**OPERATING SYSTEMS**

**LECTURE NOTES**

**B.TECH II YEAR – II SEM**

**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

|  |  |
| --- | --- |
| **Subject** | **OPERATING SYSTEMS SYLLABUS** |
| **Branch** | **CSE, IT, AI & DS** |

**COURSE OBJECTIVES:**

* Study the basic concepts and functions of operating systems.
* Understand the structure and functions of the OS.
* Learn about Processes, Threads and Scheduling algorithms.
* Understand the principles of concurrency and Deadlocks.
* Learn various memory management schemes.
* Study I/O management and File systems.

**UNIT I:**

Introduction to Operating System Concept: Types of operating systems, operating systems concepts, operating systems services, Introduction to System call, System call types.

**UNIT-II:**

Process Management – Process concept, The process, Process State Diagram, Process control block, Process Scheduling- Scheduling Queues, Schedulers, Operations on Processes, Inter-process Communication, Threading Issues, Scheduling-Basic Concepts, Scheduling Criteria, Scheduling Algorithms.

**UNIT-III:**

Concurrency & Deadlocks: Process Synchronization, The Critical- Section Problem, Synchronization Hardware, Semaphores, Classic Problems of Synchronization, Monitors, Synchronization examples. Principles of deadlock – System Model, Deadlock Characterization, Deadlock Prevention, Detection and Avoidance, Recovery form Deadlock.

**UNIT-IV:**

Memory Management: Swapping, Contiguous Memory Allocation, Paging, structure of the Page Table, Segmentation, Virtual Memory Management: Virtual Memory, Demand Paging, Page-Replacement Algorithms, Thrashing.

**UNIT-V:**

File system Interface- the concept of a file, Access Methods, Directory structure, File system mounting, and file sharing.

**File System implementation**- File system structure, allocation methods, free-space management Mass-storage structure overview of Mass-storage structure, Disk scheduling.

System Protection: Goals of protection, Principles and Domain of protection.

**TEXT BOOK:**

1. Operating System Concepts, Abraham Silberschatz, Peter Baer Galvin and Greg Gagne 9th Edition, John Wiley and Sons Inc., 2012.

2. Operating Systems – Internals and Design Principles, William Stallings, 7th Edition, Prentice Hall, 2011.

3. Operating Systems-S Halder, Alex A Aravind Pearson Education Second Edition 2016.

**REFERENCES:**

1. Modern Operating Systems, Andrew S. Tanenbaum, Second Edition, Addison Wesley, 2001.

2. Operating Systems: A Design-Oriented Approach, Charles Crowley, Tata Mc Graw Hill Education”, 1996.

3. Operating Systems: A Concept-Based Approach, D M Dhamdhere, Second Edition, Tata Mc Graw-Hill Education, 2007.

**OUTCOMES:**

1. Describe basic concepts & services provided by Operating System (OS).
2. Understand process management & various CPU scheduling algorithms
3. Understand various issues in concurrent execution of processes & dealing deadlocks
4. Understand the how the Operating System managements the memory

5. Understand the file management & protection services offered by OS.

**UNIT-II:**

**Process Management** – Process concept, The process, Process State Diagram, Process control block, Process Scheduling- Scheduling Queues, Schedulers, Operations on Processes, Inter-process Communication, Threading Issues, Scheduling-Basic Concepts, Scheduling Criteria, Scheduling Algorithms.

## Process

A process is a program at the time of execution.

## Differences between Process and Program

|  |  |
| --- | --- |
| **Process** | **Program** |
| Process is a dynamic object | Program is a static object |
| Process is sequence of instruction  execution | Program is a sequence of instructions |
| Process loaded in to main memory | Program loaded into secondary storage  devices |
| Time span of process is limited | Time span of program is unlimited |
| Process is a active entity | Program is a passive entity |

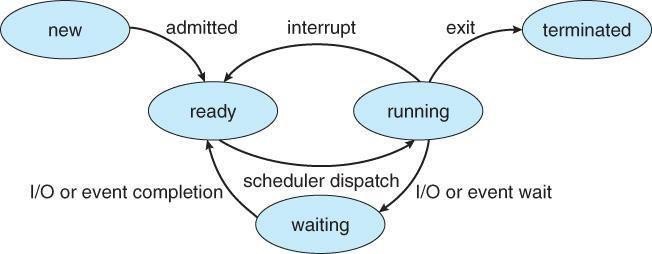
**Process States**

When a process executed, it changes the state, generally the state of process is determined by the current activity of the process. Each process may be in one of the following states:

1. New : The process is being created.
2. Running : The process is being executed.
3. Waiting : The process is waiting for some event to occur.
4. Ready : The process is waiting to be assigned to a processor.
5. Terminated : The Process has finished execution.

Only one process can be running in any processor at any time, But many process may be in ready and waiting states. The ready processes are loaded into a “ready queue”.

## Diagram of process state



1. **New ->Ready** : OS creates process and prepares the process to be executed, then OS moved the process into ready queue.
2. **Ready->Running** : OS selects one of the Jobs from ready Queue and move them from ready to Running.
3. **Running->Terminated** : When the Execution of a process has Completed, OS terminates that process from running state. Sometimes OS terminates the process for some other reasons including Time exceeded, memory unavailable, access violation, protection Error, I/O failure and soon.
4. **Running->Ready** : When the time slot of the processor expired (or) If the processorreceivedanyinterrupt signal, the OS shifted Running -> Ready State.
5. **Running -> Waiting** : A process is put into the waiting state, if the process need an event occur (or) an I/O Devicerequire.
6. **Waiting->Ready** : A process in the waiting state is moved to ready state when the event for which it has been Completed.

## Process Control Block:

Each process is represented in the operating System by a Process Control Block.

It is also called Task Control Block. It contains many pieces of information associated with a specific Process.

|  |
| --- |
| Process State |
| Program Counter |
| CPU Registers |
| CPU Scheduling Information |
| Memory – Management Information |
| Accounting Information |
| I/O Status Information |

## Process Control Block

1. **Process State**: The State may be new, ready, running, and waiting,Terminated.
2. **Program Counter**: indicates the Address of the next Instruction to be executed.
3. **CPU registers**: registers include accumulators, stack pointers, General purpose Registers.
4. **CPU-Scheduling Info: includes** a process pointer, pointers to scheduling Queues, other scheduling parameters etc.
5. **Memory management Info**: includes page tables, segmentation tables, value of base and limit registers.
6. **Accounting Information:** includes amount of CPU used, time limits, Jobs(or)Process numbers.
7. **I/O Status Information**: Includes the list of I/O Devices Allocated to the processes, list of open files.

## Threads:

A process is divide into number of light weight process, each light weight process is said to be a Thread. The Thread has a program counter (Keeps track of which instruction to execute next), registers (holds its current working variables), stack (execution History).

## Thread States:

1. born State : A thread is just created.
2. ready state : The thread is waiting for CPU.
3. Running : System assigns the processor to the thread.
4. Sleep : A sleeping thread becomes ready after the designated sleep time expires**.**
5. dead : The Execution of the thread finished.

## Eg: Word processor.

Typing, Formatting, Spell check, saving are threads.

## Differences between Process and Thread

|  |  |
| --- | --- |
| **Process** | **Thread** |
| Process takes more time to create. | Thread takes less time to create. |
| it takes more time to complete execution &terminate. | Less time to terminate. |
| Execution is very slow. | Execution is very fast. |
| It takes more time to switch b/w two processes. | It takes less time to switch b/w two threads. |
| Communication b/w two processes is difficult . | Communication b/w two threads is easy. |
| Process can’t share the same memory area. | Threads can share same memory area. |
| System calls are requested to communicate each other. | System calls are not required. |
| Process is loosely coupled. | Threads are tightly coupled. |
| It requires more resources to execute. | Requires few resources to execute. |

**Advantages**

* Thread switching does not require Kernel mode privileges.
* User level thread can run on any operating system.
* Scheduling can be application specific in the user level thread.
* User level threads are fast to create and manage.

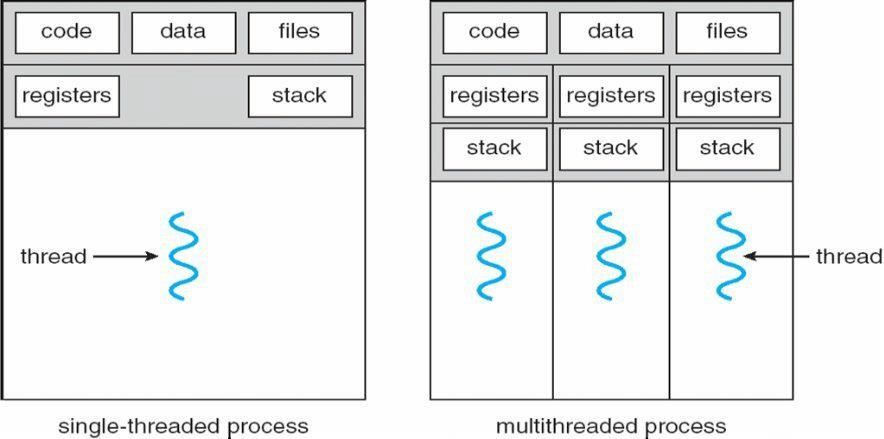
**Multithreading**

A process is divided into number of smaller tasks each task is called a Thread. Number of Threads with in a Process execute at a time is called Multithreading.

If a program is multithreaded, even when some portion of it is blocked, the whole program is not blocked. The rest of the program continues working If multiple CPU’s are available.

Multithreading gives best performance. If we have only a single thread, number of CPU’s available, No performance benefits achieved.

* Process creation is heavy-weight while thread creation is light-weight
* Can simplify code, increase efficiency



 Kernels are generally multithreaded

**CODE-** Contains instruction

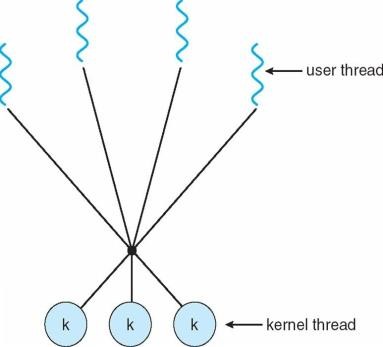
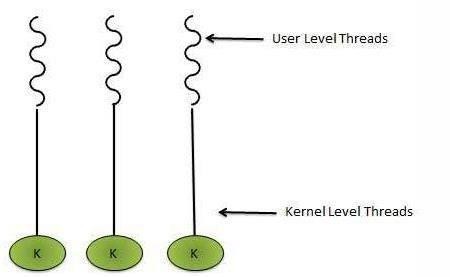
**DATA-** holds global variable **FILES-**opening and closing files

**REGISTER-** contain information about CPU state **STACK-**parameters, local variables, functions

**Types Of Threads:**

**1) User Threads** : Thread creation, scheduling, management happen in user space by Thread Library. user threads are faster to create and manage. If a user thread performs a system call, which blocks it, all the other threads in that process one also automatically blocked, whole process is blocked.

|  |
| --- |
| **OPERATING SYSTEMS NOTES II YEAR/I SEM MRCET**  **Disadvantages**   * In a typical operating system, most system calls are blocking. * Multithreaded application cannot take advantage of multiprocessing.  1. **Kernel Threads**: kernel creates, schedules, manages these threads .these threads are slower, manage. If one thread in a process blocked, over all process need not be blocked.   **Advantages**   * + Kernel can simultaneously schedule multiple threads from the same process on multiple processes.   + If one thread in a process is blocked, the Kernel can schedule another thread of the same process.   + Kernel routines themselves can multithreaded.   **Disadvantages**   * + Kernel threads are generally slower to create and manage than the user threads.   + Transfer of control from one thread to another within same process requires a mode switch to the Kernel.   **Multithreading Models**  Some operating system provides a combined user level thread and Kernel level thread facility. Solaris is a good example of this combined approach. In a combined system, multiple threads within the same application can run in parallel on multiple processors and a blocking system call need not block the entire process. Multithreading models are three types   * Many to many relationship. * Many to one relationship. * One to one relationship.   **Many to Many Model**  In this model, many user level threads multiplexes to the Kernel thread of smaller or equal numbers. The number of Kernel threads may be specific to either a particular application or a particular machine.  Following diagram shows the many to many model. In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor. |



**One to One Model**

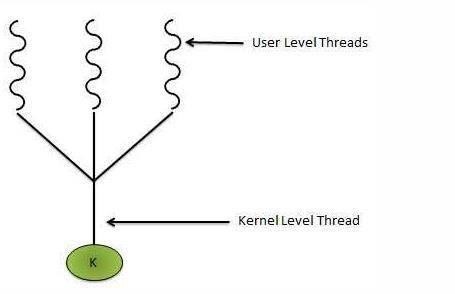
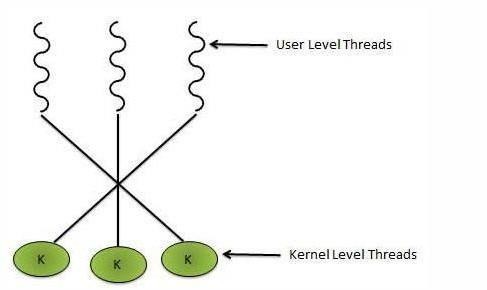
There is one to one relationship of user level thread to the kernel level thread.This model provides more concurrency than the many to one model. It also another thread to run when a thread makes a blocking system call. It support multiple thread to execute in parallel on microprocessors.

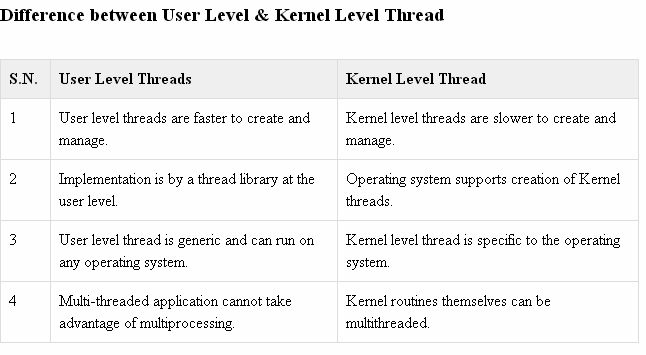
Disadvantage of this model is that creating user thread requires the corresponding Kernel thread. OS/2, windows NT and windows 2000 use one to one relationship model.

**Many to One Model**

Many to one model maps many user level threads to one Kernel level thread. Thread management is done in user space. When thread makes a blocking system call, the entire process will be blocks. Only one thread can access the Kernel at a time,so multiple threads are unable to run in parallel on multiprocessors.

If the user level thread libraries are implemented in the operating system in such a way that system does not support them then Kernel threads use the many to one relationship modes.





## PROCESS SCHEDULING:

CPU is always busy in **Multiprogramming**. Because CPU switches from one job to another job. But in **simple computers** CPU sit idle until the I/O request granted.

**Scheduling** is a important OS function. All resources are scheduled before use.(cpu, memory, devices…..)

Process scheduling is an essential part of a Multiprogramming operating systems. Such operating systems allow more than one process to be loaded into the executable memory at a time and the loaded process shares the CPU using time multiplexing.

**Scheduling Objectives**

Maximize throughput.

Maximize number of users receiving acceptable response times. Be predictable.

Balance resource use.

Avoid indefinite postponement. Enforce Priorities.

Give preference to processes holding key resources

**SCHEDULING QUEUES**: people live in rooms. Process are present in rooms knows as queues. There are 3types

1. **job queue**: when processes enter the system, they are put into a **job queue**, which consists all processes in the system. Processes in the job queue reside on mass storage and await the allocation of main memory.
2. **ready queue**: if a process is present in main memory and is ready to be allocated to cpu forexecution, is kept in **readyqueue.**
3. **device queue**: if a process is present in waiting state (or) waiting for an i/o event to complete is said to bein device queue.(or)

The processes waiting for a particular I/O device is called device queue.

**Schedulers :** There are 3 schedulers

1. Long term scheduler.
2. Medium term scheduler
3. Short term scheduler.

Scheduler duties:

* + Maintains the queue.
  + Select the process from queues assign to CPU.

## Types of schedulers

1. **Long term scheduler:**

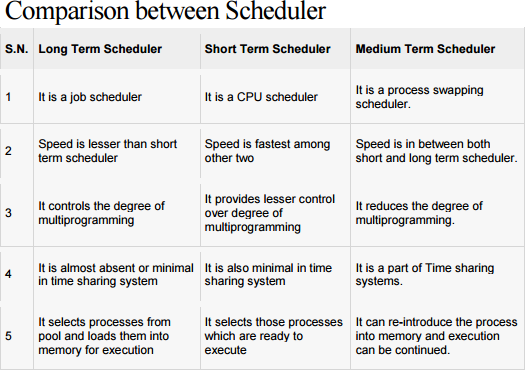
select the jobs from the job pool and loaded these jobs into main memory (ready queue). Long term scheduler is also called job scheduler.

## Short term scheduler:

select the process from ready queue, and allocates it to the cpu.

If a process requires an I/O device, which is not present available then process enters device queue.

short term scheduler maintains ready queue, device queue. Also called as cpu scheduler.

1. **Medium term scheduler**: if process request an I/O device in the middle of the execution, then the process removed from the main memory and loaded into the waiting queue. When the I/O operation completed, then the job moved from waiting queue to ready queue. These two operations performed by medium term scheduler.

**Context Switch**: Assume, main memory contains more than one process. If cpu is executing a process, if time expires or if a high priority process enters into main memory, then the scheduler saves information about current process in the PCB and switches to execute the another process. The concept of moving CPU by scheduler from one process to other process is known as context switch.

**Non-Preemptive Scheduling**: **CPU** is assigned to one process, CPU do not release until the competition of that process. The CPU will assigned to some other process only after the previous process has finished.

**Preemptive scheduling**: here CPU can release the processes even in the middle of the execution. CPU received a signal from process p2. OS compares the priorities of p1 ,p2. If p1>p2, CPU continues the execution of p1. If p1<p2 CPU preempt p1 and assigned to p2.

**Dispatcher:** The main job of dispatcher is switching the cpu from one process to another process. Dispatcher connects the cpu to the process selected by the short term scheduler.

**Dispatcher latency**: The time it takes by the dispatcher to stop one process and start another process is known as dispatcher latency. If the dispatcher latency is increasing, then the degree of multiprogramming decreases.

