A Handbook on
Computer Science

8

Operating System

CONTENTS

Processes and Threads

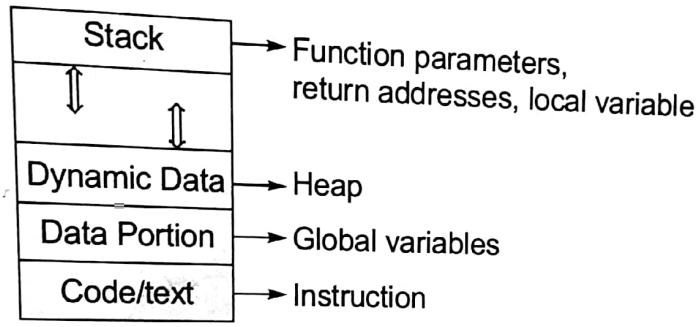
1

Process Concepts

- **System call:** It is a request made by a user program to the operating system in order to get any kind of service.
- **Process:** Program under execution. It is an instance of a program.

Program	Process
Passive entity	Active entity
Resides on disk	Resides in main memory

- **Abstract view of process:**



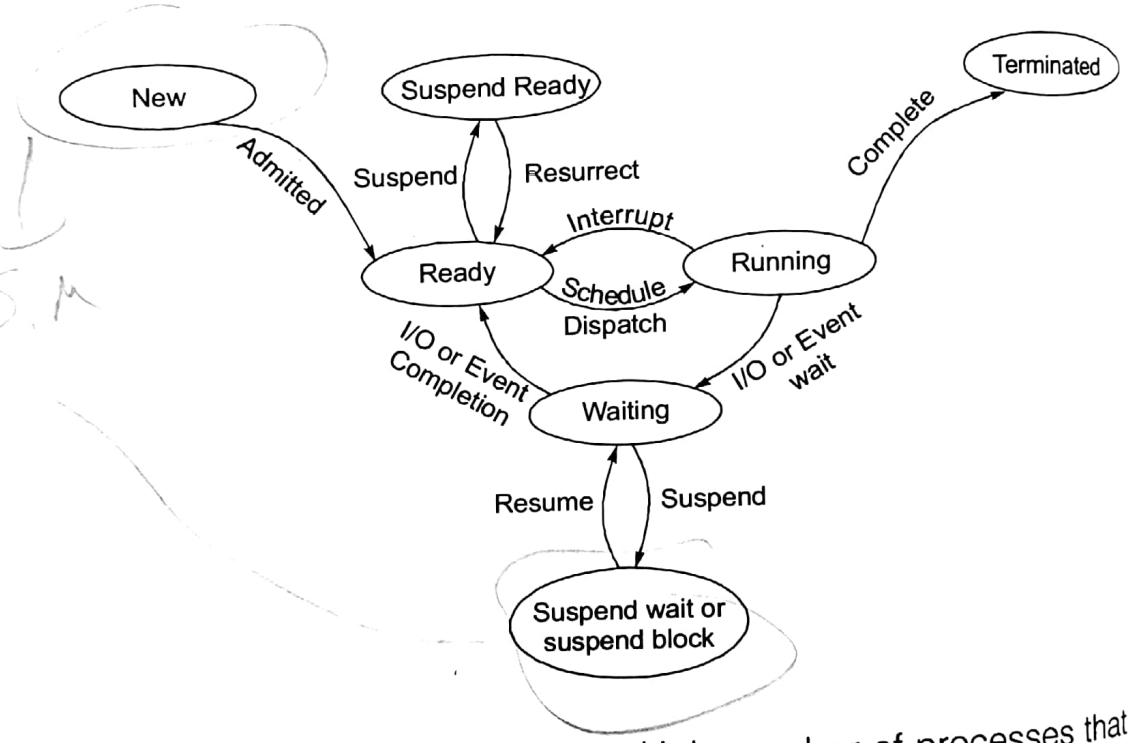
- **Operations on process:**
  - (*i*) Create a process
  - (*ii*) Scheduling/running, block a process, suspend, resume
  - (*iii*) Terminate a process
- **Dual modes of operation in operating system:**
  - (*i*) User Mode/Non privileged Mode
    - (a) Preemption is always possible
    - (b) User program executes in this mode
  - (*ii*) Kernel Mode/Supervisory/System/Privileged Mode
    - (a) No preemption (atomic execution)
    - (b) OS components executes in this mode
- **Process Control Block (PCB):** Each process is represented by a PCB. It is associated with every process to maintain process ID, starting address of page table, open file information, CPU information, security information, etc.

  Attributes of PCB:

  (*i*) Process ID (PID) and Parent Process ID (PPID)

(*ii*) Process State
(*iii*) Program Counter
(*iv*) CPU Registers
(*v*) CPU Scheduling Information
(*vi*) Memory Management Information
(*vii*) Accounting Information
(*viii*) Address space for the process
(*ix*) User Identification
(*x*) I/O status
(*xi*) List of open files

Process State Diagram



- In the wait state, there will be multiple number of processes that wi perform I/O operations simultaneously.
- When the process is in ready running or wait state, it is residing in mai memory, otherwise the process is in secondary memory.

Threads

Problems with Traditional Processes

- Traditional processes have separate address spaces.
- Process creation and switching is expensive
- Sharing memory areas among processes is non-trivial.

Thread

Thread is a basic unit of CPU utilization that shares along with the other threads of the same process.

- The address space consists of code section, data section and other common resources like files, signals, semaphore etc.
- Threads also contain thread specific data which is not shared along with the peer threads of the same process.
- Each thread maintains its own register set and a stack to monitor the control within the code section of the process.
- Thread is a light weight process.
- Context switch overhead is less in thread based systems compared to process based systems.

Difference between ULT and KLT

User Level Threads (ULT)	Kernel Level Threads (KLT)
(a) ULTs are created by the application packages or libraries at the user level without taking any support of Operating System (OS).	(a) Scheduling must be through OS.
(b) Thread management is done at user mode.	(b) Thread management is done at Kernel mode
(c) Fastest context switching	(c) Lowest context switching
(d) Entire process gets blocked when a thread requires an I/O device or any system call.	(d) No other process get blocked except the specific thread which needs I/O will be blocked.

Benefits of Multithreading

1. Responsiveness
2. Resource sharing
3. Economical Design (contest switch overhead is less).
4. Utilization of multiprocessor architectures.

Multithreading Models

1. **Many-to-one model:** It maps many user level threads to one Kernel thread, multiple threads cannot run in parallel on multiprocessors.
2. **One-to-one model:** It maps each user thread to a Kernel thread. Creating a user thread requires creating the corresponding Kernel thread.
3. **Many-to-many model:** It multiplexes many user threads to a smaller or equal number of Kernel threads.

