Name : Aniruddha Shivaji Repale
Roll No : TEIT-10

DPU

OS : Class Assignment - 3

Que.1] What conditions are generally associated with reader-writer problems? Write its pseudo code.

⇒ Ans :

i) The readers-writers problem is a classical synchronization problem in concurrent programming, often encountered when multiple threads or processes access a shared resource (like file or database).

ii) The problem occurs when we have :-

- Readers (processes that only read data) : Multiple readers can read the shared resource simultaneously since reading does not modify the data.
- Writers (processes that modify data) : A writer must have exclusive access to shared resource, meaning no other writer or reader should access the resource while writer is working.

iii) conditions associated with reader-writer problem :-

1] Readers-preferred problem

- Multiple readers are allowed to access the shared resource simultaneously
- writers are given low priority.
- If there is an ongoing read, writers may be forced to wait even when no new readers arrive, leading to potential writer starvation.

2] Writers-preferred problem

- writers are given higher priority.
- once a writer is ready, it prevents further readers from starting, but allows already reading processes to complete.
- This can lead to reader starvation if writers are continuously arriving.

3] Fair solution

- Neither readers nor writers are favored.
- Ensures no starvation of readers or writers.
- Readers & writers are handled in fair queing mechanism.

iv) Pseudo code for Readers-Preferred Solution :-

variables :

```
read_count = 0
mutex = Semaphore(1)
```

Program :

```
int readcount ;
semaphore x = 1 , wsem = 1;
void reader()
{
    while (true) {
        semWait (x);
        readcount ++;
        if (readcount == 1)
            semWait (wsem);
        semSignal(x) ;
        READUNIT();
        semWait (x);
        readcount ;
        if (readcount == 0)
            semSignal (wsem);
        semSignal (x);
    }
}
void writer() {
    while(true){
        semwait (wsem);
        WRITEUNIT();
        semSignal (wsem); }}

void main() {
    readcount=0;
    Parbegin (reader, writer);
}
```

Que.2] Describe Resource Allocation graph in detail.

⇒ Ans :

i) Resource allocation graph is introduced by Holt.

ii) It is a directed graph that depicts a state of system of resources and processes.

iii) It represents the allocation of resources to processes in a system, primarily used to model and understand deadlocks in OS.

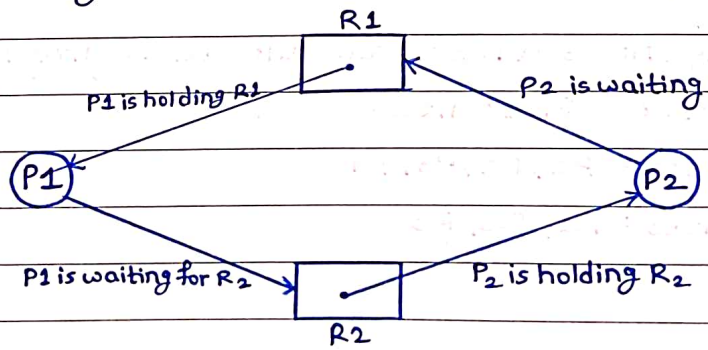
iv) It is a graphical tool that helps in analyzing the state of resource allocation & determining weather a deadlock situation can occur.

v) Components of RAG :-

1] Processes (P1, P2, ...)
- Represented by circles.
- Each process is a distinct running task that may request & hold resources.

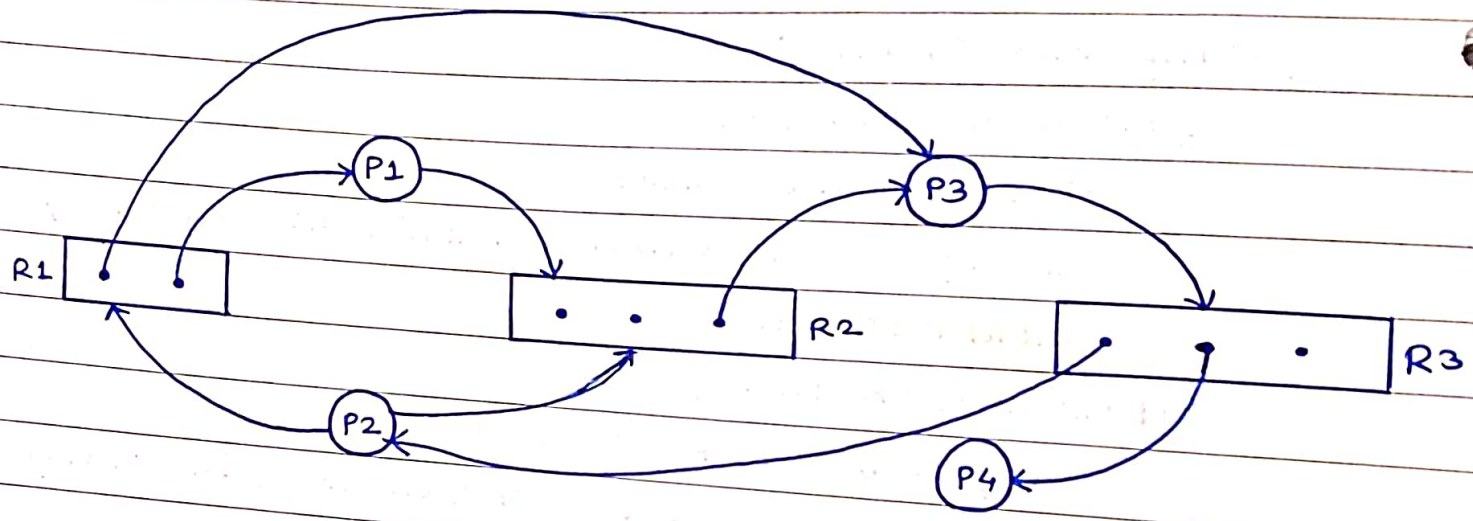
2] Resources (R1, R2, ...)
- Represented by Rectangles.
- Each resource can be a CPU, memory, file or any other type of resources that process may need.
- Each resource can contain multiple instances (represented as dots inside rectangle), or only one instance.

3] Edges
- Request edge : Directed edge from process to resource. It signifies that process is requesting the resource.
- Assignment edge : Directed edge from resource to process. It signifies that resource is currently allocated to process.



vi) If there is a cycle in the resource allocation graph, it indicates a potential deadlock. In the above example, P1 is waiting for R2, which is held by P2 & P2 is waiting for R1, which is held by P1. This forms a cycle, indicating deadlock.

vii) Example 2 :



Resources	No. of instances
R1	2
R2	3
R3	3

Above fig. shows a resource allocation graph. System consists of processes & resources.

Processes : P1, P2, P3, P4

Resources : R1, R2, R3

Que.3] Enlist different IPC techniques. Differentiate between named & unnamed pipe with suitable examples.

⇒ Ans :

i) Inter-Process communication (IPC) is the mechanism by which processes can communicate and synchronize their actions when running concurrently in an OS.

ii) Various IPC techniques are used to exchange data between processes. Here are some common IPC techniques :-

1] Shared memory

- It is a block of memory that can be shared by multiple process
- It permits processes to communicate by simply reading & writing to a specified memory location.
- It is fastest IPC mechanism since processes can access the memory directly without overhead of system calls.

2] Memory Mapped Files

- It is similar to shared memory, except that it is associated with a file in file system.
- Files are mapped into the process's virtual address space.
- This allows multiple processes to share data through files without explicitly reading or writing to files.