## SCHEDULING CRITERIA:

1. **Throughput**: how many jobs are completed by the cpu with in a timeperiod.
2. **Turn around time** : The time interval between the submission of the process and time of the completion is turn around time.

## TAT = Waiting time in ready queue + executing time + waiting time in waiting queue for I/O.

1. **Waiting time**: The time spent by the process to wait for cpu to beallocated.
2. **Response time**: Time duration between the submission and firstresponse.
3. **Cpu Utilization**: CPU is costly device, it must be kept as busy aspossible. Eg: CPU efficiency is 90% means it is busy for 90 units, 10 units idle.

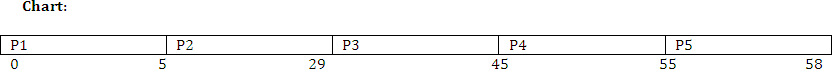
## CPU SCHEDULINGALGORITHMS: [REFER CLASS NOTES and unit2notes send in gclassroom (for advantages and disadvantages)]

**1. First come First served scheduling: (FCFS):** The process that request the CPU first is holds the cpu first. If a process request the cpu then it is loaded into the ready queue, connect CPU to that process.

Consider the following set of processes that arrive at time 0, the length of the cpu burst time given in milli seconds.

**burst time is the time, required the cpu to execute that job, it is in milli seconds.**

|  |  |
| --- | --- |
| **Process** | **Burst time(milliseconds)** |
| P1 | 5 |
| P2 | 24 |
| P3 | 16 |
| P4 | 10 |
| P5 | 3 |



## Average turn around time:

**Turn around time= waiting time + burst time**

Turn around time for p1= 0+5=5. Turn around time for p2=5+24=29 Turn around time for p3=29+16=45 Turn around time for p4=45+10=55 Turn around time for p5= 55+3=58

Average turn around time= (5+29++45+55+58/5) = 187/5 =37.5 millisecounds

## Average waiting time:

**waiting time= starting time- arrival time**

Waiting time for p1=0

Waiting time for p2=5-0=5 Waiting time for p3=29-0=29 Waiting time for p4=45-0=45 Waiting time for p5=55-0=55

Average waiting time= 0+5+29+45+55/5 = 125/5 = 25 ms.

**Average Response Time :**

**Formula :** First Response - Arrival Time Response Time for P1 =0 Response Time for P2 => 5-0 = 5 Response Time for P3 => 29-0 = 29 Response Time for P4 => 45-0 = 45 Response Time for P5 => 55-0 = 55

Average Response Time => (0+5+29+45+55)/5 =>25ms

1. **First Come FirstServe:**

It is Non Primitive Scheduling Algorithm.

|  |  |  |
| --- | --- | --- |
| **PROCESS** | **BURST**  **TIME** | **ARRIVAL**  **TIME** |
| P1 | 3 | 0 |
| P2 | 6 | 2 |
| P3 | 4 | 4 |
| P4 | 5 | 6 |
| P5 | 2 | 8 |

Process arrived in the order P1, P2, P3, P4, P5. P1 arrived at 0 ms.

P2 arrived at 2 ms. P3 arrived at 4 ms. P4 arrived at 6 ms. P5 arrived at 8 ms.



**Average Turn Around Time**

**Formula :** Turn around Time =**:** waiting time + burst time Turn Around Time for P1 => 0+3= 3

Turn Around Time for P2 => 1+6 = 7 Turn Around Time for P3 => 5+4 = 9 Turn Around Time for P4 => 7+ 5= 12 Turn Around Time for P5 => 2+ 10=12

Average Turn Around Time => ( 3+7+9+12+12 )/5 =>43/5 = 8.50 ms.

**Average Response Time :**

**Formula :** Response Time = First Response - Arrival Time Response Time of P1 = 0

Response Time of P2 => 3-2 = 1 Response Time of P3 => 9-4 = 5 Response Time of P4 => 13-6 = 7 Response Time of P5 => 18-8 =10

Average Response Time => ( 0+1+5+7+10 )/5 => 23/5 = 4.6 ms

## Advantages: Easy to Implement, Simple.

**Disadvantage: Average waiting time is very high.**

1. **Shortest Job First Scheduling ( SJF ):**

Which process having the smallest CPU burst time, CPU is assigned to that process . If two process having the same CPU burst time, FCFS is used.

|  |  |
| --- | --- |
| **PROCESS** | **CPU BURST TIME** |
| P1 | 5 |
| P2 | 24 |
| P3 | 16 |
| P4 | 10 |
| P5 | 3 |



P5 having the least CPU burst time ( 3ms ). CPU assigned to that ( P5 ). After completion of P5 short term scheduler search for nest ( P1 ).......

**Average Waiting Time :**

**Formula =** Staring Time - Arrival Time waiting Time for P1 => 3-0 = 3

waiting Time for P2 => 34-0 = 34 waiting Time for P3 => 18-0 = 18 waiting Time for P4 =>8-0=8 waiting time for P5=0

Average waiting time => ( 3+34+18+8+0 )/5 => 63/5 =12.6 ms

**Average Turn Around Time :**

**Formula =** waiting Time + burst Time

Turn Around Time for P1 => 3+5 =8 Turn Around for P2 => 34+24 =58 Turn Around for P3 => 18+16 = 34

Turn Around Time for P4 => 8+10 =18 Turn Around Time for P5 => 0+3 = 3

Average Turn around time => ( 8+58+34+18+3 )/5 => 121/5 = 24.2 ms

**Average Response Time :**

**Formula :** First Response - Arrival Time

First Response time for P1 =>3-0 = 3

First Response time for P2 => 34-0 = 34 First Response time for P3 => 18-0 = 18 First Response time for P4 => 8-0 = 8

First Response time for P5 = 0

Average Response Time => ( 3+34+18+8+0 )/5 => 63/5 = 12.6 ms SJF is Non primitive scheduling algorithm

## Advantages : Least average waiting time Least average turn around time Least average response time

Average waiting time ( FCFS ) = 25 ms

Average waiting time ( SJF ) = 12.6 ms 50% time saved in SJF.

## Disadvantages:

* + - Knowing the length of the next CPU burst time is difficult.
    - Aging ( Big Jobs are waiting for long time for CPU)

1. **Shortest Remaining Time First ( SRTF );**

This is primitive scheduling algorithm.

Short term scheduler always chooses the process that has term shortest remaining time. When a new process joins the ready queue , short term scheduler compare the remaining time of executing process and new process. If the new process has the least CPU burst time, The scheduler selects that job and connect to CPU. Otherwise continue the old process.

|  |  |  |
| --- | --- | --- |
| **PROCESS** | **BURST TIME** | **ARRIVAL TIME** |
| P1 | 3 | 0 |
| P2 | 6 | 2 |
| P3 | 4 | 4 |
| P4 | 5 | 6 |
| P5 | 2 | 8 |



P1 arrives at time 0, P1 executing First , P2 arrives at time 2. Compare P1 remaining time and P2 ( 3-2 = 1) and 6. So, continue P1 after P1, executing P2, at time 4, P3 arrives, compare P2 remaining time (6-1=5) and 4 ( 4<5 ) .So, executing P3 at time 6, P4 arrives. Compare P3 remaining time and P4 ( 4- 2=2 ) and 5 (2<5 ). So, continue P3 , after P3, ready queue consisting P5 is the least out of three. So execute P5, next P2, P4.

**FORMULA :** Finish time - Arrival Time Finish Time for P1 => 3-0 = 3 Finish Time for P2 => 15-2 = 13 Finish Time for P3 => 8-4 =4

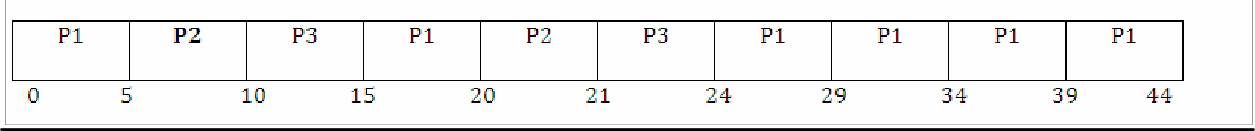
Finish Time for P4 => 20-6 = 14 Finish Time for P5 => 10-8 = 2

Average Turn around time => 36/5 = 7.2 ms.

**4 )ROUND ROBIN SCHEDULING ALGORITHM :**

It is designed especially for time sharing systems. Here CPU switches between the processes. When the time quantum expired, the CPU switched to another job. A small unit of time, called a time quantum or time slice. A time quantum is generally from 10 to 100 ms. The time quantum is generally depending on OS. Here ready queue is a circular queue. CPU scheduler picks the first process from ready queue, sets timer to interrupt after one time quantum and dispatches the process.

|  |  |
| --- | --- |
| **PROCESS** | **BURST TIME** |
| P1 | 30 |
| P2 | 6 |
| P3 | 8 |



**AVERAGE WAITING TIME :**

Waiting time for P1 => 0+(15-5)+(24-20) => 0+10+4 = 14

Waiting time for P2 => 5+(20-10) => 5+10 = 15

Waiting time for P3 => 10+(21-15) => 10+6 = 16 Average waiting time => (14+15+16)/3 = 15 ms.

**AVERAGE TURN AROUND TIME :**

**FORMULA :** Turn around time = waiting time + burst Time Turn around time for P1 => 14+30 =44

Turn around time for P2 => 15+6 = 21 Turn around time for P3 => 16+8 = 24

Average turn around time => ( 44+21+24 )/3 = 29.66 ms

1. **PRIORITY SCHEDULING :**

|  |  |  |
| --- | --- | --- |
| **PROCESS** | **BURST**  **TIME** | **PRIORITY** |
| P1 | 6 | 2 |
| P2 | 12 | 4 |
| P3 | 1 | 5 |
| P4 | 3 | 1 |
| P5 | 4 | 3 |

P4 has the highest priority. Allocate the CPU to process P4 first next P1, P5, P2, P3.



**AVERAGE WAITING TIME :**

Waiting time for P1 => 3-0 =3 Waiting time for P2 => 13-0 = 13 Waiting time for P3 => 25-0 = 25 Waiting time for P4 => 0

Waiting time for P5 => 9-0 =9

Average waiting time => ( 3+13+25+0+9 )/5 = 10 ms

**AVERAGE TURN AROUND TIME :**

Turn around time for P1 =>3+6 = 9 Turn around time for P2 => 13+12= 25 Turn around time for P3 => 25+1 = 26 Turn around time for P4 => 0+3= 3 Turn around time for P5 => 9+4 = 13

Average Turn around time => ( 9+25+26+3+13 )/5 = 15.2 ms

## Disadvantage: Starvation

**Starvation** means only high priority process are executing, but low priority process are waiting for the CPU for the longest period of the time.

## Multiple – processor scheduling:

When multiple processes are available, then the scheduling gets more complicated, because there is more than one CPU which must be kept busy and in effective use at all times.

**Load sharing** resolves around balancing the load between multiple processors. Multi processor systems may be heterogeneous (It contains different kinds of CPU’s) ( or ) Homogeneous(all the same kind of CPU).

## 

## Unit-III

## Concurrency & Deadlocks: Process Synchronization, The Critical- Section Problem, Synchronization Hardware, Semaphores, Classic Problems of Synchronization, Monitors, Synchronization examples. Principles of deadlock – System Model, Deadlock Characterization, Deadlock Prevention, Detection and Avoidance, Recovery form Deadlock.

**Process synchronization** refers to the idea that multiple processes are to join up or [handshake](https://en.wikipedia.org/wiki/Handshaking) at a certain point, in order to reach an agreement or commit to a certain sequence of action. Coordination of simultaneous processes to complete a task is known as process synchronization.

## The critical section problem

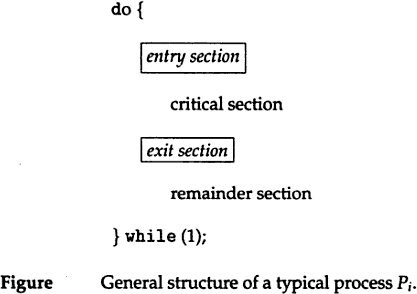
Consider a system , assume that it consisting of n processes. Each process having a segment of code. This segment of code is said to be critical section.

E.G: Railway Reservation System.

Two persons from different stations want to reserve their tickets, the train number, destination is common, the two persons try to get the reservation at the same time. Unfortunately, the available berths are only one; both are trying for that berth.

It is also called the critical section problem. Solution is when one process is executing in its critical section, no other process is to be allowed to execute in its critical section.

The critical section problem is to design a protocol that the processes can use to cooperate. Each process must request permission to enter its critical section. The section of code implementing this request is the **entry section**. The critical section may be followed by an **exit section**. The remaining code is the **remainder section.**



## A solution to the critical section problem must satisfy the following 3 requirements: 1.mutual exclusion:

Only one process can execute their critical section at any time.

## Progress:

When no process is executing a critical section for a data, one of the processes wishing to enter a critical section for data will be granted entry.

## Bounded wait:

No process should wait for a resource for infinite amount of time.

## Critical section:

The portion in any program that accesses a shared resource is called as critical section (or) critical region.

## Peterson’s solution:

Peterson solution is one of the solutions to critical section problem involving two processes. This solution states that when one process is executing its critical section then the other process executes the rest of the code and vice versa.

Peterson solution requires two shared data items:

1. **turn**: indicates whose turn it is to enter into the critical section. If turn == i ,then process i is allowed into their critical section.
2. **flag:** indicates when a process wants to enter into critical section. When process i wants to enter their critical section,it sets flag[i] to true.

**do {**flag[i] = TRUE; turn = j; **while (flag[j] && turn == j); critical section**

## flag[i] = FALSE; remainder section

**} while (TRUE);**

## Synchronization hardware

In a uniprocessor multiprogrammed system, mutual exclusion can be obtained by disabling the interrupts before the process enters its critical section and enabling them after it has exited the critical section**.**

### Disable interrupts Critical section Enable interrupts

Once a process is in critical section it cannot be interrupted. This solution cannot be used in multiprocessor environment. since processes run independently on different processors.

In multiprocessor systems, **Testandset** instruction is provided,it completes execution without interruption. Each process when entering their critical section must set **lock**,to prevent other processes from entering their critical sections simultaneously and must release the lock when exiting their critical sections.

## do { acquire lock

## critical section

## release lock

## remainder section

**} while (TRUE);**

A process wants to enter critical section and value of lock is false then **testandset** returns false and the value of lock becomes true. thus for other processes wanting to enter their critical sections **testandset** returns true and the processes do busy waiting until the process exits critical section and sets the value of lock to false.

## Definition:

boolean TestAndSet(boolean&lock)

{

boolean temp=lock;

Lock=true; return temp;

}

## Algorithm for TestAndSet

do{

while testandset(&lock)

//do nothing

//critical section lock=false

remainder section

}while(TRUE);

## Swap instruction can also be used for mutual exclusion Definition

Void swap(boolean &a, boolean &b)

{

boolean temp=a; a=b;

b=temp;

}

## Algorithm

do

{

key=true;

while(key=true)

swap(lock,key);

critical section

lock=false;

remainder section

}while(1);

lock is global variable initialized to false.

each process has a local variable key.

A process wants to enter critical section,

since the value of lock is false and key is true.

## lock=false key=true

after swap instruction,

## lock=true key=false

now key=false becomes true,process exits repeat-until,and enter into critical section. When process is in critical section (lock=true),so other processes wanting to enter critical section will have

## lock=true key=true

Hence they will do busy waiting in repeat-until loop until the process exits critical section and sets the value of lock to false.

## Semaphores

A semaphore is an integer variable.semaphore accesses only through two operations.

1. **wait:** wait operation decrements the count by1.

If the result value is negative,the process executing the wait operation is blocked.

## signaloperation:

Signal operation increments by 1,if the value is not positive then one of the process blocked in wait operation unblocked.

wait (S) {

while S <= 0 ; // no-op

S--;

}

signal (S)

{

S++;

}

In binary semaphore count can be 0 or 1. The value of semaphore is initialized to 1.

do {

wait (mutex);

// Critical Section signal (mutex);

// remainder section

} while (TRUE);

First process that executes wait operation will be immediately granted sem.count to 0. If some other process wants critical section and executes wait() then it is blocked,since value becomes -1. If the process exits critical section it executes signal().sem.count is incremented by 1.blocked process is removed from queue and added to ready queue.

## Problems:

1. **Deadlock**

Deadlock occurs when multiple processes are blocked.each waiting for a resource that can only be freed by one of the other blocked processes.

## Starvation

one or more processes gets blocked forever and never get a chance to take their turn in the critical section.

## Priority inversion

If low priority process is running ,medium priority processes are waiting for low priority process,high priority processes are waiting for medium priority processes.this is called Priority inversion.

The two most common kinds of semaphores are **counting semaphores** and **binary semaphores**. Counting semaphores represent multiple resources, while binary semaphores, as the name implies, represents two possible states (generally 0 or 1; locked or unlocked).

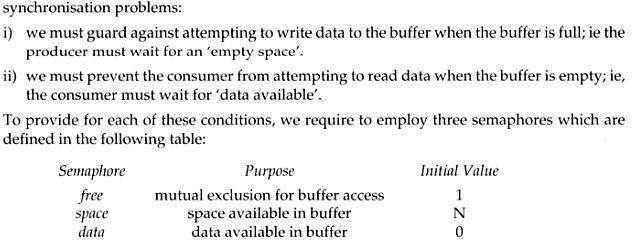
## Classic problems of synchronization

1. **Bounded-buffer problem**

Two processes share a common ,fixed –size buffer.

Producer puts information into the buffer, consumer takes it out.

The problem arise when the producer wants to put a new item in the buffer,but it is already full. The solution is for the producer has to wait until the consumer has consumed atleast one buffer. similarly if the consumer wants to remove an item from the buffer and sees that the buffer is empty,it goes to sleep until the producer puts something in the buffer and wakes it up.