■■■

IPC, Concurrency and Synchronization

2

Inter-Process Communication

- **Independent processes:** Two processes can execute independently and they will not affect the output with the order of execution.
  (*i*) No states shared with other processes.
  (*ii*) Order of scheduling does not matter.
  (*iii*) output does not depend on order of execution of processes.
- **Co-operating processes:** Two or more processes are said to be co-operating iff they get affected by or affects the execution of other process.
  (*i*) Co-operating processes require an Inter-Process Communication (IPC) mechanism that will allow them to exchange data and information like messages, pipes, shared memory, shared variables.
  (*ii*) Shared resources, speed-up and modularity are the advantages of co-operating processes.
- **Competing processes:** Two or more processes are said to be competing iff they race to access a shared resource concurrently (simultaneously).

IPC Models

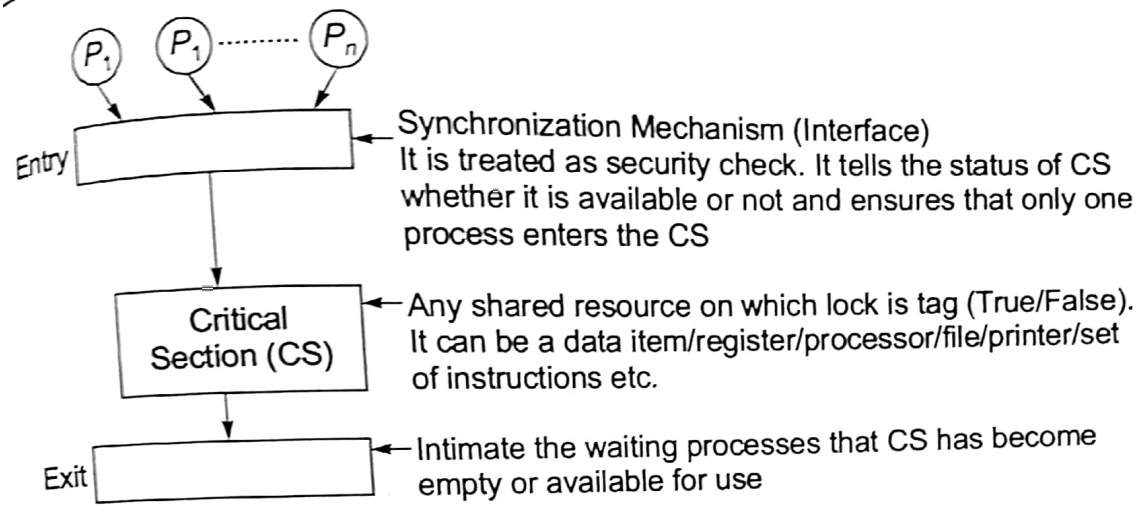
- Shared Memory.
- Message Passing.
  (*i*) Synchronous or Asynchronous Communication
  (*ii*) Direct or Indirect Communication

Concurrency

- Able to run multiple applications at the same time.
- Allows better resource utilisation.
- Better average response time of applications.
- Achieves better performance.

Synchronization

Synchronization is to use atomic operations to ensure co-operation between processes.



- **Critical Section:** That portion of the program text where shared resources are accessed.
- **Race Condition:** The final output of any variable depends on the execution sequence of the process. This type of condition is called as race condition.
- **Requirements for Synchronization Mechanism**
  - (*i*) **Mutual Exclusion:** It states that no two processes may be simultaneously present in the critical section (i.e. at the same time).
  - (*ii*) **Progress:** No process running outside the critical section should block the other interested process from entering into critical section when critical section is free.
  - (*iii*) **Bounded Waiting:** Due to mutual exclusion waiting time of a process must be definite (This is achieved by writing release of locks without forgetting).
- **Lack of Synchronization leads to:**
  - (*i*) Inconsistency
  - (*ii*) Loss of data
  - (*iii*) Deadlock
- **Locking Mechanisms:**
  - (*i*) The test and set operation
  - (*ii*) Busy waiting
  - (*iii*) Spin Locks

Semaphore

- It is special variable whose value indicates information about lock.
- **Types of Semaphore:**
  - (*i*) Counting Semaphore
  - (*ii*) Binary Semaphore

- Operations on Semaphore:
  - (*i*) Wait (P) (Down):
    - (a) It is used before entering critical section.
    - (b) It decrements semaphore value by 1.
    - (c) If resource is not free then block the process.
  - (*ii*) Signal (V) (Up):
    - (a) It is used after critical section to release the acquired lock.
    - (b) It increments semaphore value by 1.

Wait (P)
Access Critical Section
Signal (V)

⇒ Semaphore operations are executed atomically

- **Binary Semaphore:**

Semaphore *b*;
initial value *b* =1;

```
wait (b)                    signal (b)
{                           {
    while (b==0);               b++;
    b – –;
}                           }
```

- **Limitations:**
  - (*i*) There is busy wait called spin locks in binary semaphore.
  - (*ii*) Binary semaphore can work on a single instance of resource.
  - (*iii*) Binary semaphore wait operation should complete for a single process only.
- **Counting Semaphore:**

Semaphore c;
initial value c ≡ Number of instances of type *R*

```
wait (c)                    signal (c)
{                           {
c– –;                       c++;
if(c < 0)                   if(c <= 0)
{                           {
place current process       wake-up one waiting process on 'c';
 in waiting queue;          }
}                           }
}
```

Note:

- Negative value of counting semaphore indicates number of request in waiting queue waiting on type *R*.
- Positive value indicate number of instance of type *R* currently free for allocation and there is no request in waiting queue on type *R*.
- When we use multiple semaphores then it is highly important to follow proper order of semaphores (Sometimes it unnecessarily blocks the critical section).

Classical Problems on Synchronization using Semaphore

1. Producer Consumer Problem.
2. Reader Writer Problem:

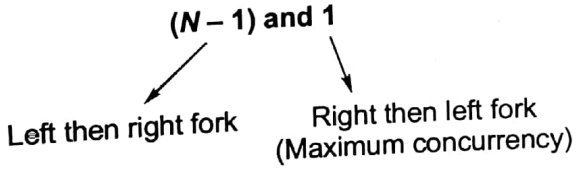
(*i*) N readers are allowed simultaneously
(*ii*) If a reader is reading
  (a) New reader is allowed
  (b) New written is NOT allowed
(*iii*) If a writer is writing
  (a) New reader
  (b) New writer
  } Not allowed

(*i*), (*ii*) } Shared Lock
(*iii*) } Exclusive Lock

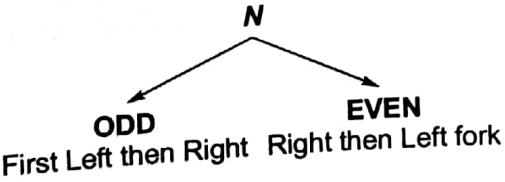
(*iii*) **Dining Philosopher Problem:** Number of philosopher N ≥ 2.