3] Pipes

- Pipe is a unidirectional communication channel that allows processes to communicate. It is FIFO, unstructured data stream of fixed max. size.
- writers add data to end of pipe & readers retrieve data from the front of the pipe.
- once read, the data is removed from pipe & is unavailable to other readers.
- Unnamed pipes : can be used only for communication bet processes that have a common ancestry.
- Named pipes (FIFOs) : can be used for communication bet unrelated processes.

4] Sockets
- Used for communication betⁿ unrelated processes even on different computers.
- Sockets provide bidirectional communication and are often used in client server architectures.

5] Message Queues
- Allows processes to send & receive msgs in a queue format.
- Messages are stored in queue until receiving process retrieves them.
- Unlike pipes, message queues allow msgs to be read in any order (not just FIFO).

6] Signals
- A limited form of IPC used to notify a process of an event.
- Signals are used to interrupt the execution of process & handle specific events like termination or completion of I/o operation

iii) Differences :-

Unnamed Pipe	Named Pipe
i) created using the pipe() system call.	i) created using mkfifo() or mknod() commands.
ii) Used for communication betⁿ processes that have common ancestor. (i.e. related processes)	ii) Used for communication betⁿ unrelated processes, even if they don't have common ancestry.
iii) Data exists only while processes are communicating.	iii) Data remains in pipe (FIFO) until read by a process.
iv) Identified by file descriptors	iv) Identified by a name in file system.

TEIT-10

Class Assignment –

DPU

Que.4] What is critical section Problem? Give a semaphore solution for producer consumer problems.

⟹ Ans :

i) A critical section is a block of code that only one process at a time can execute; so, when one process is in its critical section, no other process can be in its critical section.

ii) Critical section problem refers to the problem of ensuring that multiple processes or threads can safely access shared resources without causing inconsistencies.

iii) A critical section is a portion of program where the shared resource is accessed.

iv) The critical section problem focuses on designing protocols to manage entry into and exit from critical section, so that :-

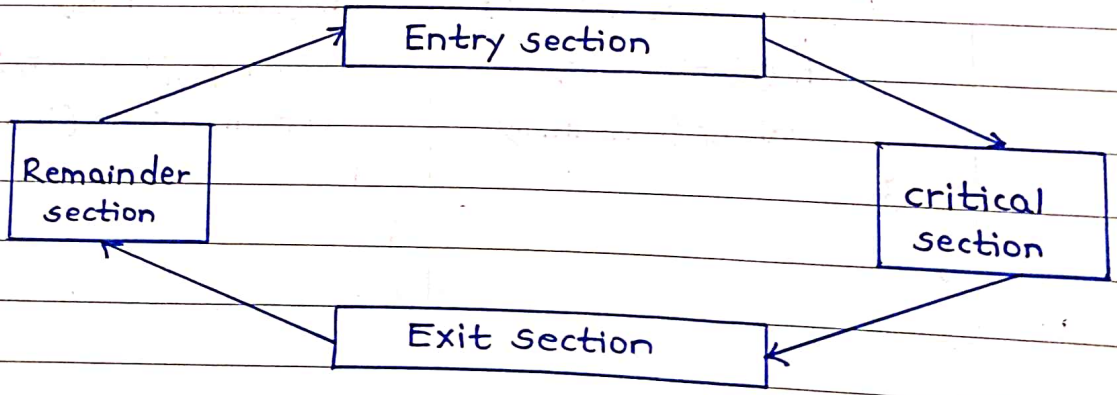
1] Mutual Exclusion : only one process is allowed in critical section at a time.

2] Process : If no process is in critical section, the next process that requests entry must be allowed in finite amount of time.

3] Bounded waiting : A process should not have to wait indefinitely to enter its critical section.

v) Each process takes permission from OS to enter into critical section.

- Entry section : It is block of code executed in preparation for entering critical section.
- Exit section : The code executed upon leaving critical section.
- Remainder section : Rest of the code is remainder section.



vi) Producer-Consumer Problem :-

- It is a classic synchronization problem where two types of processes, producers & consumers, share a fixed size buffer.
- The producer generates data & puts it into buffer & the consumer takes data from buffer & processes it.
- The problem arises when producer tries to add data to a full buffer, or consumer tries to take data from empty buffer. These two actions must be synchronized to avoid race conditions.
- In order to synchronize these processes, both producer & consumer are blocked on some condition. The producer is blocked when buffer is full & consumer is blocked when buffer is empty.

code for Producer :	code for consumer :

producer (void)
{
int item;
while (TRUE)
{
produce item (&item);
produce_if (counter == N)
sleep ();
enter_item (item);
counter = counter + 1;
if (counter = = 1)
wakeup (consumer)
}
}
	consumer (void)
	{
	int item;
	while (TRUE)
	{
	if (count == 0)
	sleep ();
	remove_item (&item);
	counter = counter - 1;
	if (count = = N-1)
	wakeup (producer);
	consume_item (item);
	}
	}

DPU

Que.5] What is Semaphore and mutex? Explain with the help of the pseudo code, how semaphore is used to solve producer consumer problems?

⇒ Ans:

i) A Semaphore is a synchronization primitive that helps control access to a shared resource by multiple processes or threads in concurrent system. It can be seen as a signaling mechanism, typically used to prevent race conditions in shared data.

ii) A semaphore has an integer value & two atomic operations are performed on it :-

1] wait()
- If semaphore value > 0, it decrements value & process proceeds
- If semaphore value ≤ 0, process is blocked until value becomes +ve

2] signal()
- Increments semaphore value.
- If any process was blocked due to semaphore being 0, one of the blocked processes is unblocked & can proceed.

iii) Semaphores come in two types :-

1] Binary : Takes values 0 and 1

2] counting : can take non-negative int values, used to control access to resources with multiple instances.

iv) A Mutex is a binary semaphore that specifically ensures mutual exclusion. It allows only one process or thread to enter the critical section at a time. It can be locked or unlocked.

- Lock (wait) : If no other process has locked the mutex, the process locks it & enters critical section. If mutex is already locked, process waits.
- Unlock (signal) : The process exits the critical section and unlocks the mutex, allowing another process to enter.

v) Semaphore solution for Producer-Consumer Problem:

```
void main()
{
   count = 0;
}
int i;
binary semaphore s=1;
int producer()
{
   while(true)
   {
     produce_data_item();
     semwaitB(s);
     append();
     count= count+1;
     if(count==1)
      semSignalB(delay);
      semsignalB(s);
   }
}
int consumer()
{
   int p;
   sem waitB(delay);
   while(true)
   {
     semwaitB(s);
     consume();
     count= count-1;
     p=count;
     semSignalB(s); consumedata();
     if(p==0)
       semwaitB(delay);
   }
}
```

- Process 1 (200 KB)

Partition 1 (100 KB)

DPU

What are four necessary conditions for deadlock ? How is a deadlock detected in a system with resources having single instances ? Explain with an example.

Ans :

i) A deadlock is a situation where a set of processes are blocked because each process is holding a resource & waiting for another resource that is held by another process.

ii) Following four conditions are necessary for deadlock to exist :-

1] Mutual exclusion :

- only one process can use the resource at a time. If another process requests that resource, the requesting process must be blocked until resource is released.
- e.g. A printer can only be used by one process at a time.