## The structure of the producer process

do {

// produce an item in nextp wait (empty);

wait (mutex);

// add the item to the buffer signal (mutex); signal (full);

} while (TRUE);

## The structure of the consumer process

do { wait (full); wait (mutex);

// remove an item from buffer to nextc signal (mutex);

signal (empty);

// consume the item in nextc

} while (TRUE);

## The readers-writers problem

A database is to be shared among several concurrent processes.some processes may want only to read the database,some may want to update the database. If two readers access the shared data simultaneously no problem.if a write,some other process access the database simultaneously problem arised.Writes have excusive access to

the shared database while writing to the database.This problem is known as readers- writes problem.

## First readers-writers problem

No reader be kept waiting unless a writer has already obtained permission to use the shared resource.

## Second readers-writes problem:

Once writer is ready,that writer performs its write as soon as possible.

A process wishing to modify the shared data must request the lock in write mode. multiple processes are permitted to concurrently acquire a reader-writer lock in read mode. A reader writer lock in read mode. but only one process may acquire the lock for writing as exclusive access is required for writers.

Semaphore mutex initialized to 1

* Semaphore wrt initialized to 1
* Integer read count initialized to 0

## The structure of a writer process

do {

wait (wrt) ;

// writing is performed signal (wrt) ;

} while (TRUE);

## The structure of a reader process

do {

wait (mutex) ; readcount ++ ;

if (readcount == 1) wait (wrt) ;

signal (mutex)

// reading is performed wait (mutex) ; readcount

- - ;

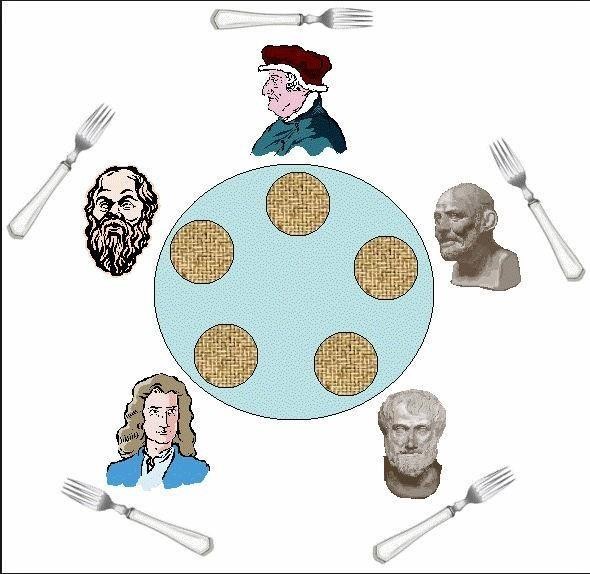
if (readcount == 0) signal (wrt) ; signal (mutex) ;

} while (TRUE);

## Dining Philosophers problem

Five philosophers are seated on 5 chairs across a table. Each philosopher has a plate full of noodles. Each philosopher needs a pair of forks to eat it. There are only 5 forks available all together. There is only one fork between any two plates of noodles.

In order to eat, a philosopher lifts two forks, one to his left and the other to his right. if he is successful in obtaining two forks, he starts eating after some time, he stops eating and keeps both the forks down.



### What if all the 5 philosophers decide to eat at the same time ?

All the 5 philosophers would attempt to pick up two forks at the same time. So,none of them succeed.

One simple solution is to represent each fork with a semaphore.a philosopher tries to grab a fork by executing wait() operation on that semaphore.he releases his forks by executing the signal() operation.This solution guarantees that no two neighbours are eating simultaneously.

Suppose all 5 philosophers become hungry simultaneously and each grabs his left fork,he will be delayed forever.

## The structure of Philosopher i:

do{

wait ( chopstick[i] );

wait ( chopStick[ (i + 1) % 5] );

// eat

signal ( chopstick[i] );

signal (chopstick[ (i + 1) % 5] );

// think

} while (TRUE);

## Several remedies:

1. Allow at most 4 philosophers to be sitting simultaneously at the table.
2. Allow a philosopher to pickup his fork only if both forks are available.
3. An odd philosopher picks up first his left fork and then right fork. an even philosopher picks up his right fork and then his left fork.

## MONITORS

The disadvantage of semaphore is that it is unstructured construct. Wait and signal operations can be scattered in a program and hence debugging becomes difficult.

A monitor is an object that contains both the data and procedures needed to perform allocation of a shared resource. To accomplish resource allocation using monitors, a process must call a **monitor entry routine**. Many processes may want to enter the monitor at the same time. but only one process at a time is allowed to enter. Data inside a monitor may be either global to all routines within the monitor (or) local to a specific routine. Monitor data is accessible only within the monitor. There is no way for processes outside the monitor to access monitor data. This is a form of information hiding.

If a process calls a monitor entry routine while no other processes are executing inside the monitor, the process acquires a lock on the monitor and enters it. while a process is in the monitor, other processes may not enter the monitor to acquire the resource. If a process calls a monitor entry routine while the other monitor is locked the monitor makes the calling process wait outside the monitor until the lock on the monitor is released. The process that has the resource will call a monitor entry routine to release the resource. This routine could free the resource and wait for another requesting process to arrive monitor entry routine calls signal to allow one of the waiting processes to enter the monitor and acquire the resource. Monitor gives high priority to waiting processes than to newly arriving ones.

## Structure:

monitor monitor-name

{

// shared variable declarations procedure P1 (…) { …. } procedurePn (…) {……} Initialization code (…) { … }

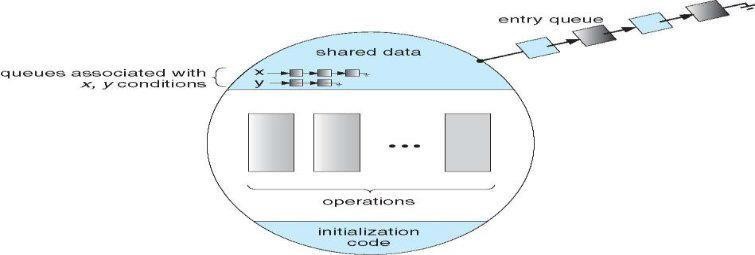
}

}

Processes can call procedures p1,p2,p3……They cannot access the local variables of the monitor

## Schematic view of a Monitor

**Monitor with Condition Variables**



Monitor provides condition variables along with two operations on them i.e. wait and signal.

### wait(condition variable) signal(condition variable)

Every condition variable has an associated queue.A process calling wait on a particular condition variable is placed into the queue associated with that condition variable.A process calling signal on a particular condition variable causes a process waiting on that condition variable to be removed from the queue associated with it.

**Solution to Producer consumer problem using monitors:**

monitor producerconsumer condition full,empty;

## int count;

procedure insert(item)

## {

if(count==MAX) wait(full) ;

insert\_item(item);

count=count+1;

if(count==1)

signal(empty);

## }

procedure remove()

## {

if(count==0)

wait(empty);

remove\_item(item);

count=count-1;

if(count==MAX-1)

signal(full);

## }

procedure producer()

## {

producerconsumer.insert(item);

## }

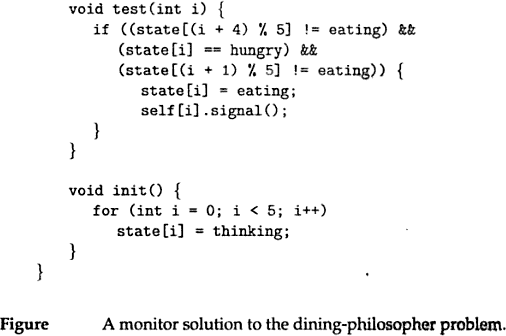
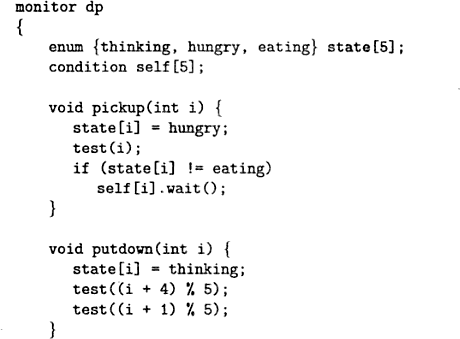
procedure consumer()

## {

producerconsumer.remove();

## }

**Solution to dining philosophers problem using monitors**



A philosopher may pickup his forks only if both of them are available.A philosopher can eat only if his two neighbours are not eating.some other philosopher can delay himself when he is hungry.

**Diningphilosophers.Take\_forks( ) :**

acquires forks ,which may block the process.

## Eat noodles ( )

**Diningphilosophers.put\_forks( ):**

releases the forks.

## Resuming processes within a monitor

If several processes are suspended on condion x and x.signal( ) is executed by some process. then

**how do we determine which of the suspended processes should be resumed next ?** solution is FCFS(process that has been waiting the longest is resumed first).In many circumstances, such simple technique is not adequate. alternate solution is to assign priorities and wake up the process with the highest priority.

## Resource allocation using monitor:

## boolean inuse=false;

## condition available;

//conditionvariable

## monitorentry void get resource()

{

## if(inuse) //is resource inuse

{

## wait(available);//wait until available issignaled

}

## inuse=true; //indicate resource is now inuse

}

## monitor entry void return resource()

{

## inuse=false; //indicate resource is not in use signal(available); //signal a waiting process to proceed

}

## DEADLOCKS

**System model:**

A system consists of a finite number of resources to be distributed among a number of competing processes. The resources are partitioned into several types, each consisting of some number of identical instances. Memory space, CPU cycles, files, I/O devices are examples of resource types. If a system has 2 CPUs, then the resource type CPU has 2 instances.

A process must request a resource before using it and must release the resource after using it. A process may request as many resources as it requires to carry out its task. The number of resources as it requires to carry out its task. The number of resources requested may not exceed the total number of resources available in the system. A process cannot request 3 printers if the system has only two.

A process may utilize a resource in the following sequence:

* 1. **REQUEST**: The process requests the resource. If the request cannot be granted immediately (if the resource is being used by another process), then the requesting process must wait until it can acquire the resource.
  2. **USE:** The process can operate on the resource .if the resource is a printer, the process can print on the printer.
  3. **RELEASE:** The process release the resource.

For each use of a kernel managed by a process the operating system checks that the process has requested and has been allocated the resource. A system table records whether each resource is free (or) allocated. For each resource that is allocated, the table also records the process to which it is allocated. If a process requests a resource that is currently allocated to another process, it can be added to a queue of processes waiting for this resource.

To illustrate a deadlocked state, consider a system with 3 CDRW drives. Each of 3 processes holds one of these CDRW drives. If each process now requests another drive, the 3 processes will be in a deadlocked state. Each is waiting for the event “CDRW is released” which can be caused only by one of the other waiting processes. This example illustrates a deadlock involving the same resource type.

Deadlocks may also involve different resource types. Consider a system with one printer and one DVD drive. The process Pi is holding the DVD and process Pj is holding the printer. If Pi requests the printer and Pj requests the DVD drive, a deadlock occurs.

## DEADLOCK CHARACTERIZATION:

In a deadlock, processes never finish executing, and system resources are tied up, preventing other jobs from starting.

## NECESSARY CONDITIONS:

A deadlock situation can arise if the following 4 conditions hold simultaneously in a system:

1. **MUTUAL EXCLUSION:** Only one process at a time can use the resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
2. **HOLD AND WAIT:** A process must be holding at least one resource and waiting to acquire additional resources that are currently being held by other processes.
3. **NO PREEMPTION:** Resources cannot be preempted. A resource can be released only voluntarily by the process holding it, after that process has completed its task.
4. **CIRCULAR WAIT:** A set {P0,P1,…..Pn} of waiting processes must exist such that P0 is waiting for resource held by P1, P1 is waiting for a resource held by P2,……,Pn-1 is waiting for a resource held by Pn and Pn is waiting for a resource held byP0.

## RESOURCE ALLOCATION GRAPH

Deadlocks can be described more precisely in terms of a directed graph called a system resource allocation graph. This graph consists of a set of vertices V and a set of edges E. the set of vertices V is partitioned into 2 different types of nodes:

P = {P1, P2….Pn}, the set consisting of all the active processes in the system. R= {R1, R2….Rm}, the set consisting of all resource types in the system.

A directed edge from process Pi to resource type Rj is denoted by Pi ->Rj. It signifies that process Pi has requested an instance of resource type Rj and is currently waiting for that resource.

A directed edge from resource type Rj to process Pi is denoted by Rj ->Pi, it signifies that an instance of resource type Rj has been allocated to process Pi.

A directed edge Pi ->Rj is called a requested edge. A directed edge Rj->Piis called an assignment edge.

We represent each process Pi as a circle, each resource type Rj as a rectangle. Since resource type Rj may have more than one instance. We represent each such instance as a dot within the rectangle. A request edge points to only the rectangle Rj. An assignment edge must also designate one of the dots in the rectangle.

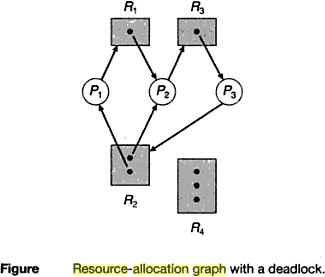
When process Pi requests an instance of resource type Rj, a request edge is inserted in the resource allocation graph. When this request can be fulfilled, the request edge is instantaneously transformed to an assignment edge. When the process no longer needs access to the resource, it releases the resource, as a result, the assignment edge is deleted.

The sets P, R, E:

P= {P1, P2, P3}

R= {R1, R2, R3, R4}

E= {P1 ->R1, P2 ->R3, R1 ->P2, R2 ->P2, R2 ->P1, R3 ->P3}



One instance of resource type R1

Two instances of resource type R2 One instance of resource type R3 Three instances of resource type R4

**PROCESS STATES:**

Process P1 is holding an instance of resource type R2 and is waiting for an instance of resource type R1.

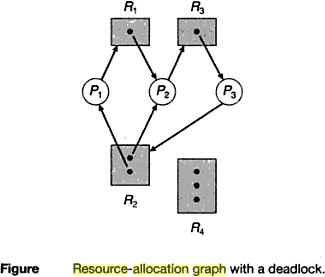
Process P2 is holding an instance of R1 and an instance of R2 and is waiting for instance of R3. Process P3 is holding an instance of R3.

If the graph contains no cycles, then no process in the system is deadlocked. If the graph does contain a cycle, then a deadlock may exist.

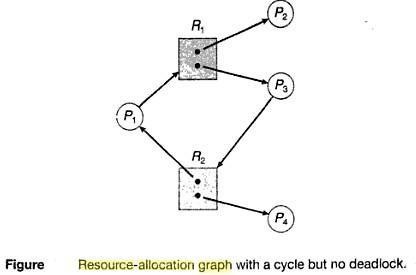
Suppose that process P3 requests an instance of resource type R2. Since no resource instance is currently available, a request edge P3 ->R2 is added to the graph.

2 cycles:

P1 ->R1 ->P2 ->R3 ->P3 ->R2 ->P1 P2 ->R3 ->P3 ->R2 ->P2



Processes P1, P2, P3 are deadlocked. Process P2 is waiting for the resource R3, which is held by process P3.process P3 is waiting for either process P1 (or) P2 to release resource R2. In addition, process P1 is waiting for process P2 to release resource R1.



We also have a cycle: P1 ->R1 ->P3 ->R2 ->P1

However there is no deadlock. Process P4 may release its instance of resource type R2. That resource can then be allocated to P3, breaking the cycle.

## DEADLOCK PREVENTION

For a deadlock to occur, each of the 4 necessary conditions must held. By ensuring that at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

**Mutual Exclusion** – not required for sharable resources; must hold for non sharable resources

**Hold and Wait** – must guarantee that whenever a process requests a resource, it does not hold any other resources

* Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none
* Low resource utilization; starvation possible

## No Preemption –

* If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held are released
* Preempted resources are added to the list of resources for which the process is waiting
* Process will be restarted only when it can regain its old resources, as well as the new ones that it is requesting

**Circular Wait** – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration **Deadlock Avoidance**

Requires that the system has some additional a priori information available

* + - Simplest and most useful model requires that each process declare the maximum number

of resources of each type that it may need

* + - The deadlock-avoidance algorithm dynamically examines the resource- allocation state to ensure that there can never be a circular-wait condition
    - Resource-allocation state is defined by the number of available and allocated resources, and the maximum demands of the processes .

## Safe State

* + - When a process requests an available resource, system must decide if immediate allocation leaves the system in a safe state

System is in **safe state** if there exists a sequence <P1, P2, …, Pn> of ALL the processes in the systems such that for each Pi, the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj, with j <I

That is:

* + If Pi resource needs are not immediately available, then Pi can wait until all

Pj have finished

* + When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate
  + When Pi terminates, Pi +1 can obtain its needed resources, and so on If a system is in safe state no deadlocks

If a system is in unsafe state possibility of deadlock Avoidance ensure that a system will never enter an unsafe state **Avoidance algorithms**

Single instance of a resource type

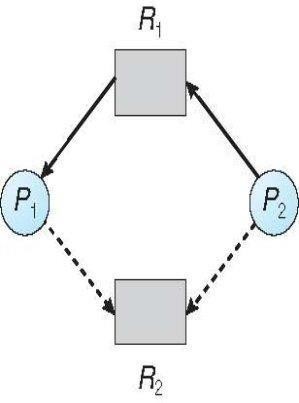
* Use a resource-allocation graph Multiple instances of a resource type
* Use the banker’s algorithm

## Resource-Allocation Graph Scheme

**Claim edge** Pi Æ Rj indicated that process Pj may request resource Rj; represented by a dashed line.

Claim edge converts to request edge when a process requests a resource Request edge converted to an assignment edge when the resource is allocated to the process

When a resource is released by a process, assignment edge reconverts to a claim edge Resources must be claimed a priori in the system



## Unsafe State In Resource-Allocation Graph

**Banker’s Algorithm**

It is applicable for Multiple instances of resource types.

Each process must a priori claim maximum use

When a process requests a resource it may have to wait

When a process gets all its resources it must return them in a finite amount of time Let n = number of processes, and m = number of resources types.