Solution:

(a)



(b)



Peterson Solution for Two Process and One shared Resource

```
# define N 2
# define TRUE 1
# define FALSE 0
int turn;
int interested [N] = {FALSE};
void entry-section (int process)
{
    int other;
    other = 1 – process;
```

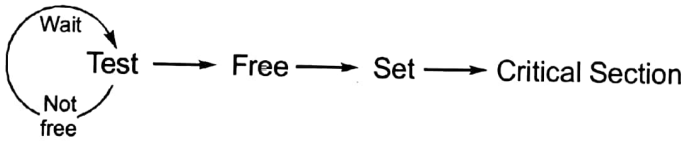
```
        interested [process] ≡ TRUE;
        turn = process;
        while (interested [other] == TRUE && turn ==process);
    }
        Critical section
        void exit section (int process)
        {
          interested [process] = FALSE;
        }
```

- Peterson solution can work for only two processes and one shared resource.
- Peterson solution has a drawback called busy wait or spinlock.
- Peterson solution needs operating system intervention so that only one waiting process enters into critical section.
- It suffers from priority inversion problem.

Test and Set Hardware Instructions

Test: To check whether resource is free or not

Set: Sets LOCK




```
Boolean Test and set( )
  {
    if(Locked == T)
    {                    } Test
      Return(T)
    }
  else
    {
      Locked = T
      Return(F)          } Set
    }
  }
```

```
While(1)
  {
    while (Test and Set( ))
    {
      Do-nothing
    }
  Access Critical Section
  Locked = F  // Release of Lock
    }
```

Monitor

- Monitors can be used to separate "use locks for mutual exclusion" and "condition variables for scheduling constraints".
- Monitor is a special kind of module or package which includes variables and data structures that are all grouped together at one place.

- Procedures running outside the monitor cannot access the monitor's internal variable and data structures. However they can activate monitor's internal procedures.
- Monitors have an important property that only one process can be active in the monitor at any time.
- A monitor is a lock with zero or more conditional variable for managing concurrent access to share data. The lock provides mutual exclusion to the shared data. A conditional variable allows a queue of waiting processes while being inside a critical section. This property is ensured by using a binary semaphore at the beginning and at the end of monitor.

Spinlock and Busy Waiting

Spinlock is a lock which causes a thread trying to acquire it to simply wait in a loop ("spin") while repeatedly checking if the lock is available. Since the thread remains active but is not performing a useful task, the use of such a lock is a kind of **busy waiting**.

■■■

Deadlock

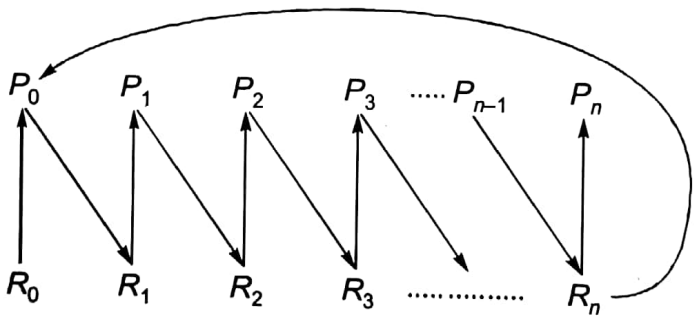
3

Introduction

Two processes are involved in dead lock iff they wait for happening of an event which would never happen. Minimum two processes are required for deadlock.

Necessary Conditions for Deadlock

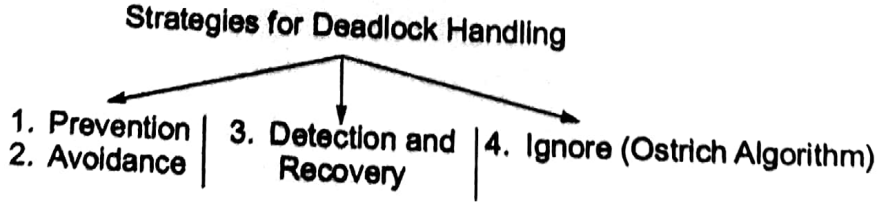
1. **Mutual Exclusion:** Only one process at a time can use the resource.
2. **Hold and Wait:** A process *'A'* holding a resource and waiting for another resource held by process *'B'*.
3. **No Preemption:** A resource can be released only voluntarily by the process holding it after completion of task.
4. **Circular Wait:** Let P_0, P_1, ..., P_n be the processes and R_0, R_1,..., R_n be the resources. P_0 is holding R_0 and waiting for R_1 which is held by P_1.



Note:

- If Resource Allocation Graph (RAG) has single instance and multiple instances or only multiple instances, cycle is only necessary condition for deadlock.
- If RAG has single instance resource cycle is necessary and sufficient condition for deadlock.

Strategies for Deadlock Handling



Prevention

It involves the following:

- Infinite resources
- No sharing
- No waiting
- Preempt resources
- Allocate all resources at the beginning
- Make everyone use the same ordering in accessing resources.
- A combination of techniques.

Avoidance through Banker's Algorithm

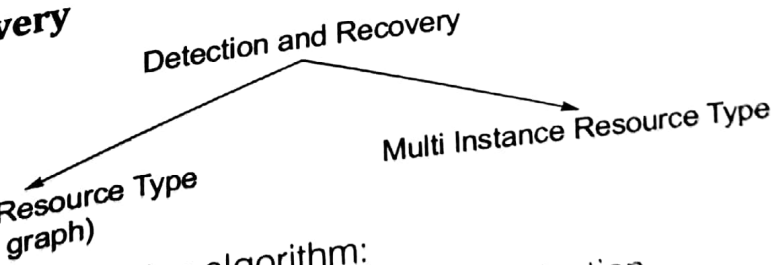
- Single instance of resources using resource allocation graph.

Note:

- A system is said to be safe iff the needs of all the processes could be satisfied with the available resources in some order.
- The order is known as safe sequence
- There may be multiple safe sequences at any given time.
- Need = Max – Allocation

- Multiple Instance of resources:
  - (*i*) Resource Request
  - (*ii*) Safety Algorithm

Detection and Recovery



- When we activate detection algorithm:
  - (*i*) CPU utilization has decreased or no CPU utilization
  - (*ii*) Most of the processes are blocked.
- Deadlock detection and Recovery involves the following approach:
  - (*i*) Scan the resource allocation graph
  - (*ii*) Detect a cycle in graph
  - (*iii*) Recovery From Deadlock

Deadlock Recovery

1. **Process Termination:** Two methods are available
   (*i*) Abort all deadlocked processes
   (*ii*) Abort one process at a time until the deadlock cycle is eliminated.
2. **Resource Preemption:** To eliminate deadlock using resource preemption, we successfully preempt some resources from processes and give these resources to other processes until the deadlock cycle is broken. Here three issues need to be addressed:
   (*i*) Selecting a Victim: Which resource should be preempted.
   (*ii*) Rollback: Rollback to safe state and restart from that state.
   (*iii*) Starvation

Note:

$$\boxed{r \geq p(n-1)+1;}$$

r = number of available resources (deadlock free)

n = maximum number of resources needed by each process.

p = number of processes.