2] Hold and wait :

- A process must be holding at least one resource & waiting to acquire additional resources that are currently held by another processes.
- e.g. A process is holding a printer and waiting for scanner & another process holds scanner & is waiting for printer.

3] No preemption :

- Resources cannot be forcibly taken away from a process holding them.
- eg. A process using a printer cannot be interrupted to forcibly give printer to another process.

4] circular wait :

There must exist a set of processes $\{P_1, P_2, \ldots, P_n\}$ such that P_1 is waiting for resource held by P_2, P_2 is waiting for the resource held by P_3, ... & so on, with P_n waiting for resource held by P_1, forming a cycle.

iii) Steps to Detect Deadlock :-

1] Construct a Resource Allocation Graph.

- Draw a directed graph where:
- Processes are represented as circular nodes.
- Resources are represented as square nodes.
- Arrows represents allocation or requests for resources.

2] check for cycles.

- A deadlock exists iff there is a cycle in graph. A cycle indicates a circular wait condition, which is necessary condition for deadlock.

Que. 7] Define mutual exclusion, race condition, semaphore, deadlock.

⇒ Ans :

i) Mutual exclusion

- It is a principle used in concurrent programming to prevent multiple processes or threads from accessing a shared resource simultaneously.
- It ensures that only one process can enter critical section at a time.
- e.g. In a printer queue, only one process can print at a time to avoid overlapping output.

ii) Race condition

- It occurs when outcome of program depends on sequence of & timing of uncontrollable events, such as concurrent execution of multiple threads or processes.
- If two or more processes or threads access shared data concurrently & the final outcome depends on order of execution, a race condition may result, often leading to incorrect results.
- e.g. If two threads simultaneously try to update bank account balance, the final balance might be incorrect depending on which thread completes its operation first.

iii) Semaphore

- It is a synchronization tool used to control access to a shared resource in concurrent system.
- It is essentially a counter that can be used to signal & wait on events, helping manage access to shared resources.

iv) Deadlock

- It is a situation in multiprogramming environment where two or more processes are unable to proceed because each process is waiting for a resource that is held by another process.
- For a deadlock to occur, four conditions may be satisfied: Mutual exclusion, Hold & wait, No preemption, circular wait.

Que.8] What is Bankers safe sequence algorithm ? Apply it for finding safe sequence of execution of 5 processes in system having snapshot at time T0. Also determine whether following requests can be granted or not.

i) Request for process P2 : 3 0 0

ii) Request for process P3 : 0 0 1

⇒ Ans :

i) The Banker's Algo. is used to avoid deadlocks by ensuring that system will remain in safe state after resource allocation. The idea is to evaluate whether or not granting a resource request will lead the system into an unsafe state, causing deadlock.

ii) A system is in a safe state if there is sequence of processes such that each process can finish its execution by getting the required resources & releasing them after use. This sequence is called safe sequence.

iii) Steps for finding safe sequence:

1] Need Matrix calculation

Need [P][R] = Max [P][R] - Allocation [P][R]

This shows remaining resources that each process may request

2] Check if system is in safe state:

- start by checking if the available resources can satisfy the needs of any process. If they can, that process can finish, release its resources & make them available to the remaining processes.
- continue this process until all processes are able to finish. If no such sequence exists, system is in unsafe state.

Given system of processes & resources is :

Process	Allocation			Max			Available			Need		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	3	2	2	7	4	3
P1	2	0	0	3	2	2	5	2	2	1	2	2
P2	3	0	2	9	0	2	7	3	3	6	0	0
P3	2	1	1	2	2	2	10	3	5	0	1	1
P4	0	0	2	4	3	3	10	4	7	4	3	1
							10					

∴ safe sequence is: < P1 – P3 – P4 – P2 – P0 >

ii) Request for process P2 : 3 0 0

⇒ Request (3,0,0) < Available (3,2,2), then continue;

Request (3,0,0) < Need (6,0,0), then continue;

Update data structures as if request is granted.

Process	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	7	4	3	3	2	2
P1	2	0	0	3	2	2	1	2	2	5	2	2
P2	6	0	2	9	0	2	3	0	0	11	2	4
P3	2	1	1	2	2	2	0	1	1	13	3	5
P4	0	0	2	4	3	3	4	3	1	13	3	57

}

Apply the safety algorithm:

- Set work (3,2,2) & Finish (F,F,F,F,F). Search for process that can terminate
- P1 can terminate : work = (5,2,2) & Finish = (F,T,F,F,F) ; continue search.
- P2 can terminate : work = (11,2,4) & Finish = (F,T,T,F,F) ; continue search.
- P3 can terminate : work = (13,3,5) & Finish = (F,T,T,T,F) ; continue search.
- P4 can terminate : work = (13,3,7) & Finish = (F,T,T,T,T) ; continue search.
- But, now P0 still can't terminate. ∴ Finish = (F,T,T,T,T)

Since, Finish contains a process that haven't terminated, the above state is unsafe & request by P2 for resources is not granted – P2 must wait.

v) Request for Process P3 : 0 0 1

⇒ Request (0,0,1) < Available (3,2,2) ; then continue.

Request (0,0,1) < Need (0,1,1) ; then continue.

Update data structures as if request is granted.

Process	Allocation			Max			Need			Available		
	A	B	C	A	B	C	A	B	C	A	B	C
P0	0	1	0	7	5	3	7	4	3	3	2	2
P1	2	0	0	3	2	2	1	2	2	5	2	2
P2	3	0	2	9	0	2	6	0	0	7	3	4
P3	2	1	2	2	2	2	0	1	0	10	3	6
P4	0	0	2	4	3	3	4	3	1	10	3	8

As like above request of P2, here also process P0 still can't terminate.

∴ Above state is unsafe & request by P3 is not granted – P3 must wait.

10/10/24

*OS : Class Assignment - 4 *

Que. 1] Consider six memory partitions of size 100 kB, 300 kB, 50 kB, 200 kB, 150 kB and 200 kB. These partitions need to be allocated to the processes of sizes 200 kB, 100 kB, 50 kB in that order. Perform the allocation of processes using dynamic partitioning algorithms given below & comment on internal & external fragmentation.

i) First Fit Algo. , ii) Best Fit Algo. , iii) worst Fit Algo.

⇒ Ans :

(i) First Fit Algorithm :

- This algo. scans memory partitions from the beginning & allocates the first partition that is large enough to fit the process.
- Process 1 (200 kB) : Allocated to partition 2 (300 kB). 100 kB leftover space. (~~no~~ internal fragmentation).
- Process 2 (100 kB) : Allocated to partition 1 (100 kB). No leftover space (no internal fragmentation).
- Process 3 (50 kB) : Allocated to partition 3 (50 kB). No leftover space (no internal fragmentation).
- Unallocated partitions : Partition 4 (200 kB), Partition 5 (150 kB), Partition 6 (200 kB).
- Fragmentation :-
  - Internal : ~~None~~ 100 kB ~~All~~ Rest two partitions fit their respective process perfectly.
  - External : 550 kB (unallocated space).
- First Fit performs well in terms of internal fragmentation but it leaves significant external fragmentation due to large chunks of memory that remain unused.