**Available**: Vector of length m. If available [j] = k, there are k instances of resource type

Rjavailable

**Max**: n x m matrix. If Max [i,j] = k, then process Pimay request at most k

instances of resource type Rj

**Allocation**: n x m matrix. If Allocation[i,j] = k then Pi is currently allocated k instances of Rj

**Need**: n x m matrix. If Need[i,j] = k, then Pi may need k more instances of

Rjto complete its task

Need [i,j] = Max[i,j] – Allocation [i,j]

## Safety Algorithm

1. Let Work and Finish be vectors of length m and n, respectively. Initialize: Work = Available

Finish [i] = false fori = 0, 1, …,n- 1

1. Find an isuch that both:
2. Finish [i] = false
3. Needi=Work

If no such iexists, go to step 4

1. Work = Work + Allocationi Finish[i] = true

go to step 2

1. IfFinish [i] == true for all i, then the system is in a safe state

## Resource-Request Algorithm for Process Pi

Request = request vector for process Pi. If Request[i][j] = k then process Pi wants

k instances of resource type Rj

1. If Request[i]<=Need[i]go to step 2. Otherwise, raise error condition, since processhas exceeded its maximum claim
2. If Request[i]<=Available, go to step 3. Otherwise Pi must wait, since resources are not available
3. Pretend to allocate requested resources to Pi by modifying the state as follows:

Available = Available – Request;

Allocation[i]=Allocation[i]+Request[i];

Need[i]=Need[i] – Request[i];

* If safe, the resources are allocated to Pi
* If unsafe ,Pi must wait, and the old resource-allocation state is restored

## Example of Banker’s Algorithm(REFER CLASS NOTES)

consider 5 processes P0 through P4; 3 resource types:

A (10 instances), B (5instances), and C (7 instances)

Snapshot at time T0:

|  |  |  |
| --- | --- | --- |
| Allocation | Max | Available |
| A B C | A B C | A B C |
| P0 0 1 0 | 7 5 3 | 3 3 2 |
| P1 2 0 0 | 3 2 2 |  |
| P2 3 0 2 | 9 0 2 |  |
| P3 2 1 1 | 2 2 2 |  |
| P4 0 0 2 | 4 3 3 |  |

The content of the matrix Need is defined to be Max

– Allocation Need A B C

**Follow Notes to solve this problem**

The system is in a safe state since the sequence <P1, P3, P4, P2, P0> satisfies safety criteria

## P1 Request (1,0,2)

Check that Request £ Available (that is, (1,0,2) (3,3,2)true

|  |  |  |
| --- | --- | --- |
| Allocation | Need | Available |
| A B C | A B C | A B C |
| P0 0 1 0 | 7 4 3 | 2 3 0 |
| P1 3 0 2 | 0 2 0 |  |
| P2 3 0 2 | 6 0 0 |  |
| P3 2 1 1 | 0 1 1 |  |
| P4 0 0 2 | 4 3 1 |  |

Executing safety algorithm shows that sequence <P1, P3, P4, P0, P2> satisfies safety requirement

## Deadlock Detection

Allow system to enter deadlock state Detection algorithm

Recovery scheme

## Single Instance of Each Resource Type

Maintain wait-for graph Nodes are processes P[i] P[j]if P[i]is waiting forP[j]

Periodically invoke an algorithm that searches for a cycle in the graph. If there is a cycle, there exists a deadlock

An algorithm to detect a cycle in a graph requires an order of n2 operations, where n is the number of vertices in the graph

## Resource-Allocation Graph and Wait-for Graph

Resource-Allocation Graph Correspondingwait-for graph

## Several Instances of a Resource Type

**Available**: A vector of length m indicates the number of available resources of each type. **Allocation**: An n x m matrix defines the number of resources of each type currently allocated to each process.

**Request**: An n x m matrix indicates the current request of each process.

If Request [i][j] = k, then process Pi is requesting k more instances of resource type.Rj.

## Detection Algorithm

Let Work and Finish be vectors of length m and n, respectively Initialize:

1. Work = Available
2. For i = 1,2, …, n, if Allocation[i] 0, then Finish[i] = false; otherwise, Finish[i] = true
3. Find an index isuch that both:
4. Finish[i] == false
5. Request[I]<=Work

If no such i exists, go to step 4

1. Work = Work + Allocationi Finish[i] = true

go to step 2

1. If Finish[i] == false, for some i, 1 to n, then the system is in deadlock state. Moreover, if

Finish[i] == false, then Pi is deadlocked

## Recovery from Deadlock:

**Process Termination**

Abort all deadlocked processes

Abort one process at a time until the deadlock cycle is eliminated In which order should we choose to abort?

* Priority of the process
* How long process has computed, and how much longer to completion
* Resources the process has used
* Resources process needs to complete
* How many processes will need to be terminated
* Is process interactive or batch?

## Resource Preemption

Selecting a victim – minimize cost

Rollback – return to some safe state, restart process for that state Starvation – same process may always be picked as victim, include number of rollback in cost factor

**UNIT-IV:**

Memory Management: Swapping, Contiguous Memory Allocation, Paging, structure of the Page Table, Segmentation, Virtual Memory Management: Virtual Memory, Demand Paging, Page-Replacement Algorithms, Thrashing.

## Logical And Physical Addresses

An address generated by the CPU is commonly refereed as **Logical Address**, whereas the address seen by the memory unit that is one loaded into the memory address register of the memory is commonly refereed as the **Physical Address**. The compile time and load time address binding generates the identical **logical and physical addresses**. However, the execution time addresses binding scheme results in differing **logical and physical addresses**.

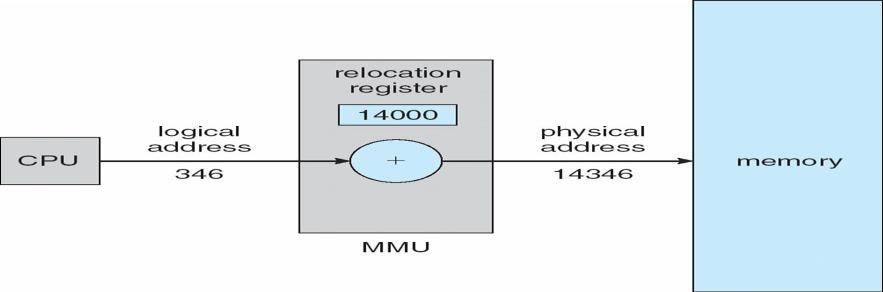
The set of all **logical addresses** generated by a program is known as **Logical Address Space**, where as the set of all **physical addresses** corresponding to these logical addresses is **Physical Address Space**. Now, the run time mapping from virtual address to **physical address** is done by a hardware device known as **Memory Management Unit**. Here in the case of mapping the base register is known as **relocation register**. The value in the relocation register is added to the address generated by a user process at the time it is sent to memory

.Let's understand this situation with the help of example: If the base register contains the value 1000,then an attempt by the user to address location 0 is dynamically relocated to location 1000,an access to location 346 is mapped to location 1346.

### Memory-Management Unit (MMU)

Hardware device that maps virtual to physical address

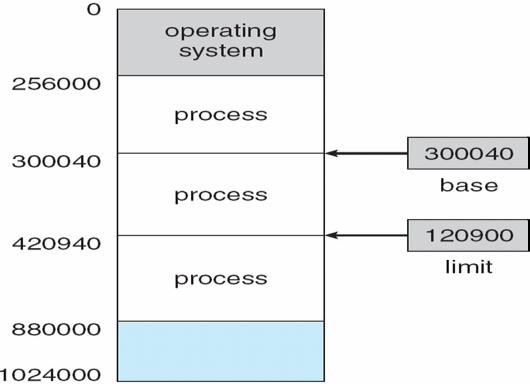
In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory

The user program deals with logical addresses; it never sees the real physical addresses

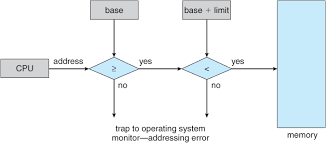
The user program never sees the **real physical address** space, it always deals with the **Logical addresses**. As we have two different type of addresses **Logical address** in the range (0 to max) and **Physical addresses** in the range(R to R+max) where R is the value of relocation register. The user generates only **logical addresses** and thinks that the process runs in location to 0 to max. As it is clear from the above text that user program supplies only logical addresses, these **logical addresses** must be mapped to **physical address** before they are used.

### Base and Limit Registers

A pair of **base** and **limit** registers define the logical address space



HARDWARE PROTECTION WITH BASE AND LIMIT



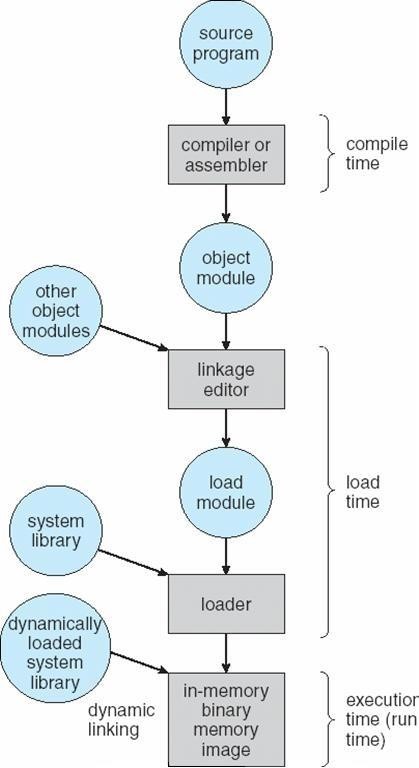
### Binding of Instructions and Data to Memory

Address binding of instructions and data to memory addresses can happen at three different stages

**Compile time**: If memory location known a priori, **absolute code** can be generated; must recompile code if starting location changes

* **Load time**: Must generate **relocatable code** if memory location is not known at compile time

**Execution time**: Binding delayed until run time if the process can be moved during its execution From one memory segment to another. Need hardware support for address maps (e.g., base and limit registers)

**Multistep Processing of a User Program**

## Dynamic Loading

Routine is not loaded until it is called

Better memory-space utilization; unused routine is never loaded

Useful when large amounts of code are needed to handle infrequently occurring cases

No special support from the operating system is required implemented through program design

### Dynamic Linking

Linking postponed until execution time

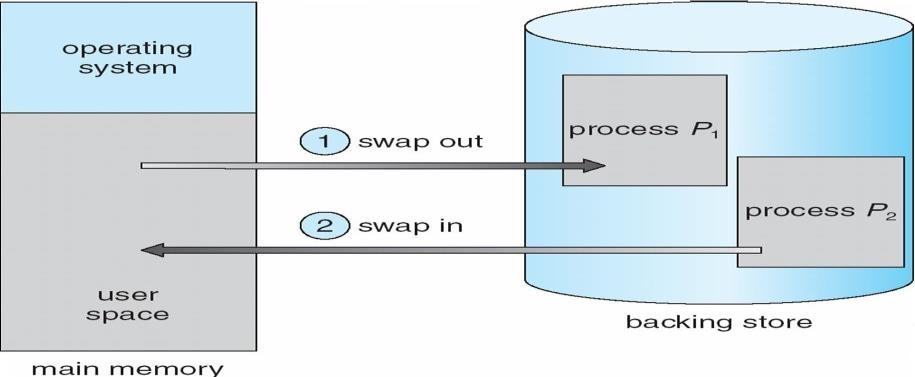
Small piece of code, stub, used to locate the appropriate memory-resident library routine Stub replaces itself with the address of the routine, and executes the routine Operating system needed to check if routine is in processes’ memory address Dynamic linking is particularly useful for libraries

System also known as **shared libraries**

### Swapping

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped and Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

System maintains a **ready queue** of ready-to-run processes which have memory images on disk

**Schematic View of Swapping**

Main memory usually into two partitions:

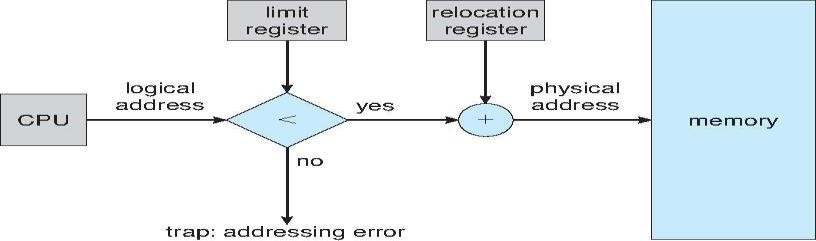
Resident operating system, usually held in low memory with interrupt vector

User processes then held in high memory and Relocation registers used to protect user processes from each other, and from changing operating-system code and data Base register contains value of smallest physical address

 Limit register contains range of logical addresses – each logical address must be less than the limit register

*MMU maps logical address dynamically

### Hardware Support for Relocation and Limit Registers



Multiple-partition allocation

Hole – block of available memory; holes of various size are scattered throughout memory When a process arrives, it is allocated memory from a hole large enough to accommodate it

# There are 2 types of memory allocations:

# 1. Contiguous Allocation-MVT,MFT

# 2. Non-contiguous allocation-Paging and Segmentation.

# Contiguous Allocation-MVT,MFT

**Contiguous memory allocation** is one of the efficient ways of allocating main memory to the processes. The memory is divided into two partitions. One for the Operating System and another for the user processes. Operating System is placed in low or high memory depending on the interrupt vector placed. In contiguous memory allocation each process is contained in a single contiguous section of memory.

Memory protection

Memory protection is required to protect Operating System from the user processes and user processes from one another. A relocation register contains the value of the smallest physical address for example say 100040. The limit register contains the range of logical address for example say 74600. Each logical address must be less than limit register. If a logical address is greater than the limit register, then there is an addressing error and it is trapped. The limit register hence offers memory protection.

The MMU, that is, Memory Management Unit maps the logical address dynamically, that is at run time, by adding the logical address to the value in relocation register. This added value is the physical memory address which is sent to the memory.

The CPU scheduler selects a process for execution and a dispatcher loads the limit and relocation registers with correct values. The advantage of relocation register is that it provides an efficient way to allow the Operating System size to change dynamically.

Memory allocation- [**Refer diagrams for MVT and MFT]**

**There are two methods namely,**

* 1. **multiple partition method-** MVT stands for Multiprogramming with a Variable number of Tasks
  2. **multiple fixed partition method -** MFT stands for Multiprogramming with a Fixed number of Tasks**.**

In multiple partition method, the memory is divided into several fixed size partitions. One process occupies each partition. This scheme is rarely used nowadays. Degree of multiprogramming depends on the number of partitions. Degree of multiprogramming is the number of programs that are in the main memory. The CPU is never left idle in multiprogramming. This was used by IBM OS/360 called MFT. MFT stands for Multiprogramming with a Fixed number of Tasks.

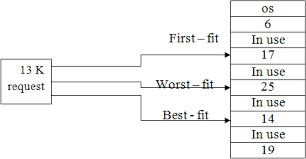
Generalization of fixed partition scheme is used in MVT. MVT stands for Multiprogramming with a Variable number of Tasks. The Operating System keeps track of which parts of memory are available and which is occupied. This is done with the help of a table that is maintained by the Operating System. Initially the whole of the available memory is treated as one large block of memory called a **hole**. The programs that enter a system are maintained in an input queue. From the hole, blocks of main memory are allocated to the programs in the input queue. If the hole is large, then it is split into two, and one half is allocated to the arriving process and the other half is returned. As and when memory is allocated, a set of holes in scattered. If holes are adjacent, they can be merged.

Now there comes a general dynamic storage allocation problem.

The following are the solutions to the **dynamic storage allocation problem**.

* First fit: The first hole that is large enough is allocated. Searching for the holes starts from the beginning of the set of holes or from where the previous first fit search ended.
* Best fit: The smallest hole that is big enough to accommodate the incoming process is allocated. If the available holes are ordered, then the searching can be reduced.
* Worst fit: The largest of the available holes is allocated.

**Example:**



First and best fits decrease time and storage utilization. First fit is generally faster. Fragmentation

The disadvantage of contiguous memory allocation is **fragmentation**. There are two types of fragmentation, namely, internal fragmentation and External fragmentation.

## Internal fragmentation

When memory is free internally, that is inside a process but it cannot be used, we call that fragment as internal fragment. For example say a hole of size 18464 bytes is available. Let the size of the process be 18462. If the hole is allocated to this process, then two bytes are left which is not used. These two bytes which cannot be used forms the internal fragmentation. The worst part of it is that the overhead to maintain these two bytes is more than two bytes.

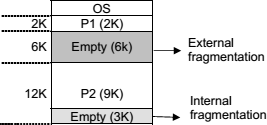
## External fragmentation

All the three dynamic storage allocation methods discussed above suffer external fragmentation. When the total memory space that is got by adding the scattered holes is sufficient to satisfy a request but it is not available contiguously, then this type of fragmentation is called external fragmentation.

The solution to this kind of external fragmentation is compaction. **Compaction** is a method by which all free memory that are scattered are placed together in one large memory block. It is to be noted that compaction cannot be done if relocation is done at compile time or assembly time. It is possible only if dynamic relocation is done, that is relocation at execution time.

One more solution to external fragmentation is to have the logical address space and physical address space to be non contiguous. Paging and Segmentation are popular non contiguous allocation methods.

## Example for internal and external fragmentation



**2. Non-contiguous allocation-Paging and Segmentation**

**Paging**

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.

### Paging Hardware

Address Translation

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

Logical Address = Page number + page offset

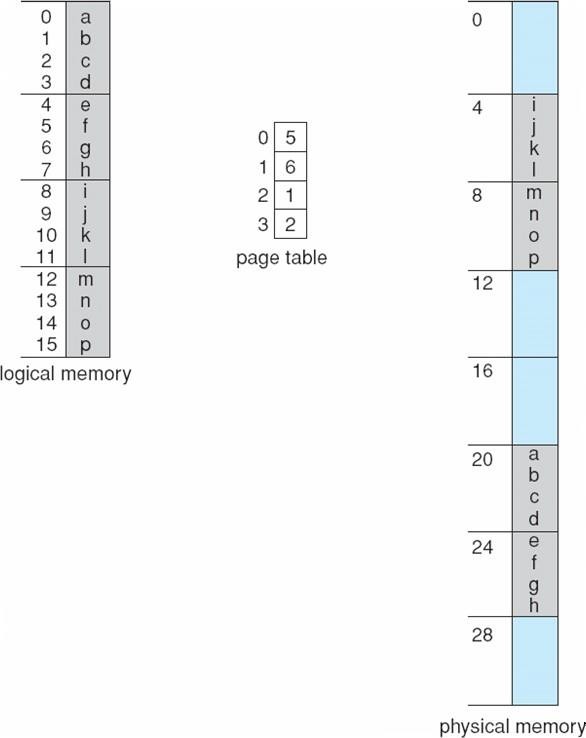
Frame address is called **physical address** and represented by a **frame number** and the **offset**.