■■■

CPU Scheduling

4

Introduction

Multiprogramming

- The objective of multiprogramming is to have some process use CPU cycles at all times, to maximize CPU utilization.

Time Sharing

- The objective of time sharing is to switch the CPU time among processes so frequently that users can interact with each program while it is running.
- The response time should be short.

Scheduling Queues

- **Job Queue:** As processes enter the system, they are put into job queue.
- **Ready Queue:** The process that are residing in main memory and are ready and waiting to execute are kept in ready queue.
- **Device Queue:** List of processes waiting for a particular I/O device. Each device has its own device queue.

Scheduler

A scheduler handles the removal of the running process from the CPU and the selection of the next running process.

A scheduler is implemented to achieve the following:

1. To minimize the waiting time.
2. To maximize the throughput (number of completed jobs per unit time).
3. To achieve fairness.

Types of Schedulers

1. **Long-term/Job scheduler:** Job scheduler controls the degree of multiprogramming (number of processes in memory). It brings processes from new state to ready state.
2. **Short-term/CPU scheduler:** CPU scheduler decides which process should run on CPU from ready queue.
3. **Medium-term scheduler:** The key idea behind a medium term scheduler is that sometimes it is advantageous to remove processes from memory and thus reduce the degree of multiprogramming. Later, the process

can be reintroduced into memory, and its execution can be continued where it is left-off. It brings processes from running state to suspend ready state or from wait state to suspend wait stage.

Context Switch

- Switching the CPU to another process requires performing a '**state save**' of the current process and a '**state restore**' of a different process. This take is known as a context switch.
- Context-switch is pure overhead, because system does no useful work while switching.

Dispatcher

- The dispatcher is the module that gives control of the CPU to the process selected by CPU scheduler.
- The function involves switching context, switching to user mode and jumping to the proper location in the user program to restart the program.
- The time taken by the dispatcher to stop one process and start another running, is known as the "Dispatch Latency".

Scheduling Criteria

- **CPU Utilization:** CPU should be as busy as possible
- **Throughput:** The number of processes that are completed per unit time
- **Turnaround Time:** Completion Time – Arrival Time
- Waiting Time = Turnaround Time – Burst Time
- Response Time = First Response Time – Arrival Time

Starvation

- Indefinite waiting of a process to get CPU cycles.

CPU Scheduling Algorithms

The following algorithms are priority/non-priority based and preemptive/ non-preemptive based.

1. **First Come First Serve (FCFS)/ First-in First-out (FIFO):** It assigns the processor in the order of arrival of requests. This algorithm is non-preemptive. In this policy, the disadvantage is shorter jobs can get stacked behind the long running jobs.

   Convoy Effect: If the first process is having large burst time, then it will have major impact on average waiting time of other processes.
2. **Shortest Job First (SJF)/Shortest Job Next (SJN):** It favours shorter jobs, it is used as a benchmark in order to compare its performance

with other process. This algorithm is non-preemptive. In this policy, the advantage is "turnaround time can be minimised". The disadvantage is it can lead to starvation of processes.

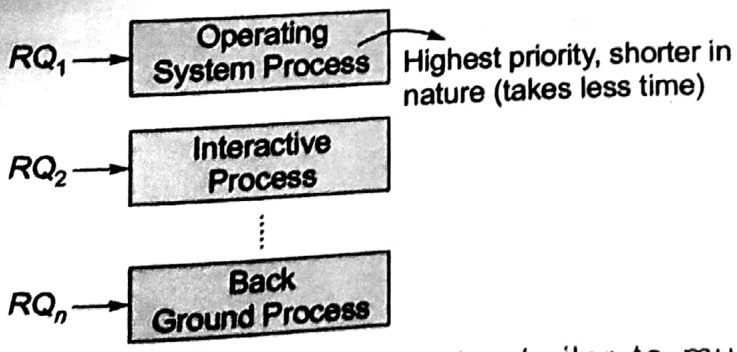
"Aging": It is a technique of gradually increasing the priority of processes that wait in the system for a long time. It avoids starvation.

3. **Shortest Job Remaining Time First (SRTF) scheduling:** Preemptive SJF scheduling is called Shortest Remaining Time First (SRTF).
4. **Round Robin Scheduling:** Designed for *time sharing* system CPU is allocated to each process for certain time interval called time *quantum* (time slice). It is preemptive algorithm. If time quantum decreases, no of context switch increases, response time decreases.
5. **Longest Job First:** It is a non preemptive algorithm, completely opposite to SJF.
6. **Longest Remaining Time First:** LRTF is preemptive LJF.
7. **Priority based scheduling** can be preemptive or non preemptive.
8. **Highest Response Ratio Next (non preemptive):** Selects the highest '**response ratio**' process

$$\text{Response Ratio} = \frac{\text{Waiting time} + \text{Burst time}}{\text{Burst time}}$$

9. **Multilevel Queue Scheduling:** As only one ready queue is there so only one scheduling technique is used. Therefore, to use more than one scheduling can be used with multilevel queue scheduling.

Each queue can maintain different priority.



10. **Multilevel Feedback Queues:** It is similar to multilevel queue scheduling.

Multilevel feedback queues adjust each job's priority as follows:

(*i*) Job starts in the highest priority queue.

(*ii*) If time slice expires, drop the job by one level.

(*iii*) If time slice does not expire, push the job up by one level.

11. **Lottery Scheduling:** Lottery scheduling is another adaptive scheduling approach that addresses the fairness problem of multilevel feedback queues. The lottery scheduler gives every job some number of lottery tickets, and on each time slice, it randomly picks a winning ticket. On average, the CPU time allotted is proportional to the number of tickets given to each job.

Scheduling algorithm	Starvation possibility
FCFS	No
SJF	Yes
SRTF	Yes
Round Robin	No
LJF	Yes
LRTF	Yes
Non preemptive priority based	Yes
Preemptive priority based	Yes
Highest response ratio next	No
Multilevel queue	Yes
Multilevel feedback queue	No

■■■

Memory Management and Virtual Memory

5

Memory Management

Dynamic Loading

- It refers to loading library into the memory during load or run time.
- A routine is not loaded until called and all routines are kept in "relocatable format".