(ii) Best Fit Algorithm :

- This algo. scans all portions (partitions) and allocates the process to the smallest partition that can fit the process, aiming to minimize leftover space.
- Process 1 (200 kB) – Partition 4 (200 kB)

  Process 2 (100 kB) – Partition 1 (100 kB)

  Process 3 (50 kB) – Partition 3 (50 kB)

- Unallocated partitions : Partition 2 (300 kB), Partition 5 (150 kB), Partition 6 (200 kB).
- Fragmentation :-
  - Internal : None, processes fit perfectly into their respective partit
  - External : 650 kB (unallocated space).

(iii) Worst Fit Algorithm :

- This algo. allocates the processes to the largest available portion. The goal is to leave largest remaining hole after each allocation.
- Process 1 (200 kB) : Allocated to partition 2 (300 kB). 100 kB remai unused. (internal fragmentation).
- Process 2 (100 kB) : Allocated to partition 4 (200 kB). 100 kB remain unused. (internal fragmentation).
- Process 3 (50 kB) : Allocated to partition 6 (200 kB). 150 kB remains unused. (Internal fragmentation).
- Unallocated : Partion 1 (100 kB), Partition 3 (50 kB), Partition 5 (150
- Fragmentation :-
  - Internal : 350 kB
  - External : 300 kB (unallocated space).
- Worst Fit increases internal fragmentation because it leaves large unused portions of memory inside allocated partitions.

(iv) Summary :

Algorithm	Internal fragmentation	External fragmentation
First fit	100 kB	550 kB
Best fit	None	650 kB
worst fit	350 kB	300 kB

✓ Conclusion :

- Internal frag. occurs when allocated memory is larger than required by process.
- External frag. occurs when there is enough total free memory, but it is scattered across multiple small partitions, making it unusable for large processes.

DPU

Que.2] ⇒ Explain Buddy system memory allocation with examples.

Ans :

i) The Buddy system is a memory allocation algorithm designed to manage dynamic memory in an efficient manner while minimizing fragmentation.

ii) It is particularly used for systems where memory requests are varied in size & for managing free memory blocks effectively.

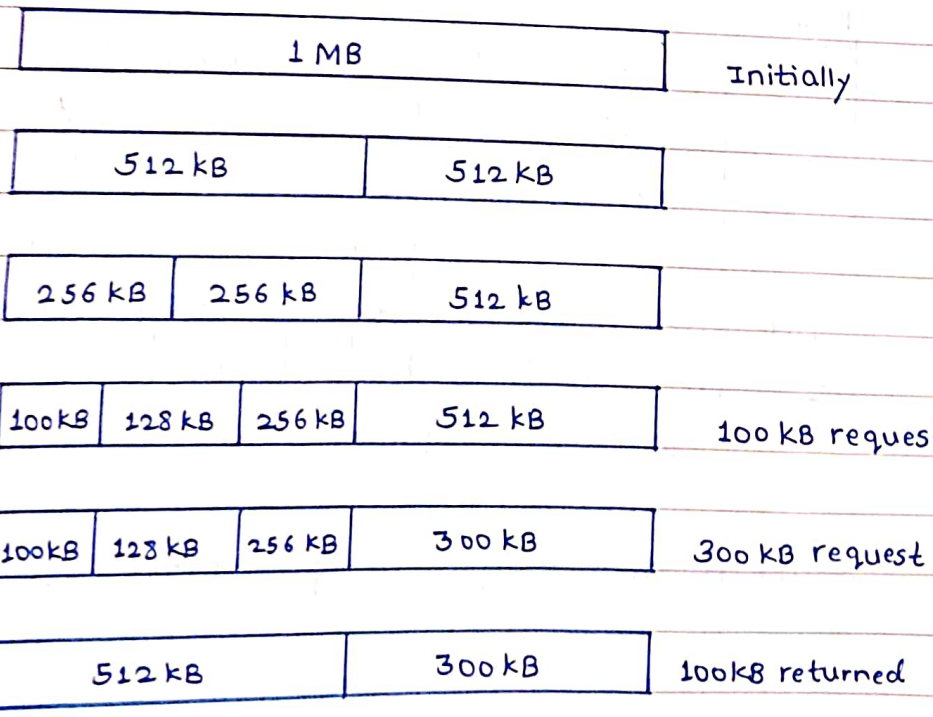
iii) Memory is divided into blocks of sizes that are powers of 2 (e.g. 16 kB, 32 kB, 64 kB, etc..). This allows system to maintain balance bet^n efficient memory use and ease of allocation & deallocation.

iv) Each memory block can be split into two buddies of equal size. when a block is allocated or freed, the system checks whether its buddy (the adjacent block of same size) is also free. If both buddies are free, they can be merged back into a larger block, thus reducing the fragmentation.

v) when a request for memory is made, system searches for smallest block that is large enough to fulfill the request. If a block is too large, it is split into two equal halves until right size is found.

vi) when a block is freed, system checks if its buddy is also free. If so, they are merged.

vii) e.g.



... no deadlock

viii) Advantages :-

1] Easy to implement.
2] Allocates block of correct size.
3] Easy to merge adjacent holes.
4] Fast to allocate & deallocate memory.

ix) Disadvantages :-

1] It requires all allocation unit to be powers of two.
2] It leads to internal fragmentation.
3] Merging of holes is recursive so, poor worst-case behaviour.

Que.3] Find the no.of page faults for the reference string given below 6,5,1,2,5,3,5,4,2,3,6,3,2,1,2 with frame size of 3, using FIFO, LRU & OPR page replacement strategies.

⇒ Ans :

① First In First Out (FIFO) :

String / Frames	6	5	1	2	5	3	5	4	2	3	6	3	2	1	2
1	6	6	6	2	2	2	2	4	4	4	6	6	6	1	1
2	–	5	5	5	5	3	3	3	2	2	2	2	2	2	2
3	–	–	1	1	1	1	5	5	5	3	3	3	3	3	3
Page-Fault	PF	PF	PF	PF		PF	PF	PF	PF	PF	PF			PF	

Total no.of Page Faults = 11

In FIFO, the oldest page is replaced first when page fault occurs.

DPU

(ii) Least Recently Used (LRU):

In LRU, page that has not been used for longest time is replaced.

	6	5	1	2	5	3	5	4	2	3	6	3	2	1	2
1	6	6	6	2	2	2	2	4	4	4	6	6	6	1	1
2	-	5	5	5	5	5	5	5	5	3	3	3	3	3	3
3	-	-	1	1	1	3	3	3	2	2	2	2	2	2	2
Page Fault	PF	PF	PF	PF		PF		PF	PF	PF	PF			PF	

∴ Total no. of page faults = 10

(iii) Optimal Page Replacement (OPR):

In OPR, we replace the page that will not be used for longest period in future.

	6	5	1	2	5	3	5	4	2	3	6	3	2	1	2
1	6	6	6	6	6	3	3	3	3	3	3	3	3	1	1
2	-	5	5	5	5	5	5	4	4	4	6	6	6	6	6
3	-	-	1	2	2	2	2	2	2	2	2	2	2	2	2
Page Fault	PF	PF	PF	PF		PF		PF			PF			PF	

∴ Total no. of page faults = 8

Que. 4] Explain Belady's anomaly with suitable examples.