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.

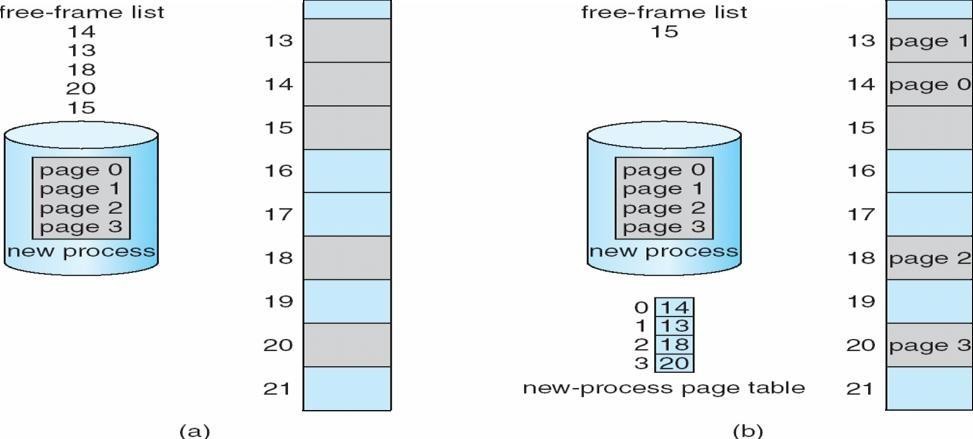
## Paging Model of Logical and Physical Memory

**Paging Example**



## 32-byte memory and 4-byte pages

### Free Frames



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

## Implementation of Page Table

Page table is kept in main memory

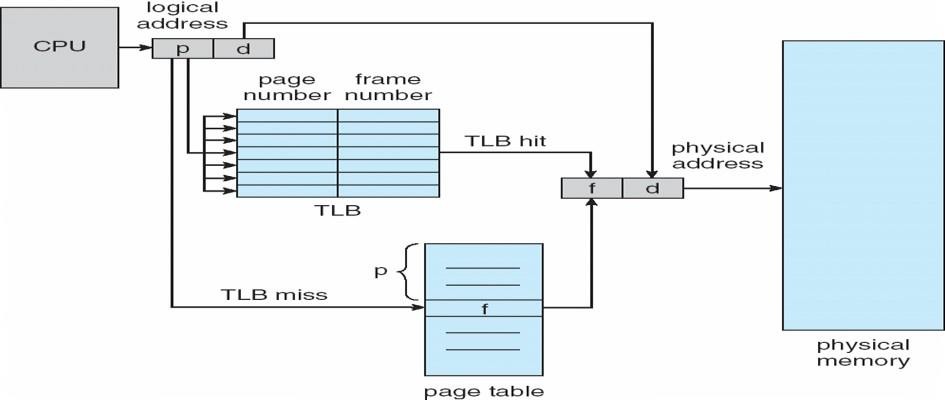
**Page-table base register (PTBR)** points to the page table

**Page-table length register (PRLR)** indicates size of the page table

In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**

### Paging Hardware With TLB



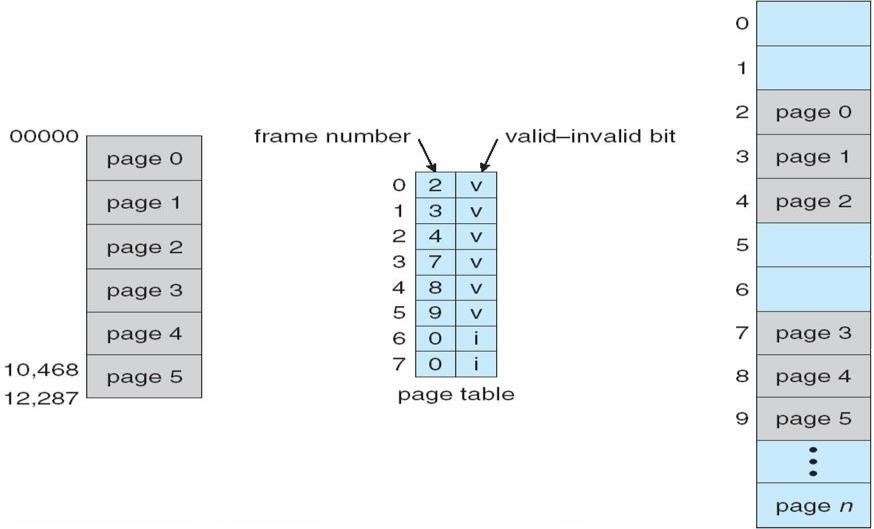
**Memory Protection**

Memory protection implemented by associating protection bit with each frame

**Valid-invalid** bit attached to each entry in the page table:

“valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page “invalid” indicates that the page is not in the process’ logical address space

Valid (v) or Invalid (i) Bit In A Page Table



## Shared Pages Shared code

One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).

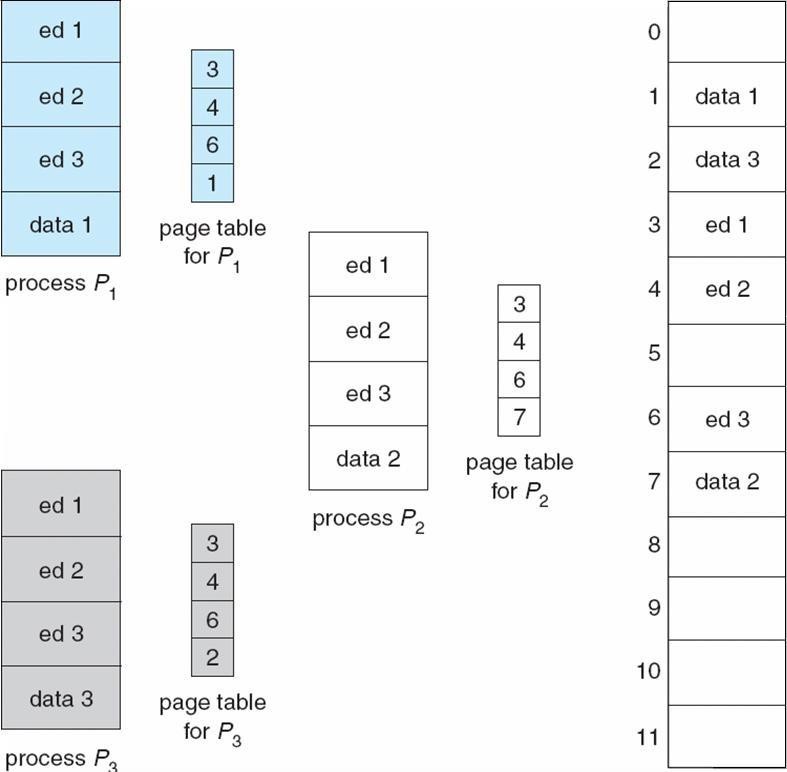
Shared code must appear in same location in the logical address space of all processes

### Private code and data

Each process keeps a separate copy of the code and data

The pages for the private code and data can appear anywhere in the logical address space

**Shared Pages Example**



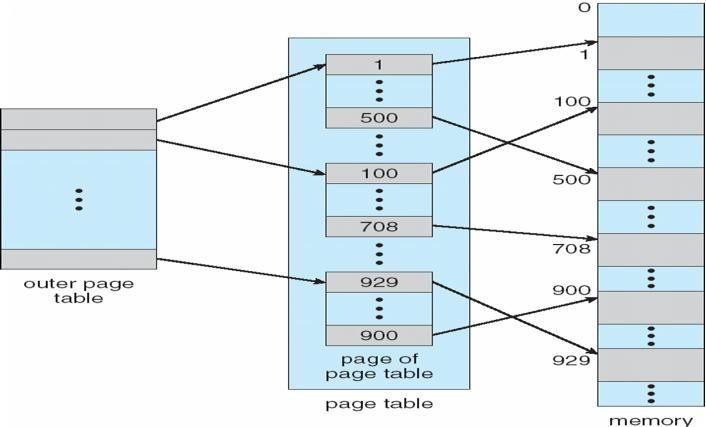
## Structure of the Page Table

Hierarchical Paging

Hashed Page Tables

Inverted Page Tables

### Hierarchical Page Tables

Break up the logical address space into multiple page tables A simple technique is a two-level page table

**Two-Level Page-Table Scheme**

## Two-Level Paging Example

A logical address (on 32-bit machine with 1K page size) is divided into: a page number consisting of 22 bits

a page offset consisting of 10 bits

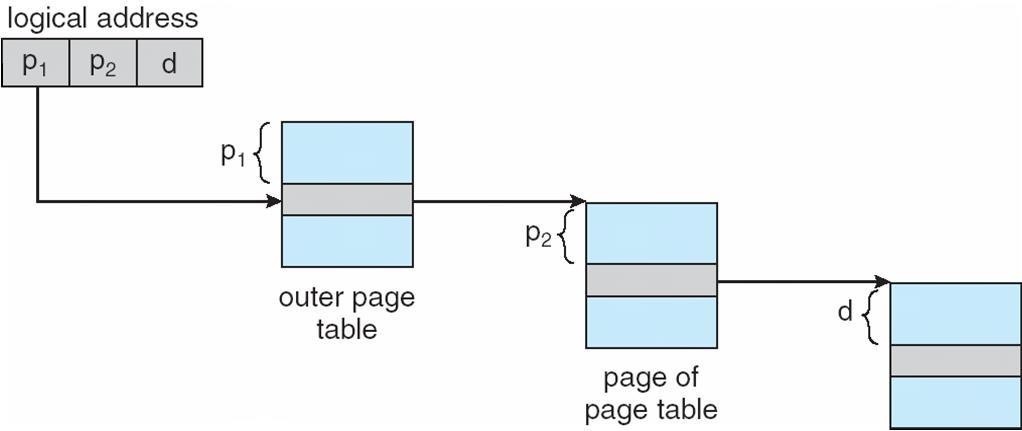
Since the page table is paged, the page number is further divided into:

a 12-bit page number a 10-bit page offset Thus, a logical address is as follows:

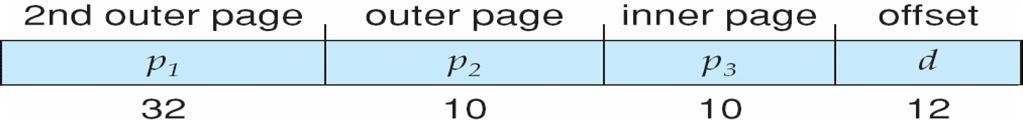
where pi is an index into the outer page table, and p2 is the displacement within the page of the outer page table

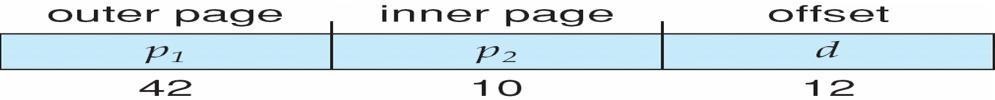
|  |  |  |
| --- | --- | --- |
| Page number | | page offset |
| pi  12 | p2  10 | d10 |

**Address-Translation Scheme**



## Three-level Paging Scheme





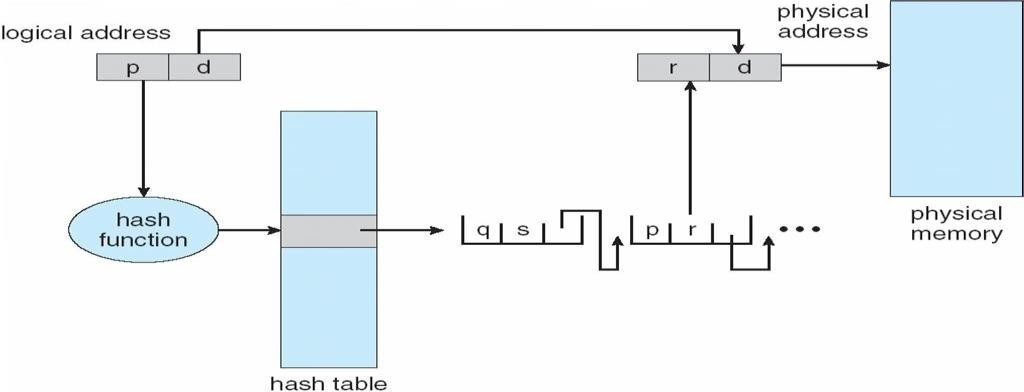
**Hashed Page Tables**

Common in address spaces > 32 bits

The virtual page number is hashed into a page table

This page table contains a chain of elements hashing to the same location Virtual page numbers are compared in this chain searching for a match

If a match is found, the corresponding physical frame is extracted



## Fig:Hashed Page Table

### Inverted Page Table

One entry for each real page of memory

Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page

Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs

Use hash table to limit the search to one — or at most a few — page-table entries

### Inverted Page Table Architecture

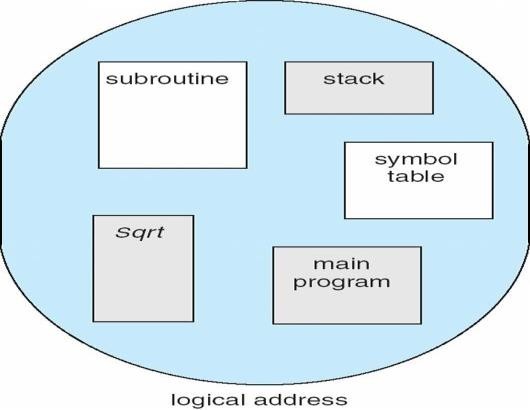
Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging −

* Paging reduces external fragmentation, but still suffers 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.

## Segmentation

*Memory-management scheme that supports user view of memory A program is a collection of segments

* A segment is a logical unit such as:
* main program
* Procedure
* function method
* object
* local variables, global variables
* common block
* stack
* symbol table
* arrays

**User’s View of a Program**

## Segmentation Architecture

Logical address consists of a two tuple: o <segment-number, offset>,

**Segment table** – maps two-dimensional physical address; **base** – contains the starting physical address where the segments reside in memory **limit** – specifies the length of the segment

**Segment-table base register (STBR)** points to the segment table’s location in memory **Segment-table length register (STLR)** indicates number of segments used by a program; segment number **s** is legal if **s** < **STLR**

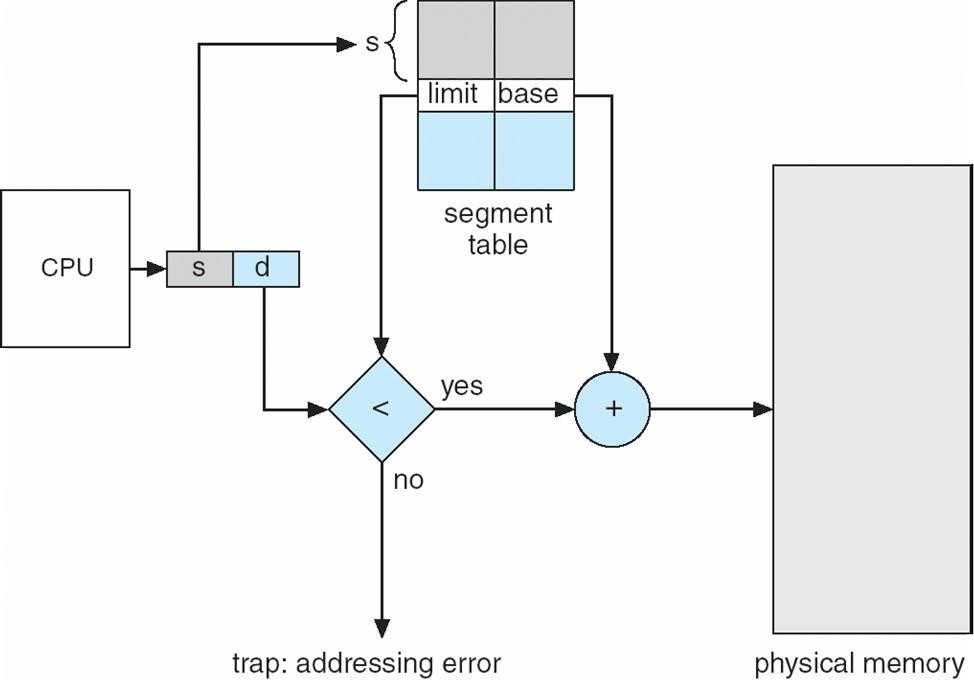
Protection With each entry in segment table associate:

validation bit = 0 Þ illegal segment read/write/execute privileges

Protection bits associated with segments; code sharing occurs at segment level

Since segments vary in length, memory allocation is a dynamic storage-allocation problem A segmentation example is shown in the following diagram

**Segmentation Hardware**



## Example of Segmentation

**Segmentation with paging**

Instead of an actual memory location the segment information includes the address of a [page](https://en.wikipedia.org/wiki/Page_table) [table](https://en.wikipedia.org/wiki/Page_table) for the segment. When a program references a memory location the offset is translated to a memory address using the page table. A segment can be extended simply by allocating another memory page and adding it to the segment's page table.

An implementation of [virtual memory](https://en.wikipedia.org/wiki/Virtual_memory) on a system using segmentation with paging usually only moves individual pages back and forth between main memory and secondary storage, similar to a paged non-segmented system. Pages of the segment can be located anywhere in main memory and need not be contiguous. This usually results in a reduced amount of input/output between primary and secondary storage and reduced memory fragmentation.

