on the path, then an unsuspecting user may be fooled into running the wrong program by mistake. Another classic Trojan horse is a **login emulator**, which records a user's account name and password, issues a "password incorrect" message, and then logs off the system. The user then tries again (with a proper login prompt), logs in successfully, and doesn't realize that their information has been stolen. Two solutions to Trojan Horses are to have the system **print usage statistics on logouts**, and to **require the typing of non-trappable key sequences such as Control-Alt-Delete in order to log in**. *(This*

is why modern Windows systems require the Control-Alt-Delete sequence to commence logging in, which cannot be emulated or caught by ordinary programs. I.e. that key sequence always transfers control over to the operating system.)

Spyware is a version of a Trojan horse that is often included in "free" software downloaded off the Internet. Spyware programs generate pop-up browser windows, and may also accumulate information about the user and deliver it to some central site. Another common task of spyware is to send out spam e-mail messages, which then purportedly come from the infected user.

A **Trap Door** is when a designer or a programmer (or hacker) deliberately inserts a security hole that they can use later to access the system. Because of the possibility of trap doors, once a system has been in an untrustworthy state, that system can never be trusted again. Even the backup tapes may contain a copy of some cleverly hidden back door. A clever trap door could be inserted into a compiler, so that any programs compiled with that compiler would contain a security hole. This is especially dangerous, because inspection of the code being compiled would not reveal any problems.

A ***Logic Bomb*** is code that is not designed to cause havoc all the time, but only when a certain set of circumstances occurs, such as when a particular date or time is reached or some other noticeable event. A classic example is the ***Dead-Man Switch***, which is designed to check whether a certain person (e.g. the author) is logging in every day, and if they don't log in for a long time (presumably because they've been fired), then the logic bomb goes off and either opens up security holes or causes other problems.

A virus is a fragment of code embedded in an otherwise legitimate program, designed to replicate itself (by infecting other programs), and (eventually) wreaking havoc. Viruses are more likely to infect PCs than UNIX or other multi-user systems, because programs in the latter systems have limited authority to modify other programs or to access critical system structures (such as the boot block.) Viruses are delivered to systems in a ***virus dropper,*** usually some form of a Trojan Horse, and usually via e-mail or unsafe downloads. Viruses take many forms. Figure below shows typical operation of a boot sector virus.

``````````End``````````