⇒ Ans:

i) Belady's Anomaly is a surprising situation in computer memory management where increasing the no. of page frames (the memory space allocated for pages) can actually lead to more page faults.

ii) This goes against common expectation that more memory should reduce page faults.

iii) A page fault happens when a prog. tries to access a page of memory that is not currently loaded into RAM. When this occurs, OS must bring required page into memory, which can cause delays.

iv) This anomaly primarily occurs with certain page replacement algo. like FIFO.

v) Example:

Reference string : 1, 2, 3, 1, 2, 4, 1, 2, 5, 1, 2, 3, 4, 5

Case.1] 3 page frames:

	1	2	3	1	2	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	4	4	4	5	5	5	5	5	5
2	-	2	2	2	2	2	1	1	1	1	1	3	3	3
3	-	-	3	3	3	3	3	2	2	2	2	2	4	4
Page Fault	PF	PF	PF			PF	PF	PF	PF			PF	PF	

∴ Total no. of page faults = 9

Case.2] 4 page frames

(same string)

	1	2	3	1	2	4	1	2	5	1	2	3	4	5
1	1	1	1	1	1	1	1	1	5	5	5	5	4	4
2	-	2	2	2	2	2	2	2	2	1	1	1	1	5
3	-	-	3	3	3	3	3	3	3	3	2	2	2	2
4	-	-	-	-	-	4	4	4	4	4	4	3	3	3
Page Fault	PF	PF	PF			PF			PF	PF	PF	PF	PF	PF

∴ Total no. of page faults = 10

vi) Thus, when we increased the no. of page frames from 3 to 4, no. of page faults increased.
This behaviour is what we call Belady's Anomaly.

DPU

Que.5] Given memory partitions of 150kB, 650 kB, 280 kB, 390 kB & 540 kB, how would each of the First Fit, Best Fit & Worst Fit Algorithms place processes of 212 kB, 457 kB, 112 kB, 510 kB & 326 kB (in order)?

⇒ Ans:

(i) First Fit :-

- Process 212 kB : Allocated to Partition 2 (650 kB).
  Remaining partition sizes : 150 kB, 438 kB (650-212), 280 kB, 390 kB, 540 kB
- Process 457 kB : Allocated to Partition 5 (540 kB).
  Remaining partition sizes : 150 kB, 438 kB, 280 kB, 390 kB, 83 kB (540-457)
- Process 112 kB : Allocated to Partition 1 (150 kB).
  Remaining Partition sizes : 38 kB (150-112), 438 kB, 280 kB, 390 kB, 83 kB.
- Process 510 kB : cannot be allocated. (∴ wait).
- Process 326 kB : Allocated to partition 2 (438 kB).
  Remaining Partition sizes : 38 kB, 112 kB (438-326), 280 kB, 390 kB, 83 kB.

(ii) Best Fit :-

- Process 212 kB : Allocated to partition 3 (280 kB)
  Remaining partitions : 150 kB, 650 kB, 68 kB (280-212), 390 kB, 540 kB
- Process 457 kB : Allocated to partition 5 (540 kB)
  Remaining partitions : 150 kB, 650 kB, 68 kB, 390 kB, 83 kB (540-457)
- Process 112 kB : Allocated to partition 1 (150 kB)
  Remaining partitions : 38 kB (150-112), 650 kB, 68 kB, 390 kB, 83 kB.
- Process 510 kB : Allocated to partition 2 (650 kB)
  Remaining partitions : 38 kB, 140 kB (650-510), 68 kB, 390 kB, 83 kB.
- Process 326 kB : Allocated to partition 4 (390 kB)
  Remaining partitions : 38 kB, 140 kB, 68 kB, 64 kB (390-326), 83 kB.

(iii) Worst Fit :-

- Process 212 kB : Allocated to Partition 2 (650 kB).
  Remaining partitions : 150 kB, 438 kB, 280 kB, 390 kB, 540 kB.

- Process 457 kB : Allocated to partition 5 (540 kB)
  Remaining partitions : 150 kB, 438 kB, 280 kB, 390 kB, 83 kB.
- Process 112 kB : Allocated to partition 42 (438 kB)
  Remaining partitions : 150 kB, (438-112) 326 kB, 280 kB, 390 kB, 83 kB.
- Process 510 kB : cannot fit any remaining partition (∴ wait).
- Process 326 kB : Allocated to partition 4 (390 kB)
  Remaining partitions : 150 kB, 326 kB, 280 kB, 64 kB, 83 kB.

Que.6] Consider following segment table.

Segment	Base	Length
0	1790	350
1	2722	1050
2	520	925
3	5200	450
4	4200	655

What are physical addresses for following logical addresses?
i) 0, 330 ii) 2, 525 iii) 4, 700 iv) 3, 400 v) 1, 1110

⇒ Ans :

The logical addresses are given as pairs (segment number, offset).
The physical address is calculated using formula :-

Physical Address = Base Address + offset

where,

offset should be within segment length (given), to ensure the address is valid.

① Logical Address: (0, 330)
- Segment 0 : Base = 1790, Length = 350
- offset : 330 (valid since 330 < 350)
- ∴ Physical address = (1790 + 330) = 2120

DPU

(ii) Logical Address : (2,525)

- Segment 2 : Base = 5200, Length = 925
- offset : 525 (valid since 525 < 925)
- ∴ Physical address = (5200 + 525) = 1045

(iii) Logical Address : (4,700)

- Segment 4 : Base = 4200, Length = 655
- offset : 700 (invalid since 700 > 655)
- ∴ Invalid address.

(iv) Logical Address : (3,400)

- Segment 3 : Base = 5200, Length = 450
- offset : 400 (valid since 400 < 450)
- ∴ Physical address = (5200 + 400) = 5600

(v) Logical Address : (1,1110)

- Segment 1 : Base = ~~1790~~ 2722, Length = 350 1050
- offset : 1110 (~~invalid~~ since 1110 > ~~350~~ 1050)
- ∴ Invalid address.

Que.7] What are the distinctions among logical, relative & physical addresses?

⇒ Ans :

i) Logical Address :

- A logical address (virtual address) refers to an address generated by CPU during program execution. This address is a part of program's virtual address space.
- OS uses logical address to provide each process with its own independent address space, allowing for easier management of memory.
- They are translated to physical addresses by Memory Management Unit (MMU), which uses base address & offset to perform this translation.

DPU

ii) Relative Address :

- It specifies location in memory relative to certain point. It doesn't specify an absolute memory location but is offset from base address.
- Typically used in segment based or paging systems.
- It is translated to physical address by adding base address to it.

iii) Physical Address :

- It is the actual location in computer's memory (RAM). It corresponds to a specific location in physical memory hardware.
- The OS maintains data structures (like page tables or segment tables) that map logical addresses to physical addresses to (map) manage memory access effectively.

Logical address	Relative address	Physical address
i) Address generated by CPU, referring to a location in program's virtual memory.	i) offset from a base address, indicating a location within a segment or page.	i) Actual address in physical memory (RAM) where the data is stored.
ii) Used by programs during execution.	ii) Used in memory management systems.	ii) Used by the memory hardware.
iii) Unique per process.	iii) Dependent on base addre	iii) Global across system.
iv) Physical address = Base address + offset.	iv) Physical address = Base address + Relative address.	iv) The address itself, used directly for data access.
v) Provides process isolation & security.	v) Less isolation (as it depends on base address)	v) No isolation; directly corresponds to main memory.

Name : Aniruddha Shivaji Repale.