## Virtual Memory

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

In real scenarios, most processes never need all their pages at once, for following reasons :

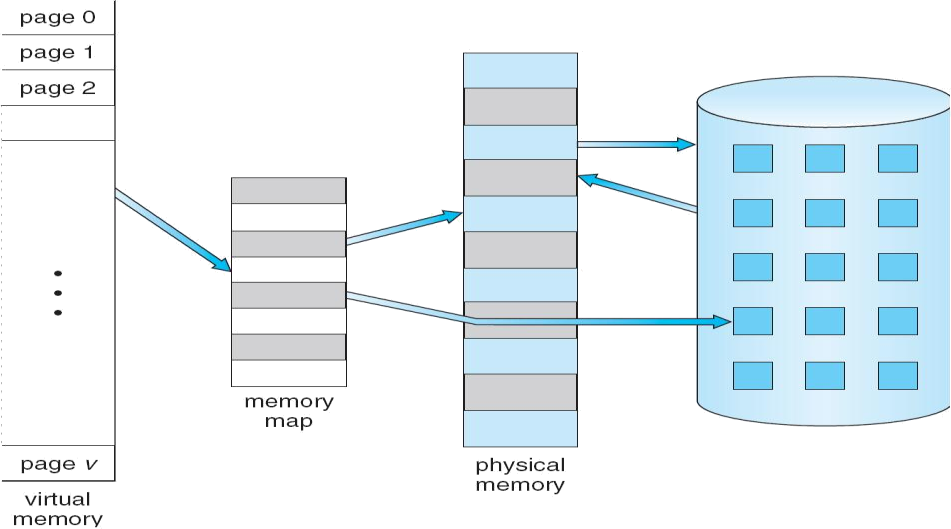
* Error handling code is not needed unless that specific error occurs, some of which are quite rare.
* Arrays are often over-sized for worst-case scenarios, and only a small fraction ofthe arrays are actually used in practice.
* Certain features of certain programs are rarely used.

Fig. Diagram showing virtual memory that is larger than physical memory.

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.

## Benefits of having Virtual Memory :

1. Large programs can be written, as virtual space available is huge compared to physical memory.
2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

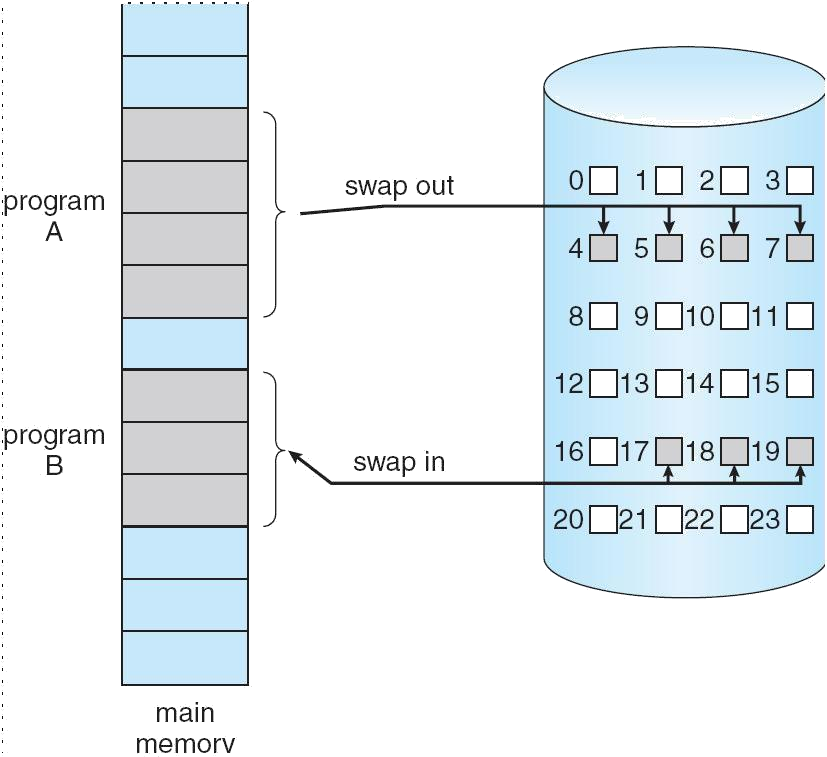
### Demand Paging

A demand paging is similar to a paging system with swapping(Fig 5.2). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory.

When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked checking the bit and marking a page will have no effect if the process never attempts to access the pages. While the process executes and accesses pages that are memory resident, execution proceeds normally.

Fig. Transfer of a paged memory to continuous disk space



Access to a page marked invalid causes a page-fault trap. This trap is the result of the operating system's failure to bring the desired page into memory.

Initially only those pages are loaded which will be required the process immediately.

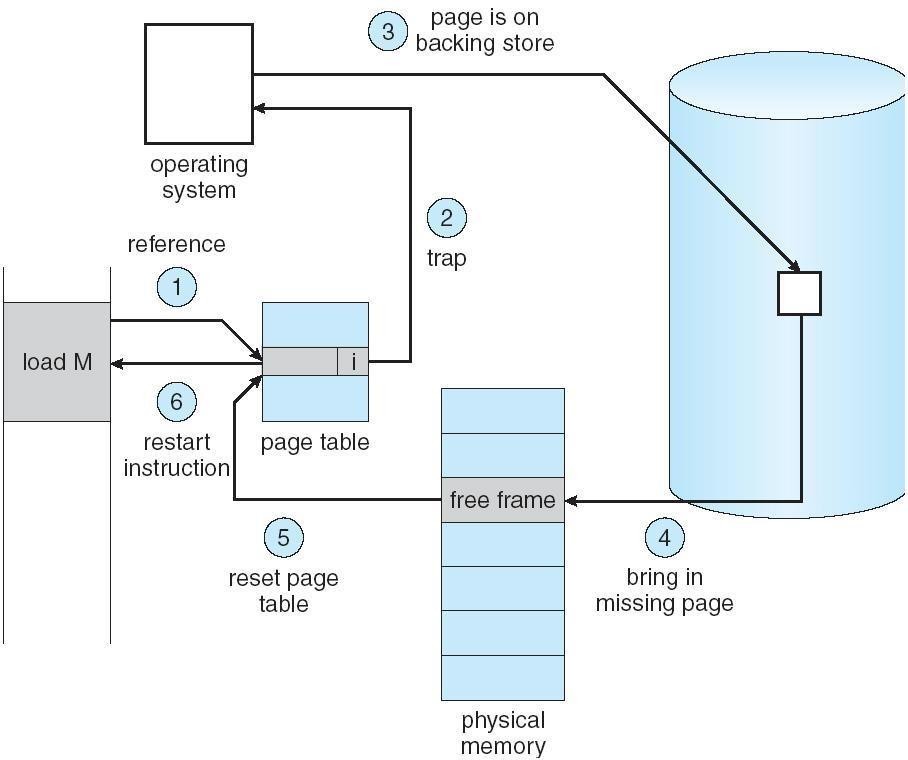
The pages that are not moved into the memory are marked as invalid in the page table. For

an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page.

When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If its found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from disk to the specified memory location. ( This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime. )
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.

Fig. Steps in handling a page fault



1. The instruction that caused the page fault must now be restarted from the beginning. There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Pure Demand Paging.**

The only major issue with Demand Paging is, after a new page is loaded, the process starts execution from the beginning. It is not a big issue for small programs, but for larger programs it affects performance drastically.

## What is dirty bit?

When a bit is modified by the CPU and not written back to the storage, it is called as a dirty bit. This bit is present in the memory cache or the virtual storage space.

### Advantages of Demand Paging:

1. Large virtual memory.
2. More efficient use of memory.
3. Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

### Disadvantages of Demand Paging:

1. Number of tables and amount of processor over head for handling page interrupts are greater than in the case of the simple paged management techniques.
2. due to the lack of an explicit constraints on a jobs address space size.

### Page Replacement

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem :

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called **Page replacement** and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

## Page Replacement Algorithm[REFER TO NOTES]

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.

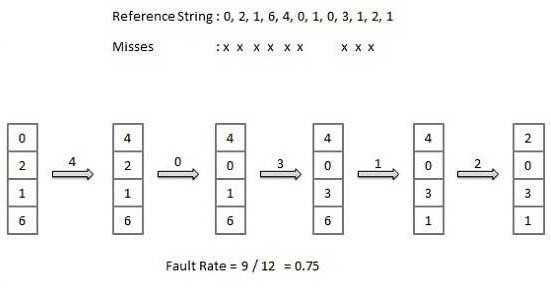
When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

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

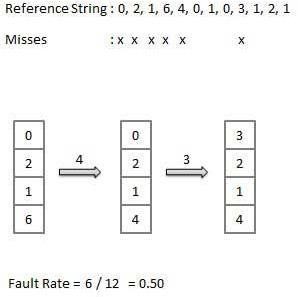
The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference.

The latter choice produces a large number of data, where we note two things.

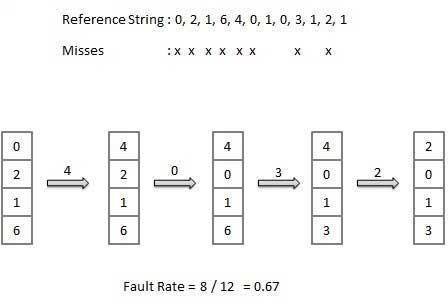
* For a given page size, we need to consider only the page number, not the entire address.
* If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references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

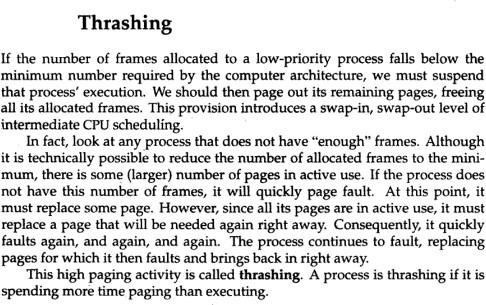
**Optimal Page algorithm**

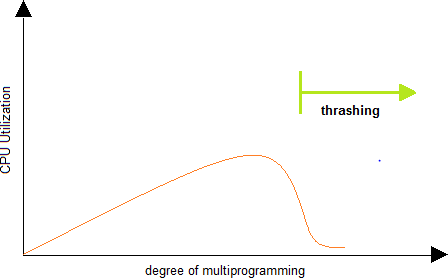
* An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.
* Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.



## 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.





**UNIT-V:**

**File system Interface-** the concept of a file, Access Methods, Directory structure, File system mounting, and file sharing.

**File System implementation**- File system structure, allocation methods, free-space management Mass-storage structure overview of Mass-storage structure, Disk scheduling.

System Protection: Goals of protection, Principles and Domain of protection.

**File System**

**File Concept:**

Computers can store information on various storage media such as, magnetic disks, magnetic tapes, optical disks. The physical storage is converted into a logical storage unit by operating system. The logical storage unit is called FILE. A file is a collection of similar records. A record is a collection of related fields that can be treated as a unit by some application program. A field is some basic element of data. Any individual field contains a single value. A data base is collection of related data.

|  |  |  |  |
| --- | --- | --- | --- |
| Student | Marks | Marks | Fail/Pas |
| KUMA | 85 | 86 | P |
| LAKSH | 93 | 92 | P |

## DATA FILE

Student name, Marks in sub1, sub2, Fail/Pass is fields. The collection of fields is called a RECORD. **RECORD:**

|  |  |  |  |
| --- | --- | --- | --- |
| LAKSH | 93 | 92 | P |

Collection of these records is called a data file.

## FILE ATTRIBUTES :

* 1. Name : A file is named for the convenience of the user and is referred by its name. A name is usually a string of characters.
  2. Identifier : This unique tag, usually a number ,identifies the file within the file system.
  3. Type : Files are of so many types. The type depends on the extension of the file.

Example:

.exe Executable file

.obj Object file

.src Source file

* 1. Location : This information is a pointer to a device and to the location of the file on that device.
  2. Size : The current size of the file (in bytes, words,blocks).
  3. Protection : Access control information determines who can do reading, writing, executing and so on.
  4. Time, Date, User identification : This information may be kept for creation, last modification, last use.

## FILE OPERATIONS

1. Creating a file : Two steps are needed to create a file. They are:
   * Check whether the space is available ornot.
   * If the space is available then made an entry for the new file in the directory. The entry includes name of the file, path of the file,etc…
2. Writing a file : To write a file, we have to know 2 things. One is name of the file and second is the information or data to be written on the file, the system searches the entired given location for the file. If the file is found, the system must keep a write pointer to the location in the file where the next write is to take place.
3. Reading a file : To read a file, first of all we search the directories for the file, if the file is found, the system needs to keep a read pointer to the location in the file where the next read is to take place. Once the read has taken place, the read pointer is updated.
4. Repositioning within a file : The directory is searched for the appropriate entry and the current file position pointer is repositioned to a given value. This operation is also called file seek.
5. Deleting a file : To delete a file, first of all search the directory for named file, then released the file space and erase the directory entry.
6. Truncating a file : To truncate a file, remove the file contents only but, the attributes are as it is.

**FILE TYPES:**The name of the file split into 2 parts. One is name and second is Extension. The file type is depending on extension of the file.

|  |  |  |
| --- | --- | --- |
| File Type | Extension | Purpose |
| Executable | .exe  .com  .bin | Ready to run (or) ready to run machine |
| Source code | .c  .cpp  .asm | Source code invarious languages. |
| Object | .obj  .o | Compiled,machine |
| Batch | .bat  .sh | Commands to the command |

|  |  |  |
| --- | --- | --- |
| Text | .txt  .doc | Textual data, documents |
| Word processor | .doc  .wp  .rtf | Various word processor form ats |
| Library | .lib  .dll | Libraries of routines for |
| Print or View | .pdf  .jpg | Binary file in a format for |
| Archive | .arc  .zip | Related files grouped into a |
| Multimedia | .mpeg  .mp3  .avi | Binary file containing audio  or audio/video |

## FILE STRUCTURE

File types also can be used to indicate the internal structure of the file. The operating system requires that an executable file have a specific structure so that it can determine where in memory to load the file and what the location of the first instruction is. If OS supports multiple file structures, the resulting size of OS is large. If the OS defines 5 different file structures, it needs to contain the code to support these file structures. All OS must support at least one structure that of an executable file so that the system is able to load and run programs.

## INTERNAL FILE STRUCTURE

In UNIX OS, defines all files to be simply stream of bytes. Each byte is individually addressable by its offset from the beginning or end of the file. In this case, the logical record size is 1 byte. The file system automatically packs and unpacks bytes into physical disk blocks, say 512 bytes per block.

The logical record size, physical block size, packing determines how many logical records are in each physical block. The packing can be done by the user’s application program or OS. A file may be considered a sequence of blocks. If each block were 512 bytes, a file of 1949 bytes would be allocated 4 blocks (2048 bytes). The last 99 bytes

would be wasted. It is called internal fragmentation all file systems suffer from internal fragmentation, the larger the block size, the greater the internal fragmentation.

## FILE ACCESS METHODS

Files stores information, this information must be accessed and read into computer memory. There are so many ways that the information in the file can be accessed.

## Sequential file access:

Information in the file is processed in order i.e. one record after the other. Magnetic tapes are supporting this type of file accessing.

Eg : A file consisting of 100 records, the current position of read/write head is 45th record, suppose we want to read the 75th record then, it access sequentially from 45, 46, 47

…….. 74, 75. So the read/write head traverse all the records between 45 to 75.



## Direct access:

Direct access is also called relative access. Here records can read/write randomly without any order. The direct access method is based on a disk model of a file, because disks allow random access to any file block.

Eg : A disk containing of 256 blocks, the position of read/write head is at 95th block. The block is to be read or write is 250th block. Then we can access the 250th block directly without any restrictions.

Eg : CD consists of 10 songs, at present we are listening song 3, If we want to listen song 10, we can shift to 10.

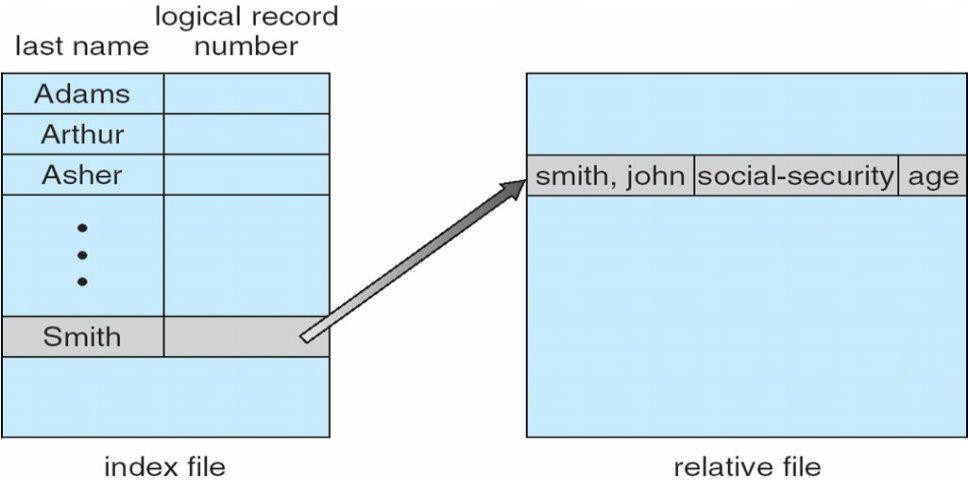
## Indexed Sequential File access

The main disadvantage in the sequential file is, it takes more time to access a Record

.Records are organized in sequence based on a key field. Eg :

A file consisting of 60000 records,the master index divide the total records into 6 blocks, each block consisiting of a pointer to secondary index.The secondary index divide the 10,000 records into 10 indexes.Each index consisting of a pointer to its orginal

location.Each record in the index file consisting of 2 field, A key field and a pointer field.



## DIRECTORY STRUCTURE

Sometimes the file system consisting of millions of files,at that situation it is very hard to manage the files. To manage these files grouped these files and load one group into one partition.

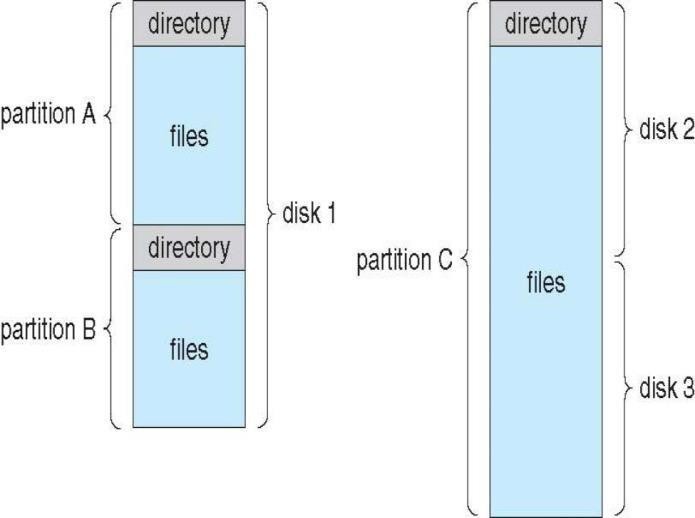
Each partition is called a directory .a directory structure provides a mechanism for organizing many files in the file system.