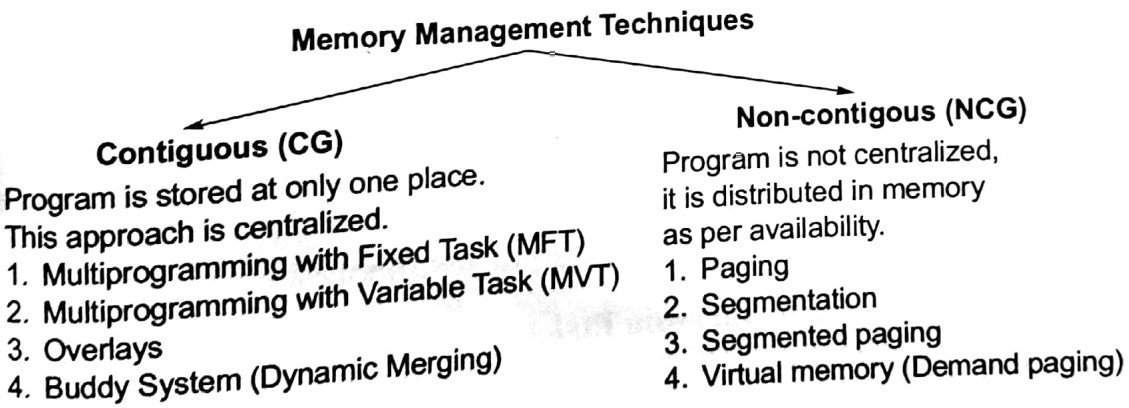
Dynamic Linking

- It refers to the linking that is done during load or runtime and not when .exe is created.
- It requires support from operating system.
- Library which link dynamically is called Dynamic Link Library (DLL).

Static Linking

All modules of program are stored in memory before calling.

Memory Management Techniques



Partitions

The following **functions** can be used for partitions:

1. Allocation
2. Protection (using limit registers)
3. Free space management
4. Deallocation

The following **goals** should be achieved for partitions:

1. Minimum wastage of memory (fragmentation)
2. Run (manage) larger programs in smaller memory area.

Contiguous Memory Allocation

Two ways which support the above goals are overlays and virtual memory so that degree of multiprogramming will increase, there is more CPU utilization, throughput increases.

Overlays

- It is a very old memory management technique focus is on a single process information retained in RAM at a time.
- It will suit to sequential processing but multitasking, multiprogramming cannot be supported.
- Overlaying is possible when the program is divided into modules and the modules are independent.

Allocation Policies

1. **First Fit:** Allocation of first available free space.
2. **Best Fit:** Allocation of best available free space.
3. **Worst Fit:** Allocation of largest available free space.
4. **Next Fit:** Same as first fit, but it starts from last allocated process.

Multiprogramming with Fixed Task

- Maximum process size $\propto$ Maximum partition size.
- Degree of multiprogramming $\propto$ Number of partitions.
- Suffers from both Internal and External fragmentation.

Multiprogramming with Variable Tasks

- Degree of multiprogramming varies.
- Suffers from only External Fragmentation.
- Worst Fit is the best allocation policy.

Buddy System

- The "buddy system" allocates memory from a fixed size segment consisting of physically contiguous pages.
- Memory is allocated from this segment using a power-of-2 allocator, which satisfies requests in units sized as a power of 2.

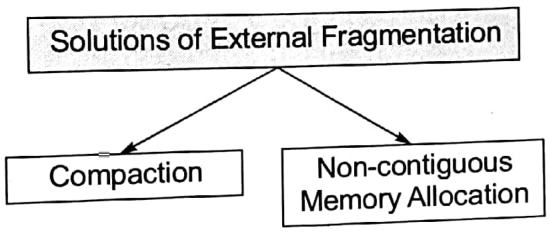
- An advantage of the buddy system is how quickly adjacent buddies can be combined to form larger segments using a technique known as "coalescing".

Internal Fragmentation

In fixed size partitioning if process size is smaller than partition size some unused free space is left within the partition and these are called memory fragments/holes/waste memory/unused free space and their sum is called internal fragmentation.

External Fragmentation

In variable size partitioning initially processes are contiguous but as processes leave their space these partitions may not suit the new processes for allocation. Those unused partitions is called External Fragmentation.



Compaction

- Processes are on one side and free space is on other side in the memory.
- During memory compaction:
  - (*i*) Block processes for compaction
  - (*ii*) Move them towards lower address space
  - (*iii*) Allocate memory for pending request
  - (*iv*) Resume blocked process w.r.t. compaction
- It consumes CPU time (overhead)
- Program need to support the dynamic allocation.

Non-contiguous Allocation

Size of Logical Address Space (LAS) generated by CPU is greater than or equal to size of Physical Address Space (PAS) generated by Main Memory. So that there will be efficient use of multiprogramming, can run larger programs in smaller memory area.

Paging

Logical Address Space

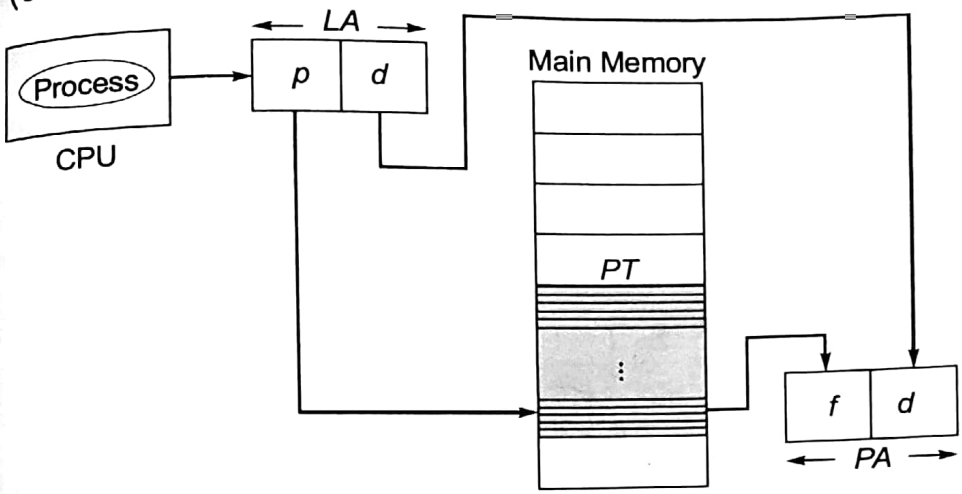
- Logical Address Space is divided into equal size pages.
- Page size (PS) is generalisation of power of 2.
- Number of pages in LAS (N) = $\frac{\text{LAS}}{\text{PS}}$
- Page No (P) bits = $\lceil \log_2 N \rceil$
- Page offset depends on page size
- Page No dependents on number of pages
- Page offset (d) bits = $\lceil \log_2 PS \rceil$

Physical Address Space

- Physical Address Space is divided into equal size frames (Page frames)
- Frame size = Page size
- Number of frames (M) = $\frac{\text{PAS}}{\text{PS (FS)}}$
- Frame No (f) bits = $\lceil \log_2 M \rceil$
- Frame offset = Page offset = d

Page Table

- Each process has its own pagetable stored in memory.
- Pagetable is organised as series of entries.
- Number of entries in page table = Number of pages in logical address space (single level paging)
- Page table entry contain frame number.
- Page table implementation maintains a four-bit per page table entry.
  - (*i*) **Use bit:** Set when a page is referenced, cleared by the clock algorithm
  - (*ii*) **Modified bit:** Set when page is modified, cleared when a page is written to disk
  - (*iii*) **Valid bit:** Set when a program can make legitimate use of this page table entry
  - (*iv*) **Read-only:** Set for a program to read the page, but not to modify it.