Roll No.: TEIT-10

DPU

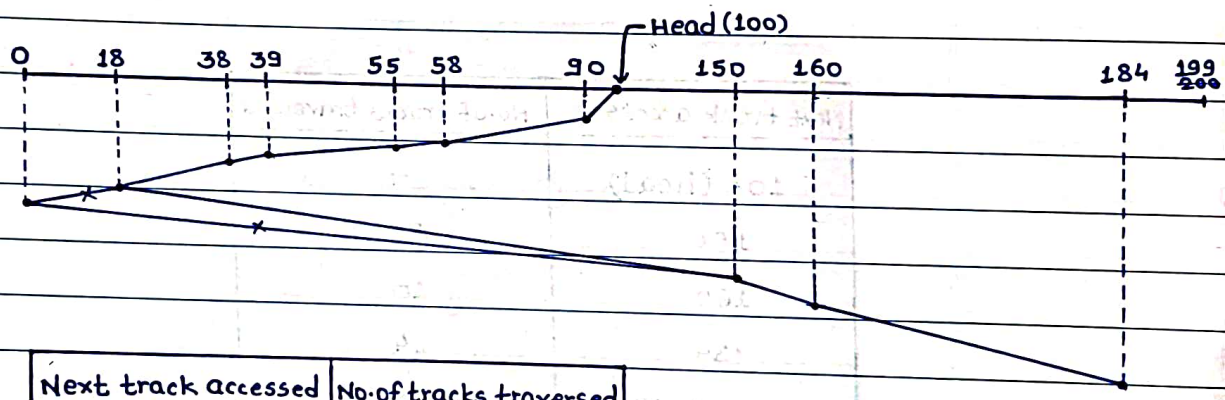
✳ OS : Class Assignment – 5 ✳

Que.1] Assume a disk with 200 tracks and the disk request queue has random requests in it as follows : 55, 58, 39, 18, 90, 160, 150, 38, 184. Find the number of tracks traversed and avg. seek length if 1) SSTF 2) SCAN 3) C-SCAN is used & initially head is at track number 100.

⇒ **Ans :**

1] SSTF (Shortest Seek Time First) :

- This algo. serves the request closest to current head position.
- Starting at track 100, following is the order of requests served using SSTF :



Next track accessed	No. of tracks traversed
100 (head)	–
90	10
58	32
55	03
39	16
38	01
18	20
150	132
160	10
184	24

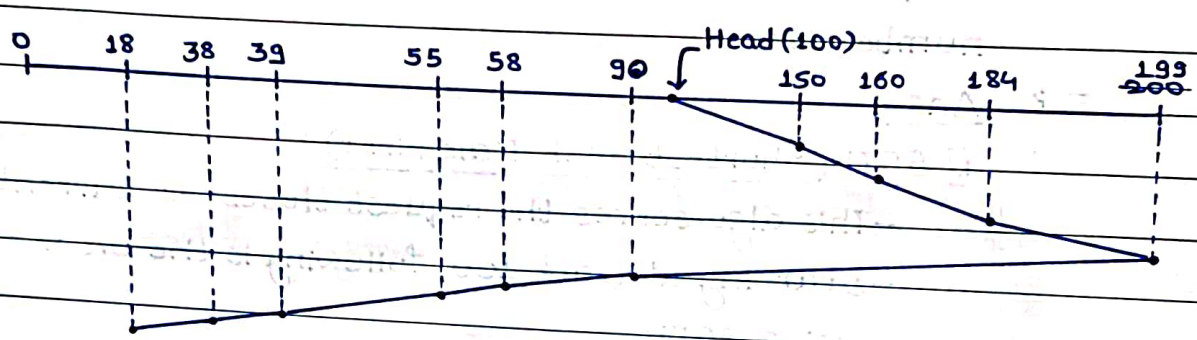
∴ Total tracks traversed : 10+32+03+16+01+20+132+10+24 = 248

∴ Avg. seek length = $\frac{248}{9}$ = 27.55

($\frac{\text{tracks traversed}}{\text{requests received}}$)

2] SCAN :

-This algo. moves the head in one direction and services requests until it reaches the end of the disk & then reverses direction serving consequent requests.



Next track accessed	No. of tracks traversed
100 (head)	–
150	50
160	10
184	24
~~200~~ 199	~~16~~ 15
90	~~110~~ 109
58	32
55	03
39	16
38	01
18	20

∴ Total tracks traversed : $50+10+24+\overset{15}{\cancel{16}}+\overset{109}{\cancel{110}}+32+3+16+1+20 = 280$

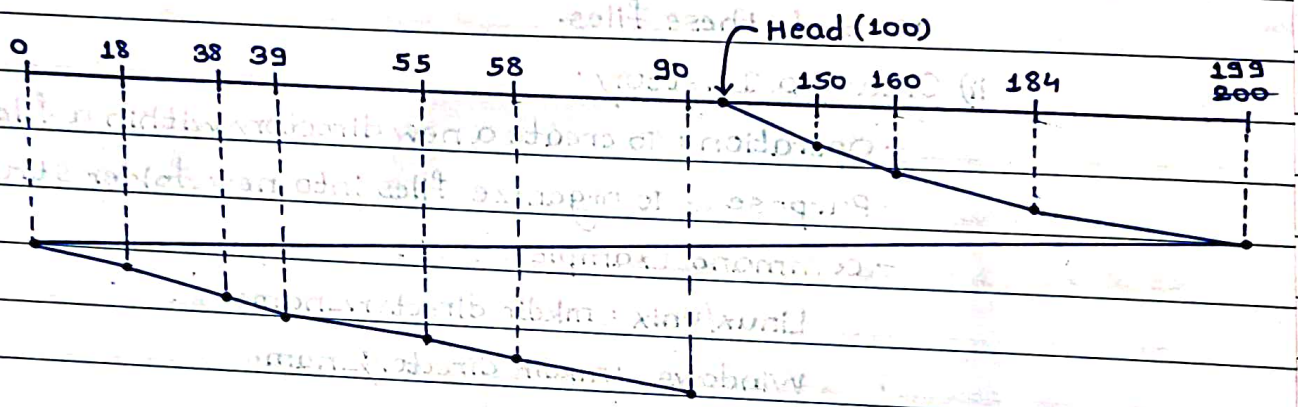
∴ Avg. seek length = $\frac{280}{10} = 28.00$

↑ total requests received.

DPU

3] C-SCAN (circular-scan) :

-This algo. also moves in one direction, servicing requests, but when it reaches the end of disk, it jumps back to other end without servicing any requests on return trip & then starts over again.



Next track accessed	No. of tracks traversed
100 (head)	—
150	50
160	10
184	24
~~200~~ 199	~~16~~ 15
0	~~200~~ 199
18	18
38	20
39	01
55	16
58	03
90	32

∴ Total tracks Traversed : $50+10+24+\overset{15}{\cancel{16}}+\overset{199}{\cancel{200}}+18+20+1+16+3+32 = 388$

∴ Avg. seek length = $\frac{388}{11} = 35.4527$

↑ total requests received.

Que. 2] What are typical operations that may be performed on a directory?

⇒ Ans :

i) In most operating systems, directories (or folders) serve as the containers for files and other directories. The operations performed on directories typically involve organizing, managing & controlling access to these files.

ii) Create a Directory :

- Operation : To create a new directory within a file system.
- Purpose : To organize files into new folder structure.
- Command Example :

  Linux/Unix : mkdir directory_name.