## OPERATION ON THE DIRECTORIES :

1. Search for a file : Search a directory structure for requiredfile.
2. createafile : New files need to be created, added to thedirectory.
3. Deleteafile : When a file is no longer needed,we want to remove it fromthe

directory.

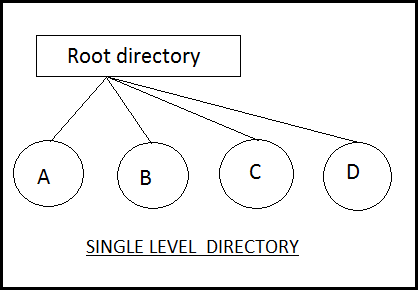
1. List adirectory : We can know the list of files in thedirectory.
2. Renameafile : When ever we need to change the name of the file,wecanchange thename.
3. Traverse the file system : We need to access every directory and every file with in a directory structure we can traverse the file system



The **VARIOUS DIRECTORY STRUCTURES [refer ppt for -advantages and disadvantages]**

## Single level directory:

The directory system having only one directory,it consisting of all files some times it is said to be root directory.



E.g :- Here directory containing 4 files (A,B.C,D).the advantage of the scheme is its simplicity and the ability to locate files quickly.The problem is different users may accidentally use the same names for their files.

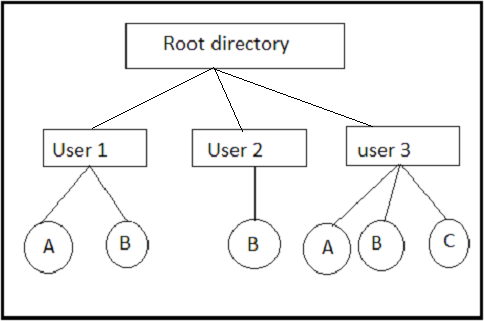
E.g :- If user 1 creates a files caled sample and then later user 2 to creates a file called sample,then user2’s file will overwrite user 1 file.Thats why it is not used in the multi user system.

## Two level directory:

The problem in single level directory is different user may be accidentally use

the same name for their files. To avoid this problem each user need a private directory,

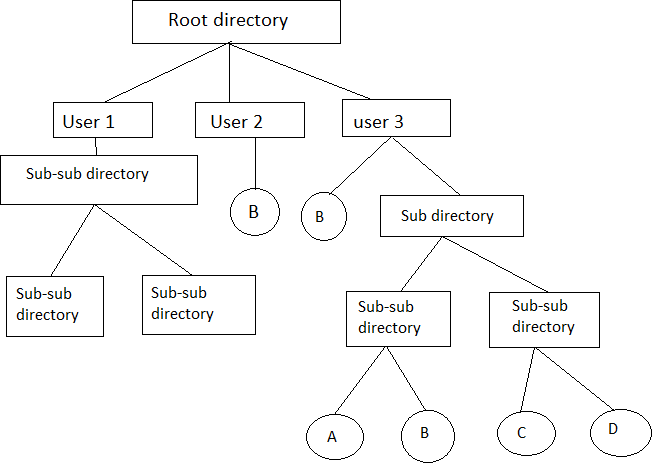
Names chosen by one user don't interfere with names chosen by a different user.



Root directory is the first level directory.user 1,user2,user3 are user level of directory A,B,C are files.

## Tree structured directory:

Two level directory eliminates name conflicts among users but it is not satisfactory for users with a large number of files.To avoid this create the sub- directory and load the same type of files into the sub-directory.so, here each can have as many directories are needed.



There are 2 types of path

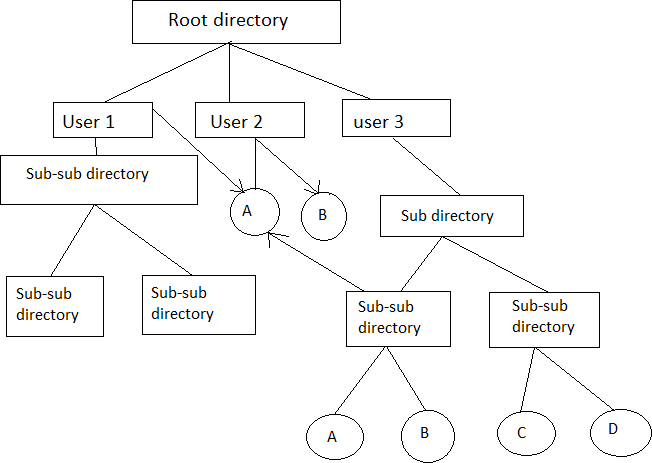
1. Absoulte path
2. Relative path

Absoulte path : Begging with root and follows a path down to specified files giving directory, directory name on the path.

Relative path : A path from current directory.

## Acyclic graphdirectory

Multiple users are working on a project, the project files can be stored in a comman sub-directory of the multiple users. This type of directory is called acyclic graph directory .The common directory will be declared a shared directory. The graph contain no cycles with shared files, changes made by one user are made visible to other users.A file may now have multiple absolute paths. when shared directory/file is deleted, all pointers to the directory/ files also to be removed.



Advantages :- Traversing is easy. Easy sharing is possible.

## File system structure:

Disk provides the bulk of secondary storage on which a file system is maintained. They have 2 characteristics that make them a convenient medium for storing multiple files.

1. A disk can be rewritten in place. It is possible to read a block from the disk, modify the block, and write it back into same place.
2. A disk can access directly any block of information it contains.

Devices

I/O Control

Basic File System

File Organisation Module

Logical File System

Application Programs

I/O Control: consists of device drivers and interrupt handlers to transfer information between the main memory and the disk system. The device driver writes specific bit patterns to special locations in the I/O controller’s memory to tell the controller which device location to act on and what actions to take.

The Basic File System needs only to issue commands to the appropriate device driver to read and write physical blocks on the disk. Each physical block is identified by its numeric disk address (Eg. Drive 1, cylinder 73, track2, sector 10).

The File Organization Module knows about files and their logical blocks and physical blocks. By knowing the type of file allocation used and the location of the file, file organization module can translate logical block address to physical addresses for the basic file system to transfer. Each file’s logical blocks are numbered from 0 to n. so, physical blocks containing the data usually do not match the logical numbers. A translation is needed to locate each block.

The Logical File System manages all file system structure except the actual data (contents of file). It maintains file structure via file control blocks. A file control block (inode in Unix file systems) contains information about the file, ownership, permissions, location of the file contents.

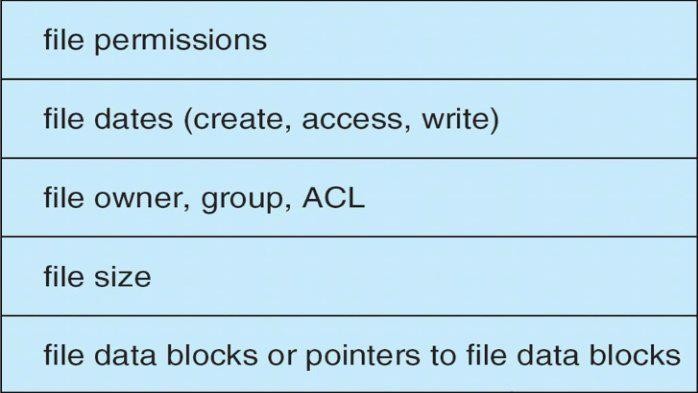
## File System Implementation:

Overview:

A Boot Control Block (per volume) can contain information needed by the system to boot an OS from that volume. If the disk does not contain an OS, this block can be empty.

A Volume Control Block (per volume) contains volume (or partition) details, such as number of blocks in the partition, size of the blocks, a free block, count and free block pointers, free FCB count, FCB pointers.

## A Typical File Control Block



A Directory Structure (per file system) is used to organize the files. A PER-FILE FCB contains many details about the file.

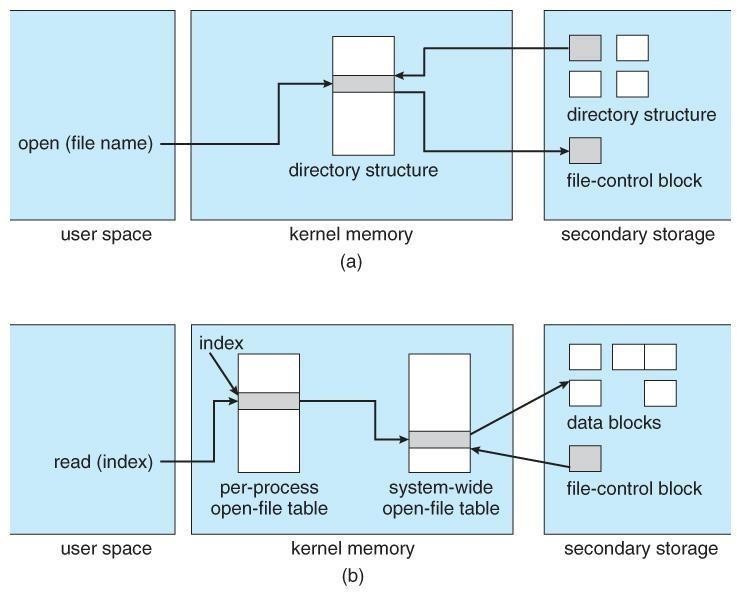
A file has been created; it can be used for I/O. First, it must be opened. The open( ) call passes a file name to the logical file system. The open( ) system call First searches the system wide open file table to see if the file is already in use by another process. If it is ,a per process open file table entry is created pointing to the existing system wide open file table. If the file is not already open, the directory structure is searched for the given file name. Once the file is found, FCB is copied into a system

wide open file table in memory. This table not only stores the FCB but also tracks the number of processes that have the file open.

Next, an entry is made in the per – process open file table, with the pointer to the entry in the system wide open file table and some other fields. These are the fields include a pointer to the current location in the file (for the next read/write operation) and the access mode in which the file is open. The open () call returns a pointer to the appropriate entry in the per-process file system table. All file operations are preformed via this pointer. When a process closes the file the per- process table entry is removed. And the system wide entry open count is decremented. When all users that have opened the file close it, any updated metadata is copied back to the disk base directory structure. System wide open file table entry is removed.

System wide open file table contains a copy of the FCB of each open file, other information. Per process open file table, contains a pointer to the appropriate entry in the system wide open file

table, other information.

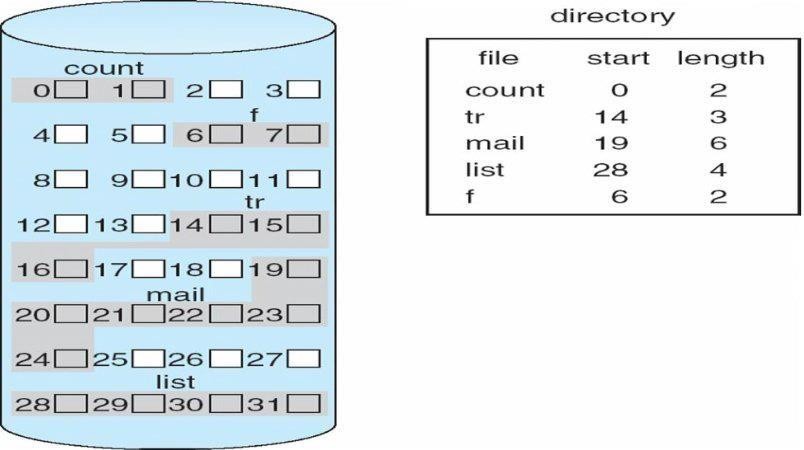


## FILE ALLOCATION METHODS – [refer ppt for advantages and disadvantages]

An allocation method refers to how disk blocks are allocated for files:

**1.Contiguous allocation** – each file occupies set of contiguous blocks o Best performance in most cases

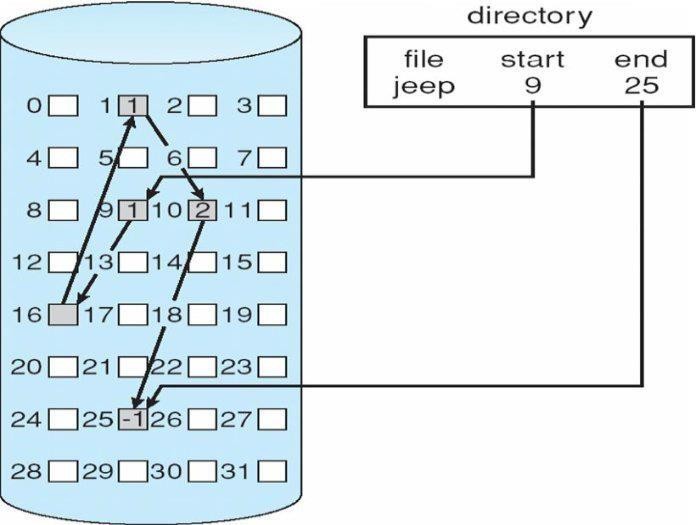
* Simple – only starting location (block #) and length (number ofblocks) are required
* Problems include finding space for file, knowing file size, external fragmentation, need for **compaction off-line** (**downtime**) or **on-line**



## Linked

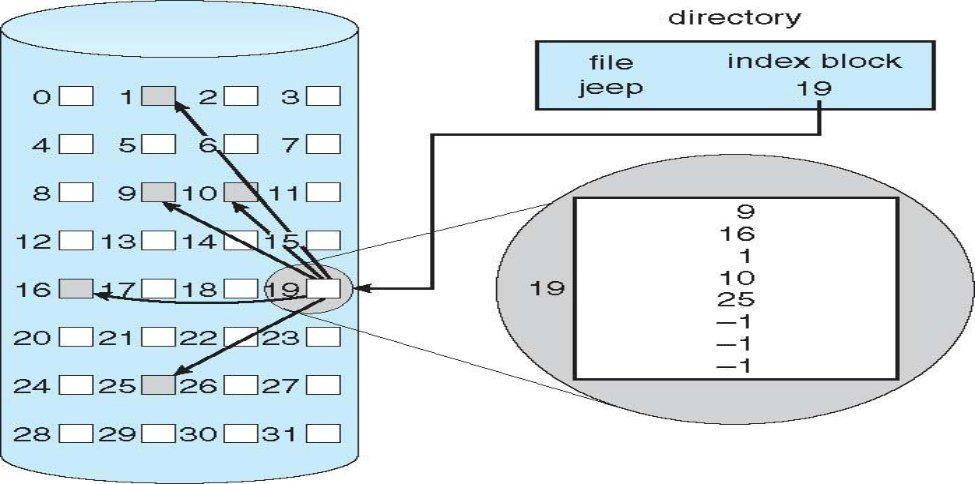
**2.Linked allocation** – each file a linked list of blocks o File ends at nil pointer

* No external fragmentation
* Each block contains pointer to next block
* No compaction, external fragmentation
* Free space management system called when new block needed
* Improve efficiency by clustering blocks into groups but increases internal fragmentation
* Reliability can be a problem
* Locating a block can take many I/Os and disk seeks FAT (File Allocation Table) variation
* Beginning of volume has table, indexed by block number
* Much like a linked list, but faster on disk and cacheable



## File-Allocation Table

**3.Indexed allocation**

* Each file has its own **index block**(s) of pointers to its data blocks

## Free-Space Management

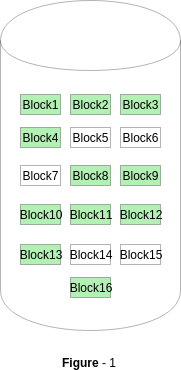
File system maintains **free-space list** to track available blocks/clusters Linked list (free list)

* Cannot get contiguous space easily
* No waste of space
* No need to traverse the entire list

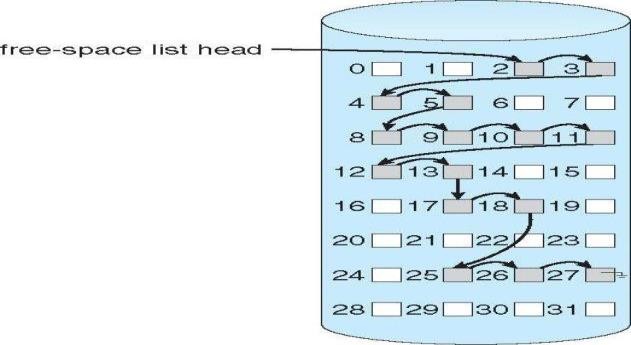
1. **Bit map or Bit vector –** A Bitmap or Bit Vector is series or collection of bits where each bit corresponds to a disk block. The bit can take two values: 0 and 1: 0 indicates that the block is allocated and 1 indicates a free block. The given instance of disk blocks on the disk in Figure 1 (where green blocks are allocated) can be represented by a bitmap of 16 bits as: **0000111000000110**.

## Advantages –

* + Simple to understand.
  + Finding the first free block is efficient. It requires scanning the words (a group of 8 bits) in a bitmap for a non-zero word. (A 0-valued word has all bits 0). The first free block is then found by scanning for the first 1 bit in the non-zero word.



## 2.Linked Free Space List on Disk



In this approach, the free disk blocks are linked together i.e. a free block contains a pointer to the next free block. The block number of the very first disk block is stored at a separate location on disk and is also cached in memory.

## 3.Grouping

Modify linked list to store address of next n-1 free blocks in first free block, plus a pointer to next block that contains free-block-pointers (like this one).

An **advantage** of this approach is that the addresses of a group of free disk blocks can be found easily

## 4.Counting

Because space is frequently contiguously used and freed, with contiguous- allocation allocation, extents, or clustering.

Keep address of first free block and count of following free blocks. Free space list then has entries containing addresses and counts.

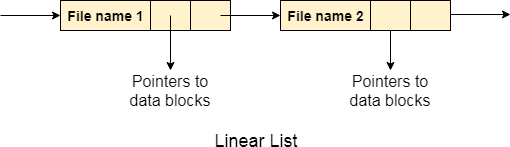
## Directory Implementation

1**. Linear List**

In this algorithm, all the files in a directory are maintained as singly lined list. Each file contains the pointers to the data blocks which are assigned to it and the next file in the directory.