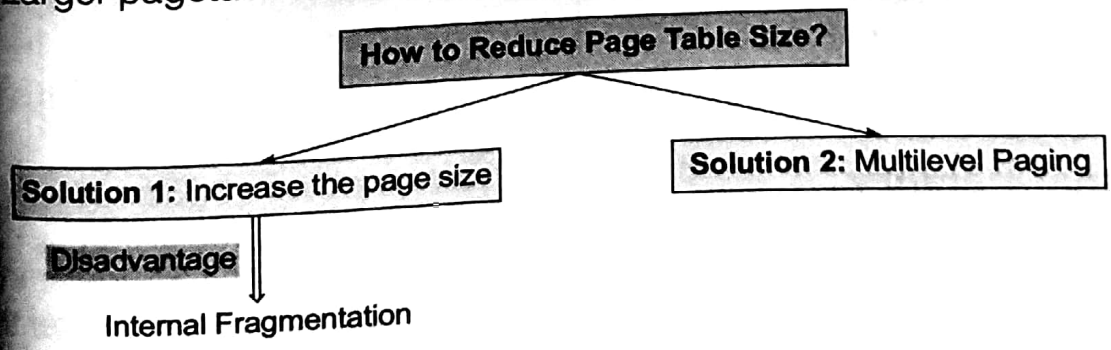
- Pagetable (PT) size in bytes = $N \times e$; e = page table entry size in bytes ($e \geq 1$)
- 

Performance of Paging

- **Paging without TLB:**
  - (*i*) Main Memory Access Time = '*m*'
  - (*ii*) Effective Memory Access Time (EMAT) = '2*m*'
- **Paging with TLB:**
  - (*i*) TLB Access Time = '*c*' [$c <<< m$]
  - (*ii*) TLB Hit Ratio(x) = $\frac{\text{Number of hits}}{\text{Total References}}$
  - (*iii*) Effective Memory Access Time = $x(c + m) + (1 - x)(c + 2m)$

ltilevel Paging

Larger pagetable is not desirable because they take lot of memory.



timal Page Size

LAS = '*S*' bytes
PT entry size = '*e*' bytes
Page size (PS) = *P*

- Page Table (PT) size = $\frac{S}{P} \times e$
- Internal Fragmentation (IF) = $P/2$
- Total overhead = PT size + IF = $\left(\frac{P}{2} + \frac{S}{P} \times e\right)$ = Should be minimum
- Differentiating w.r.t. to P

$$\frac{d}{dP}\left(\frac{P}{2} + \frac{S}{P} \times e\right) \Rightarrow \frac{1}{2} + \left(\frac{-1}{P^2} \times Se\right) = 0$$

$$\Rightarrow \frac{Se}{P^2} = \frac{1}{2}$$

$$\Rightarrow P = \sqrt{2\,Se}$$

- Performance of multilevel paging:

$$EMAT = x(c + m) + (1 - x)\,(c + (n+1)\,m)$$

where n is number of levels in paging, c is cache access time and m is memory access time.

Difference between Paging and Segmentation

Paging	Segmentation
(a) Paging suffers from Internal Fragmentation only in the last page.	(a) There is no Internal Fragmentation
(b) Paging does not preserve users view of memory allocation	(b) Segmentation is visible to the programmer
(c) There is no External Fragmentation	(c) There is External Fragmentation

Virtual Memory

- Virtual memory gives an illusion to the programmer that a huge amount of memory is available for waiting and executing programs greater than size of available physical address space.
- **Page Fault Service:**
  - (*i*) Process gets blocked
  - (*ii*) Change the mode bit

    Virtual Memory Manager (VMM) runs in Kernel mode

(iii) VMM → Device Manger → Device Driver → Disk (Device)

(iv) Cases:

(a) Empty frame may be available

(b) No empty frame is available ⇒ Use page replacement algorithm

- More number of page faults, Page Fault Service Time (PFST) is more, execution time increases and throughput decreases.
- Virtual memory is implemented by demand paging.

Performance of Virtual Memory

Effective Memory Access Time (EMAT):

$$EMAT_{DP} = P \times s + (1 - P)m$$

Main Memory Access Time = 'm' [in nsec/μsec]

PFST = 's' [$s >>> m$] (in msec)

Page fault rate = 'P'

Page hit ratio = $1 - P$

Demand Paging

- It allows the pages that are being referenced actively to be stored in memory.
- It allows bigger virtual address spaces and provides the illusion of infinite physical memory.
- It allows more processes to fit in the physical memory, and to be running at the same time.
- Demand paging means that page tables need to sometimes point to the disk locations as opposed to memory locations.
- Page table needs an additional *present* or *valid* bit.

(*i*) If present, the page table entry points to a page in memory.

(*ii*) If not present, a reference to an invalid page ***(page fault)*** will cause the hardware to trap, and the OS performs the following steps while running another process in the meantime.

(a) Choose an old page in memory to replace

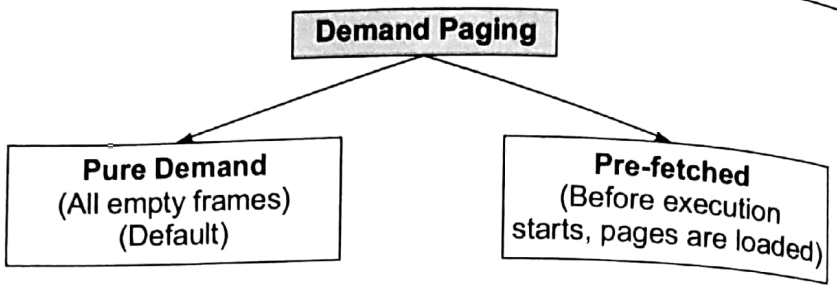
(b) If the old page has been modified, write contents back to disk

(c) Change the corresponding page table entry and TLB entry

(d) Load new page into memory from disk

(e) Update page table entry

(f) Continue the process



Page Replacement

- **Page reference string:** Set of successively unique pages referred in given list of logical address or virtual address.
- **Frame Allocation Policies:** All frames are not allocated, less than maximum demand are allocated.

  Total number of frames = 'm'

  Total number of processes = 'n' $[P_1, P_2, \ldots, P_n]$

  Demand of each process = s_i

  Total demand of all processes = $S = \sum_{i=1}^{n} s_i$ $(s >>> m)$

Note:

- If process requires 'n' frames and operating system allocate 'n' frames to it at once, then there is no need of page replacement.