  Windows : mkdir directory_name

iii) Delete a Directory :

- operation : Removing a directory and, optionally, its contents.
- Purpose : To free up space or remove unused directories.
- Command Example :

  Linux/Unix : rmdir directory_name. (removes only empty directories)

  rm -r directory_name. (removes directories with content)

  Windows : rmdir directory_name.

  del /S /Q directory_name. (to delete contents)

iv) Rename a Directory :

- operation : changing the name of directory.
- Purpose : To update the directory name to reflect its contents better or follow a new naming convention.
- Command Example :

  Linux/Unix : mv old_name new_name

  Windows : ren old_name new_name

v) List Directory contents :

- operation : viewing the files and subdirectories within a directory.
- Purpose : To see what contents a directory contains.
- Command Example :

  Linux/Unix : ls directory_name

  Windows : dir directory_name

vi) Search within Directory :

- operation : searching for files or subdirectories within directory.
- Purpose : To locate specific files based on names, patterns, etc.
- command Example :

  Linux/unix : find directory_name -name "file_name"

  Windows : dir directory_name /s /p file_name

Que.3] What is I/o buffering ? Why I/o buffering is needed ? State and explain different approaches of I/o buffering.

⇒ Ans :

i) I/o buffering is a technique used in computer systems to temporarily hold data being transferred between two devices in a dedicated area of memory known as buffer.

ii) This process helps improve system performance and manage differences in data transfer rate between devices.

iii) Need of I/o Buffering :

1] Speed Mismatch

Different devices operate at different speeds (e.g. CPU, disks, and network interfaces). Buffers help smooth out these speed differences in data transfer rates between devices.

2] Efficient data transfer

Buffering allows data to be collected & transferred in larger blocks rather than byte by byte, which can reduce overhead of I/o operations.

3] Asynchronous I/O operations

With buffering, applications can continue executing while data is being read from or written to a device, improving overall system responsiveness & performance.

4] Minimized waiting Time.

By using buffers, programs can avoid waiting for the slow I/O operations to complete. Instead, they can perform other computations while data is being transferred.

iv) Different approaches of I/O Buffering :

1] Simple Buffering :

- A single buffer is allocated for I/O operations. Data is read into or written from this buffer during single I/O operation.
- simple to implement & manage.

Process -> Single Buffer -> I/O Device

2] Double Buffering :

- Two buffers are used to overlap I/O and processing. while one buffer is being filled with data from disk, the application can process data into other buffer.
- Reduces wait time & improves throughput ; but complex.

Process -> Buffer A (Filling) -> Buffer B (Writing) -> I/O Device

3] Circular Buffering :

- A circular buffer (ring buffer) is a fixed size buffer that treats the end of buffer as connected to beginning, allowing continuous writing & reading.
- Efficient for streaming data, minimizes need for constant allocation & deallocation of buffer space, but can lead to data loss if not handled properly.

Process -> Buffer 1 -> Buffer 2 -> Buffer 3 -> ... (Circular) -> I/O Device

4] No buffering :

- I/O operations are performed directly betⁿ application & I/O device without using buffer.

Que. 4] Explain with example any three disk scheduling criteria.

⇒ Ans :

i) Disk scheduling is crucial in optimizing the performance of the disk I/o system. When multiple I/o requests are generated, a disk scheduling algo. decides the order in which these requests are processed. There are several criteria to consider for effective disk scheduling:

ii) Seek Time :

- The time it takes for the disk's read/write head to move to the correct cylinder where the requested data is located.
- It can be minimized by choosing requests that require least movement of the head.
- e.g. Suppose current position of disk head is at cylinder 50 & there are requests for cylinder 10, 90, 20 and 60. A disk scheduling algo. like STIF would select requests in a way that minimizes the total head movement, which reduces seek time.

iii) Rotational Latency :

- The time spent waiting for the desired disk sector to rotate under read/write head after head is positioned on the correct track.
- Minimizing this latency can reduce overall I/o operation times.
- e.g. If a request needs data from a sector that is almost aligned with read/write head, rotational latency will be low. For example, after positioning the head on desired track, if target sector is at 30 degrees of rotation & head is currently at 25 degrees, rotational latency is minimal (5 degrees).

iv) Throughput :

- The total no. of I/o requests completed in a given amount of time.
- Maximizing throughput means the system is handling a higher number of requests efficiently.

requests received ✓

Que.5] A disk drive has 200 tracks, numbered 0-199. The drive is currently serving request at track no. 53. The queue of pending requests in FIFO order is 98, 183, 37, 122, 14, 124, 65, 67. Starting from current head position what is the total distance that disk arm moves to satisfy all pending requests for following disk scheduling algorithms. Assume that head is moving in increasing order for SCAN and C-LOOK.

i) FCFS ii) SCAN iii) C-LOOK iv) SSTF.

⇒ Ans :

① FCFS (First-come, First-Served):

- In FCFS, requests are served in the order they arrive.
- order : 98 → 183 → 37 → 122 → 14 → 124 → 65 → 67
- Head movement:

53 → 98 = 45

98 → 183 = 85

183 → 37 = 146

37 → 122 = 85

122 → 14 = 108

14 → 124 = 110

124 → 65 = 59

65 → 67 = 2

- ∴ Total Head movement = 45+85+146+85+108+110+59+2 = 640

② SCAN Algorithm:

- In SCAN, head moves towards one end servicing requests along the way, then reverses direction.
- Here, assumed that head is moving in increasing direction first.
- order : 53 - 65 - 67 - 98 - 122 - 124 - 183 - (199) - 37 - 14
- Head movement:

53 → 65 = 12

65 → 67 = 2

67 - 98 = 31

98 → 122 = 24

122 → 124 = 2

124 → 183 = 59

183 → 199 = 16

199 → 37 = 162

37 → 14 = 23

∴ Total head movement = 12+2+31+24+2+59+16+162+23 = 331 tracks

(iii) C-Look (Circular Look):

- In C-Look, head moves in one direction (here, in increasing order) to the last request & then jumps back to the first request in the queue, without servicing in reverse direction.
- order: 53 → 65 → 67 → 98 → 122 → 124 → 183 → 14 → 37
- Head movement:

53 → 65 = 12

65 → 67 = 2

67 → 98 = 31

98 → 122 = 24

122 → 124 = 2

124 → 183 = 59

183 → 14 = 169

14 → 37 = 23

∴ Total head movement = 12+2+31+24+2+59+169+23 = 322 tracks.

(iv) SSTF Algorithm:

- In SSTF, disk arm moves to nearest request first, minimizing the seek time for each request.
- order: 53 → 65 → 67 → 98 37 → 14 → 98 → 122 → 124 → 183

- Head movement:

53 → 65 = 12

65 → 67 = 2

67 → 37 = 30

37 → 14 = 23

14 → 98 = 84

98 → 122 = 24

122 → 124 = 2

124 → 183 = 59

∴ Total head movement = 12+2+30+23+84+24+2+59 = 236 tracks.

Que.6] ⇒ Explain with diagrams different I/O buffering techniques.