## Characteristics

1. When a new file is created, then the entire list is checked whether the new file name is matching to a existing file name or not. In case, it doesn't exist, the file can be created at the beginning or at the end. Therefore, searching for a unique name is a big concern because traversing the whole list takes time.
2. The list needs to be traversed in case of every operation (creation, deletion, updating, etc) on the files therefore the systems become inefficient.

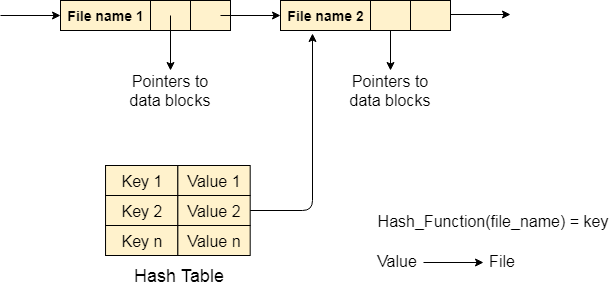


**2.Hash Table**

To overcome the drawbacks of singly linked list implementation of directories, there is an alternative approach that is hash table. This approach suggests to use hash table along with the linked lists.

A key-value pair for each file in the directory gets generated and stored in the hash table. The key can be determined by applying the hash function on the file name while the key points to the corresponding file stored in the directory.

Now, searching becomes efficient due to the fact that now, entire list will not be searched on every operating. Only hash table entries are checked using the key and if an entry found then the corresponding file will be fetched using the value.



**Efficiency and Performance**

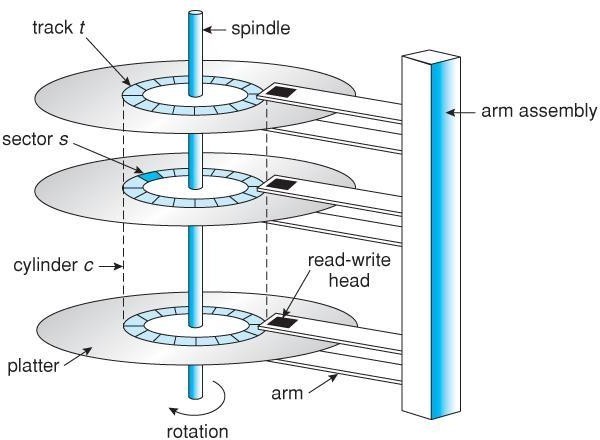
Efficiency dependent on:

* Disk allocation and directory algorithms
* Types of data kept in file’s directory entry Performance
* Disk cache – separate section of main memory for frequently used blocks
* free-behind and read-ahead – techniques to optimize sequential access
* improve PC performance by dedicating section of memory as virtual disk, or RAM disk

## SECONDARY STORAGE STRUCTURE:

**Overview of mass storage structure**

Magnetic disks: Magnetic disks provide the bulk of secondary storage for modern computer system. Each disk platter has a flat circular shape, like a CD. Common platter diameters range from 1.8 to 5.25 inches. The two surfaces of a platter are covered with a magnetic material. We store information by it magnetically on the platters.



### Moving head disk mechanism

A read /write head files just above each surface of every platter. The heads are attached to a disk arm that moves all the heads as a unit. The surface of a platter is logically divided into circular tracks, which are sub divided into sectors. The set of tracks that are at one arm position makes up a cylinder. There may be thousands of concentric cylinders in a disk drive, and each track may contain hundreds of sectors.

When the disk in use, a driver motor spins it at high speed. Most drivers rotate 60 to 200 times per second. Disk speed has 2 parts. The transfer rate is the at which data flow between the drive and the computer. To read/write, the head must be positioned at the desired track and at the beginning of the desired sector on the track, the time it takes to position the head at the desired track is called seek time. Once the track is selected the disk controller waits until desired sector reaches the read/write head. The time it takes to reach the desired sector is called **latency time or rotational dealy- access time**. When the desired sector reached the read/write head, then the real data transferring starts.

A disk can be removable. Removable magnetic disks consist of one platter, held in a plastic case to prevent damage while not in the disk drive. Floppy disks are in expensive removable magnetic disks that have a soft plastic case containing a flexible platter. The storage capacity of a floppy disk is 1.44MB.

A disk drive is attached to a computer by a set of wires called an I/O bus. The data transfer on a bus are carried out by special processors called controllers. The host controller is the controller at the computer end of the bus. A disk controller is built into each disk drive . to perform i/o operation, the host controller operates the disk drive hardware to carry out the command. Disk controllers have built in cache, data transfer at the disk drive happens b/w cache and disk surface. Data transfer at the host, occurs b/w cache and host controller.

**Magnetic Tapes**: magnetic tapes was used as an early secondary storage medium. It is permanent and can hold large amount of data. It access time is slow compared to main memory and magnetic disks. Tapes are mainly used for back up, for storage of infrequently used information. Typically they store 20GB to 200GB.

**Disk Structure**: most disks drives are addressed as large one dimensional arrays of logical blocks. The one dimensional array of logical blocks is mapped onto the sectors of the disk sequentially. sector 0 is the fist sector of the first track on the outermost cylinder. The mapping proceeds in order through that track, then through the rest of the tracks in that cylinder, and then through the rest of the cylinder from outermost to inner most. As we move from outer zones to inner zones, the number of sectors per track decreases. Tracks in outermost zone hold 40% more sectors then innermost zone. The number of sectors per track has been increasing as disks technology improves, and the outer zone of a disk usually has several hundred sectors per track. Similarly, the number of cylinders per disk has been increasing; large disks have tens of thousands of cylinders.

### Disk attachment

Computer access disk storage is 2 ways.

1. Via I/O ports(host attachedstorage)
2. Via a remote host in a distributed file system(network attachedstorage).

**1 .Host attached storage** : host attached storage are accessed via local I/O ports. The desktop pc uses an I/O bus architecture called IDE. This architecture supports maximum of 2 drives per I/O bus. High end work station and servers use SCSI and FC.

SCSI is an bus architecture which have large number of conductor’s in a ribbon cable (50 or 68) scsi protocol supports maximum of 16 drives an bus. Host consists of a controller card (SCSI Initiator) and upto 15 storage device called SCSI targets.

Fc(fiber channel) is the high speed serial architecture. It operates mostly on optical fiber (or) over 4 conductor copper cable. It has 2 variants. One is a large switched fabric having a 24-bit address space. The other is an (FC-AL) arbitrated loop that can address 126 devices.

A wide variety of storage devices are suitable for use as host attached.( hard disk,cd,dvd,tape devices)

1. **Network-attached storage**: A(NAS) is accessed remotely over a data network

.clients access network attached storage via remote procedure calls. The rpc are carried via tcp/udp over an ip network-usually the same LAN that carries all data traffic to theclients.



LAN/WAN

CLIENT

CLIENT

NAS

NAS

NAS provides a convenient way for all the computers on a LAN to share a pool of storage with the same ease of naming and access enjoyed with local host attached storage .but it tends to be less efficient and have lower performance than direct attached storage.

1. **Storage area network**: The drawback of network attached storage(NAS) is storage I/O operations consume bandwidth on the data network. The communication b/w servers and clients competes for bandwidth with the communication among servers and storagedevices.

A storage area network(SAN) is a private network using storage protocols connecting servers and storage units. The power of a SAN is its flexibility. multiple hosts and multiple storage arrays can attach to the same SAN, and storage can be dynamically allocated to hosts. SANs make it possible for clusters of server to share the same storage

## Disk Scheduling Algorithms

Disk scheduling algorithms are used to allocate the services to the I/O requests on the disk . Since seeking disk requests is time consuming, disk scheduling algorithms try to minimize this latency. If desired disk drive or controller is available, request is served immediately. If busy, new request for service will be placed in the queue of pending requests. When one request is completed, the Operating System has to choose which pending request to service next. The OS relies on the type of algorithm it needs when dealing and choosing what particular disk request is to be processed next. The objective of using these algorithms is keeping Head movements to the amount as possible. The less the head to move, the faster the seek time will be. To see how it works, the different disk scheduling algorithms will be discussed and examples are also provided for better understanding on these different algorithms.

### First Come First Serve(FCFS)

It is the simplest form of disk scheduling algorithms. The I/O requests are served or processes according to their arrival. The request arrives first will be accessed and served first. Since it follows the order of arrival, it causes the wild swings from the innermost to the outermost tracks of the disk and vice versa . The farther the location of the request being serviced by the read/write head from its current location, the higher the seek time will be.

Example: Given the following track requests in the disk queue, compute for the Total Head Movement (THM) of the read/write head :

95, 180, 34, 119, 11, 123, 62, 64

Consider that the read/write head is positioned at location 50. Prior to this track location 199 was serviced. Show the total head movement for a 200 track disk (0-199).

### Solution:

**Total Head Movement Computation**: (THM) =

(180 - 50) + (180-34) + (119-34) + (119-11) + (123-11) + (123-62) + (64-62) =

130 + 146 + 85 + 108 + 112 + 61 + 2 (THM) = 644 tracks

Assuming a seek rate of 5 milliseconds is given, we compute for the seek time using the formula: Seek Time = THM \* Seek rate

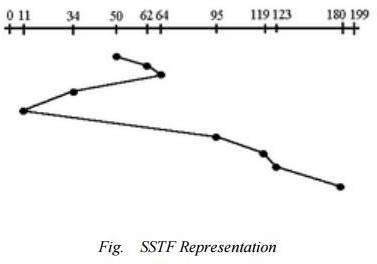
=644 \* 5 ms

Seek Time = 3,220 ms.

### Shortest Seek Time First(SSTF):

This algorithm is based on the idea that that he R/W head should proceed to the track that is closest to its current position . The process would continue until all the track requests are taken care of. Using the same sets of example in FCFS the solution are as follows:

### Solution:



(THM) = (64-50) + (64-11) + (180-11) =

14 + 53 + 169 (THM) = 236 tracks

Seek Time = THM \* Seek rate

= 236 \* 5ms

Seek Time = 1,180 ms

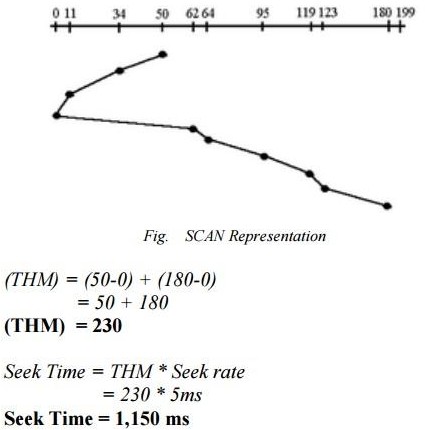
In this algorithm, request is serviced according to the next shortest distance. Starting at 50, the next shortest distance would be 62 instead of 34 since it is only 12 tracks away from 62 and 16 tracks away from 34 . The process would continue up to the last track request. There are a total of 236 tracks and a seek time of 1,180 ms, which seems to be

a better service compared with FCFS which there is a chance that starvation3 would take place. The reason for this is if there were lots of requests closed to each other, the other requests will never be handled since the distance will always be greater.

### SCAN Scheduling Algorithm

This algorithm is performed by moving the R/W head back-and-forth to the innermost and outermost track. As it scans the tracks from end to end, it process all the requests found in the direction it is headed. This will ensure that all track requests, whether in the outermost, middle or innermost location, will be traversed by the access arm thereby finding all the requests. This is also known as the Elevator algorithm. Using the same sets of example in FCFS the solution are as follows:

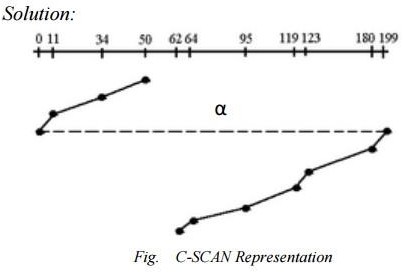
### Solution:



This algorithm works like an elevator does. In the algorithm example, it scans down towards the nearest end and when it reached the bottom it scans up servicing the requests that it did not get going down. If a request comes in after it has been scanned, it will not be serviced until the process comes back down or moves back up. This process moved a total of 230 tracks and a seek time of 1,150. This is optimal than the previous algorithm.

### 4 .Circular SCAN (C-SCAN)Algorithm

This algorithm is a modified version of the SCAN algorithm. C-SCAN sweeps the disk from end-to-end, but as soon it reaches one of the end tracks it then moves to the

other end track without servicing any requesting location. As soon as it reaches the other end track it then starts servicing and grants requests headed to its direction. This algorithm improves the unfair situation of the end tracks against the middle tracks. Using the same sets of example in FCFS the solution are as

follows:

Notice that in this example an alpha3 symbol (α) was used to represent the dash line. This return sweeps is sometimes given a numerical value which is included in the computation of the THM . As analogy, this can be compared with the carriage return lever of a typewriter. Once it is pulled to the right most direction, it resets the typing point to the leftmost margin of the paper . A typist is not supposed to type during the movement of the carriage return lever because the line spacing is being adjusted . The frequent use of this lever consumes time, same with the time consumed when the R/W head is reset to its starting position.

Assume that in this example, α has a value of 20ms, the computation would be as follows: (THM) = (50-0) + (199-62) + α

= 50 + 137 + 20 (THM)

= 207 tracks

Seek Time = THM \* Seek rate

= 187 \* 5ms Seek Time = 935 ms .

The computation of the seek time excluded the alpha value because it is not an actual seek or search of a disk request but a reset of the access arm to the starting position .

**System Protection:**

**Protection**

* Operating system consists of a collection of objects, hardware or software
* Each object has a unique name and can be accessed through a well-defined set of operations.
* Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.

The need to protect files is a direct result of the ability to access files.

* complete protection by prohibiting access not useful.
* Or, free access with no protection.

What is needed is **controlled access**.

* Types of access
* Read (“r” in Unix)
* Write (“w”)
* Execute (“x”)
* Append (“w”)
* Delete (owner)
* List (“r” for a directory)
* Search (“x” for a directory)
* Other operations (renaming, copying, and editing) use these basic set of operations

**Protection - Access Control**

Most common approach is to make access dependent on the identity of the user

* Generally implemented using access control list (ACL)
  + specifying user names and the types of access allowed for each user.
  + Each request is checked against this list to determine whether it is valid or not.
* A condensed version is usually used.
  + Owner, Group, Universe.
* Only three fields are needed to define protection
  + UNIX system defines three fields of 3 bits each - **rwx**
* Another approach for protection is to assign each file a password
  + Large # of passwords to remember
  + Protection is on an all-or-none basis if one password is used for all files

**Protection of UNIX access lists**

|  |  |  |
| --- | --- | --- |
| * Three modes of access: read, write, execute * Three classes of users |  | RWX |
| a) owner access | 7 | Þ 1 1 1 |
|  |  | RWX |
| b) group access | 6 | Þ 1 1 0 |
|  |  | RWX |
| c) public access | 1 | Þ 0 0 1 |

**Protection - A Sample UNIX Directory Listing**

-rwx------ 1 cs4520 users 20096 Sep 15 14:31 a.out

-rw-r--r-- 1 cs4520 student 1764 Aug 30 16:05 assign1.c

-rw------- 1 cs4520 users 4624 Aug 30 15:48 assign1.lis

**Protection Summary**

* File systems implement some kind of protection system

Who can access a file

How they can access it

* Objects are “what”, subjects are “who”, actions are “how”
* A protection system dictates whether a given action performed by a given subject on a given object should be allowed

Can read and/or write your files, but others cannot

can read “/etc/motd”, but you cannot write it

* Ask manager to create a group (unique name), say G, and add some users to the group.
* For a particular file (say *game*) or subdirectory, define an appropriate access.
* Access control lists.
  + Each entry specifies user/group name and types of access allowed.

owner group public

chmod 761 game

chgrp G game

**Goals of Protection**

* + Operating system consists of a collection of objects, hardware or software
  + Each object has a unique name and can be accessed through a well- defined set of operations.
  + The operations that are possible may depend on the object (read , write, rewind, open,…etc)
  + Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so.
  + Protection: control access to a system by limiting the types of file access permitted to users.
  + Ensure that only processes that have gained proper authorization from the operating system can operate on memory segments, the CPU, and other resources.
  + The O.S. provides protection mechanisms, which are described, so that an application designer can use them in designing her or his own protection software.

**Principles of Protection**

* Guiding principle – principle of least privilege
* Programs, users and systems should be given just enough privileges to perform their tasks.
* The role of protection in a computer system is to provide a mechanism for the enforcement of the policies governing resource use.
* Mechanism vs Policy
* Mechanisms determine ***how*** something will be done; policies decide ***what*** will be done
* Guiding principle – principle of least privilege
* Programs, users and systems should be given just enough privileges to perform their tasks
* failure or compromise of an OS component does the minimum damage and allows the minimal damage to be done
* *need-to-know* principle: a process should be able to access only those resources that it currently requires to complete its task
* useful in limiting the amount of damage a faulty process can cause in the system.

**Domain of Protection:**

* A process operates within a protection domain, which specifies the resources that the process may access.
* Each domain defines a set of objects and the types of operations that may be invoked on each object.
* The ability to execute an operation on an object is an access right.
* A domain is a collection of access rights, each of which is an ordered pair: <object-name, rights-set>

Example: If domain D has the access right: <file F, {read, write}>, then a process executing in domain D can only read and write file F.

**Domain Structure**

* Access-right = <object-name, rights-set>

where rights-set is a subset of all valid operations that can be performed on the object.

* Domain = set of access-rights
* Domains may share access rights.
* A process executing in either D2 or D3 can print O4
* A process must be executing in D1 to read and write O1. Also, only process in D3 may execute O1
* A process operates within a protection domain, which specifies the resources that the process may access.
* Each domain defines a set of objects and the types of operations that may be invoked on each object.
* The ability to execute an operation on an object is an access right.
* A domain is a collection of access rights, each of which is an ordered pair: <object-name, rights-set>
* Example: If domain D has the access right: <file F, {read, write}>, then a process executing in domain D can only read and write file F.

**Domain Implementation (UNIX)**

System consists of 2 domains:

* + - * User
      * Supervisor

UNIX

* + - * Domain = user-id
      * Domain switch accomplished via file system.
      * Each file has associated with it a domain bit (setuid bit).

When file is executed and setuid = on, then user-id is set to owner of the file being executed. When execution completes user-id is reset.