Page Replacement Algorithms

First-In First-Out (FIFO)

- In general within the increase in page frames the page fault rate of the process should decrease (natural characteristic).
- Sometimes with increase in number of page frames of the process, the page fault rate also increases. This unnatural behavior is called **Belady's Anamoly**.
- FIFO and FIFO based page replacement algorithms (Second Chance/ Clock Algorithm as it degenerates to FIFO) are bound to suffer from Belady's Anamoly.

Optimal

- Replace the page that will not be used for the longest period of time.
- It does not follow Beladys Anamoly.
- It is not implementable. It is used as a Benchmark to measure the performance of other page replacement algorithms.

Least Recently Used (LRU)

- Replace the page that has not been used for the longest period of time.
- Performance of LRU is closer to optimal as it is practical approximation to optimal page replacement.
- Criteria is Time of Reference.

Most Recently Used (MRU)

- Replace the page that is used most recently.
- Criteria is Time of Reference.

Counting Algorithms

- Least Frequently Used (LFU): Replace the page with the smallest count.
- Most Frequently Used (MFU).
- Implementation is expensive.

Variation of LRU/LRU Approximation

- LRU approximation are not exactly LRU, they approximation to the behaviour of LRU (i.e. they work like LRU).

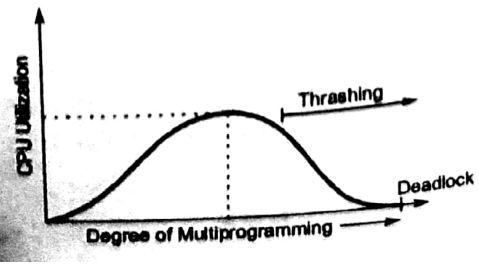
- Reference bit (R)
  - 0 → Page is not referred so far during the current epoch
  - 1 → Page is referred atleast once during the current epoch (a period of time)

- **Second Chance/Clock Algorithm:** It degenerates to FIFO. Criteria is Arrival Time and Reference Bit.
- **Enhanced Second Chance/Not Recently Used:** Criteria is Reference Bit and Modify/Dirty Bit

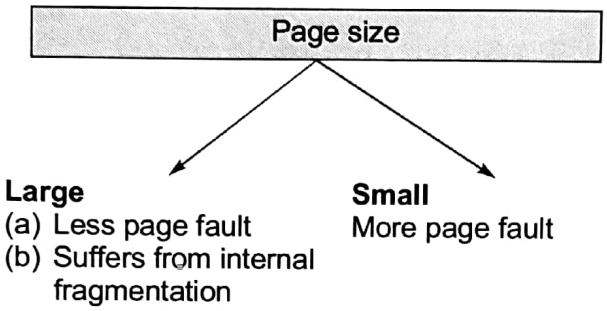
Modify Bit
- 0 → Page is clean
- 1 → Page is modified

Thrashing

- High paging activity is called thrashing. A process is thrashing if it is spending more time paging than executing.



- **Thrashing** occurs when the memory is over committed, and pages are tossed out while still needed.
- CPU utilization is very low when the page fault rate is high.
- Reasons:
  - (*i*) Primary Reasons:
    - (a) Lack of memory frames
    - (b) High degree of multiprogramming
  - (*ii*) Secondary Reasons:
    - (a) Page replacement policy
    - (b)



- Control strategies:
  - (*i*) **Prevention:** Controlling the degree of multiprogramming.
  - (*ii*) Detection:
    - (a) Low CPU utilization
    - (b) High degree of multiprogramming
    - (c) High paging disk utilization
  - (*iii*) Recovery:
    - (a) Process suspension
    - (b) Increase RAM

Inverted PageTable

- An inverted page table has one entry for each real page (or frame) of memory.
- Each entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- Only one page table is in the system and it has only one entry for each page of physical memory.
- Each inverted page table entry is a pair < process-id, page-number> where the process id assumes the role of the address space identifier.

■■■

I/O and File Systems, protection and Security

6

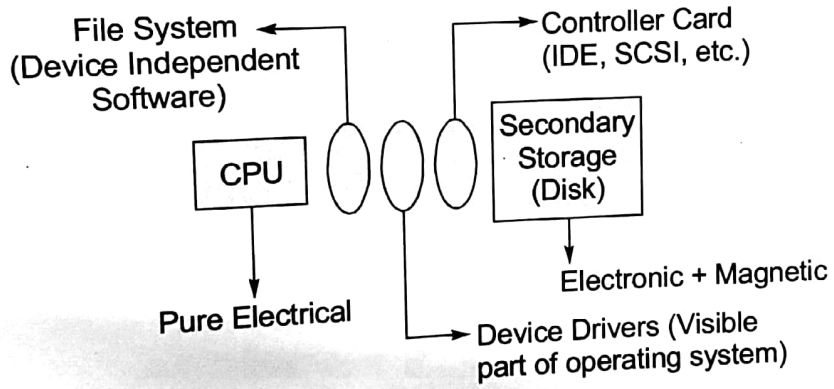
File System

There are three ways to access a file.

1. **Sequential access:** Bytes are accessed in order.
2. **Random access:** Bytes are accessed in any order.

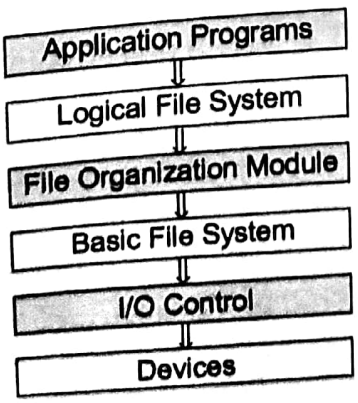
A file system consists of the following major components:

- **Disk management** organises disk blocks into files.
- **Naming** provides file names and directories to users, instead of track and sector numbers.
- **Protection** keeps information secure from other users
- **Reliability** protects information loss due to system crashes.



File System Structure

- A file system is used to allow the data to be stored, located, and retrieved easily.
- The file system itself is generally composed of many different levels:

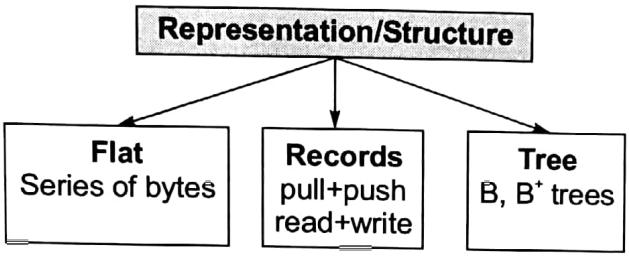


File Usage Patterns

- Most files are small and most file references are to small files.
- Large files use up most of the disk space.
- Large files account for most of the bytes transferred between memory and disk.

File

- A file is logically related records. From programmers perspective file is a DATA STRUCTURE that has definition, representation/structure, operations and attributes.



- **Operations:** Create, Open, Read, Write, Modify, Seek, Copy, Move, Rename, Close, Delete.
- **Attributes:** Name, Type, Size, Owner, Path, Location etc.
- **Directory:** It is also a file which contains information about files. It is a metadata of files.

Logical Structure of Disk

- **Partitions:**
  - (*i*) **Primary/Bootable:** In which operating system and data can be stored. Atleast one primary partition should be there in disk.
  - (*ii*) **Secondary/Non Bootable:** extended (1 or more logical drives)
- **File Control Block (FCB):** It contains information about the file, including ownership, permission, location etc.
  - (*i*) **UNIX:** UNIX FILE SYSTEM (Inode block)
  - (*ii*) **Windows:** FAT, FAT32,WTFS (information within master file table)
  - (*iii*) **Linux:** Extended File System (ext2 and ext3)
- **Allocation Methods:**
  - (*i*) **Contiguous Allocation:**
    - (a) There is internal and external fragmentation
    - (b) Inflexible to increase file size
    - (c) Random access (Faster as it is direct access).

(ii) Linked Allocation:

(a) Each file block on disk is associated with a pointer to the next block.

(b) Flexibility to increase file size

(c) Sequential access so it is slow

(d) Space is consumed for links

(e) Suffers from external fragmentation

(f) MS-DOS file system, which uses the File Attribute Table (FAT) to implement linked -list files

(iii) Indexed Allocation:

(a) Uses an index to directly track the file block locations.

(b) No fragmentation

(c) Flexibility to increase file size

(d) Suffers from wasted space, supports direct access.

(iv) Multilevel Indexed Allocation:

(a) Index entries point to index blocks as opposed to data blocks.

(b) The file header, (*i*_node data structure), holds index pointers.

(c) The first pointers point to data blocks.

(d) The pointer points to a **single indirect block,** which contains additional pointers to data blocks.

(d) The pointer in the file header points to a **double indirect block,** which contains pointers to single indirect blocks.

(e) The pointer points to a **triple indirect block,** which contains pointers to double indirect blocks.

- **Free Space Management:**

(i) BitMap/Bit vector:

(a) A bit vector of size equal to the number of disk blocks is used.

(b) If i^{th} block is free then bit vector value at i^{th} location will be 0 otherwise 1.

(c) Memory requirements for storing bit vector also increases with the increase in the disk size.

(d) Advantages of this approach include: Relatively simple and efficient to find the first free blocks or n-consecutive free blocks on the disk.

(e) To find the fist non-zero value, the calculation of the block number is:

Block Number = (Number of bits per word × Number of zero-value words) + offset of first 1 bit

Consider a disk with 'B' blocks, 'F' free, 'D' – Disk Block Address (DBA), 'S' – Disk Block size in Bytes (DBS).

Actual Disk size = BS

Maximum possible Disk size = $2^D \times S$

$\therefore \boxed{B \le 2^D}$

Condition in which free list uses less space than BitMap.

BitMap ⇒ 'B' bits

Free List → $F \times D$ bits

$$\boxed{FD < B}$$

(*ii*) **Linked list approach:**

(a) It is to link together all the free disk blocks.

(b) It is not easy to traverse the free list.

(*iii*) **Grouping:**

(a) The addresses of n-free blocks are stored in the fist free block.

(b) The first $(n-1)$ of these blocks are actually free.

(c) The last block contains addresses of another n-blocks.

(d) The importance of this approach is that the addresses of a large number of free blocks can be found quickly.

(*iv*) **Counting:** We usually allocate and free a number of contiguous blocks at the same time.

(a) Rather than have the free-space list contain just block numbers.

(b) The free-space list can contain paris (block number, count)

(c) Although each entry is larger, the list, in general, will take less space.

Disk Scheduling → we access cylinder no.

First-come First-serve (FCS)

- Requests are served in the order of arrival.
- This policy is fair among requesters, but requests may land on random spots on disk.
- The seek time may be long.
- It does not provide the fastest service.

Shortest Seek Time First (SSTF)

- It picks the request that is closest to the current disk arm position.
- Selects the request with minimum seek time from the current head position

- It includes the rotational delay in calculation, since rotation can be as long as seek.
- SSTF is good at reducing seeks, but may result in starvation.

SCAN (Elevator)

- It takes the closet request in the direction of travel.
- The head continuously scans back and forth across the disk.
- It guarantees no starvation, but retains the flavor of SSTF.

LOOK

- Read write head exactly stops at the last head i.e., look at last readwrite (r/w) operation and go back.

C-SCAN

- The disk arm always serves requests by scanning in one direction.
- Once the arm finishes scanning for one direction, it quickly returns to the 0^{th} track for the next round of scanning.
- Head is imagined to move in circular motion (which is not practically possible).

Disk Access Time

The disk access time to read or write a disk section includes three components.

- **Seek time:** The time to position heads over a cylinder.
- **Rotational delay:** The time to wait for the target sector to rotate underneath the head.
- **Transfer time:** The time to transfer bytes.
- Disk access time = Seek time + Rotational delay + Transfer time.

Protection and Security

- **Protection** refers to the actual mechanisms implemented to enforce the specified policy.
- **Security** refers to the policy of authorising accesses. Security aims to prevent intentional misuses of a system, while protection aims to prevent either accidental or intentional misuses.
- A secure system tries to accomplish three goals:
  - (*i*) **Data confidentiality:** secret data remains secret.
  - (*ii*) **Data integrity:** unauthorised users should not be able to modify any data without the owner's permission.
  - (*iii*) **System availability:** nobody can disturb the system to make it unusable.

- There are three components of security:
  (*i*) **Authentication** determines who the user is.
  (*ii*) **Authorization** determines who is allowed to do what.
  (*iii*) **Enforcement** makes sure that people do only what they are supposed to do.

Security Attacks

Eavesdropping

- **Eavesdropping** is the listener approach.
- One can tap into the serial line on the Ethernet, and see everything typed in; almost everything goes over network unencrypted.

Abuse of Privilege

If the superuser is evil, there is nothing you can do.

Imposter

- An imposter breaks into the system by pretending to be someone else.
- A countermeasure against the imposter attack is to use **behavioral monitoring** to look for suspicious activates.

Trojan Horse

- A **Trojan horse** is a seemingly innocent program that contains code that will perform an unexpected and undesirable function.
- A countermeasure against the Trojan horse is **integrity checking.**

Salami Attack

- The idea is to build up a chunk one-bit at a time.
- A countermeasure is for companies to have ***code*** **reviews** as a standard practice.

Logic Bombs

- A programmer may secretly insert a piece of code into the production system.
- A countermeasure is to have **code reviews.**

Denial-of-service Attack

- Denial-of-service attacks refer to attacks on system availability.
- A handful of compromised machines can flood a victim machine with network packets to disrupt its normal use.

■■■