Ans :

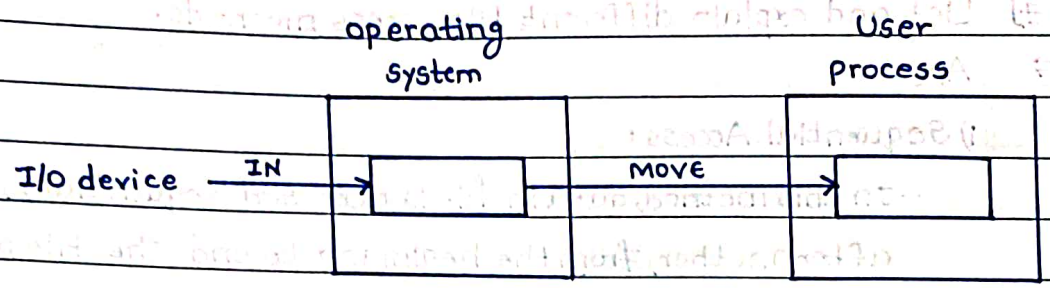
i) A buffer is a memory area that stores data being transferred betⁿ two devices or betⁿ a device & application.

ii) I/O buffering is a technique used in computer systems to improve the the efficiency of I/O operations. It involves temporary storage of data in buffer, which is reserved area of memory, to reduce no. of I/O operations & manage the flow of data betⁿ fast & slow devices or processes.

iii) There are 3 Types of I/O Buffering Techniques :

1] Single Buffering :

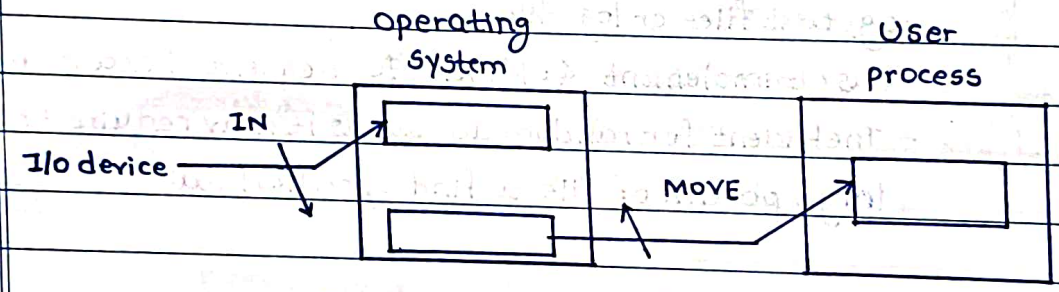
- Using one buffer to store data temporarily.
- A buffer is provided by OS to system portion of main memory.
- The system buffer takes I/P. → After taking I/P, block gets transferred to user space by process & then the process requests for another block. → Two blocks work simultaneously, when one block of data is processed by user process, next block is being read in.



SINGLE I/O BUFFERING.

2] Double Buffering :

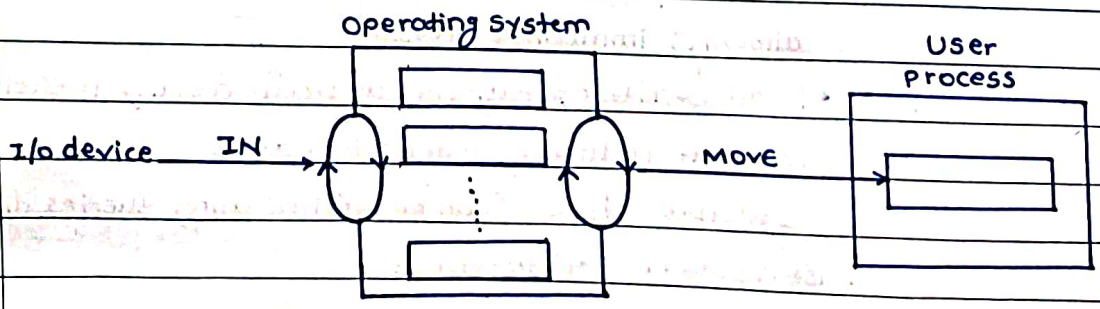
- In this technique, OS uses two buffers to allow continuous data transfer betⁿ two process or two devices.



DOUBLE I/O BUFFERING

3] Circular Buffering :

- In this technique, the OS uses a circular buffer to manage continuous data streams efficiently.
- In this, the data do not directly passed from producer to consumer because the data would change due to overwriting of buffers before they had been consumed.



CIRCULAR I/O BUFFERING

Que.7] List and explain different file access methods.

⇒ Ans :

i) Sequential Access :

- In this method, data in file is accessed sequentially, one record after another, from the beginning to end. The file pointer is automatically moved forward with each operation.
- working:
  - The OS reads data in order from the start of the file, processing records in sequence.
  - Write operations append data at end of file.
  - Read & write operations are usually carried out in linear order.
- e.g. text files or log files.
- Easy to implement & efficient for sequential access use cases.
- Inefficient for random access, as it may require traversing a large portion of file to find specific data.

ii) Direct (Random) Access :

- In this method, file can be accessed at any point directly, without reading data sequentially. The file is divided into fixed size blocks or records & OS can jump to any record using an index or relative position.
- working:
  - The OS uses an index or record no. to calculate exact location of desired data.
  - The file pointer is moved directly to a specified location, allowing immediate access.
  - Read & write operations can occur at any location in file without following order of records.
- e.g Database files. (can be fetched using queries, directly).
- Fast access to specified data.
- More complex to manage.

iii) Indexed Access :

- It uses an index to maintain a lookup table for the locations of records. The index points to different blocks or records in file, allowing efficient searching & retrieval.
- An index is created that holds pointers to different parts of file.
- e.g. Library systems : Searching for a book by title.

iv) Hashed Access :

- It uses a hash function to map keys directly to specific locations in file. It ensures fast & efficient retrieval of records without need for searching or indexing.
- A hash function is applied to a key (e.g. record ID) to generate a hash value. This hash value is used to calculate exact location in file where record is stored.
- e.g. Hash tables.
- Extremely fast access for retrieving records.
- collisions can occur (when two keys map to same location), requiring additional management.

v) Clustered Access :

Here, related data is stored together in clusters on the disk. This method is useful for applications where accessing related data together is common, reducing disk seek time.

Que. 8] Describe different methods of record blocking.

⇒ Ans :

1) Record blocking is a technique used in file systems & databases to store multiple records together as a single block or unit. This helps in optimizing I/o operations by minimizing overhead of accessing individual records.

ii) Fixed Length Blocking :

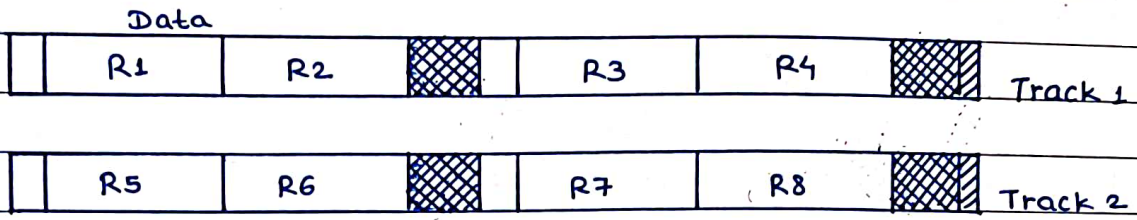
- Each record in the file has fixed size. This means that every record occupies the same amount of space, regardless of the actual size of data being stored.
- Multiple fixed length blocks are grouped together to form a block. If a record is smaller than fixed size, it is padded with additional bytes to meet required length.
- Simple implementation & easy to calculate offsets.
- Predictable access time since size of record is known.
- Wasted space for records that do not fill entire fixed size.

iii) Variable-length Spanned Blocking :

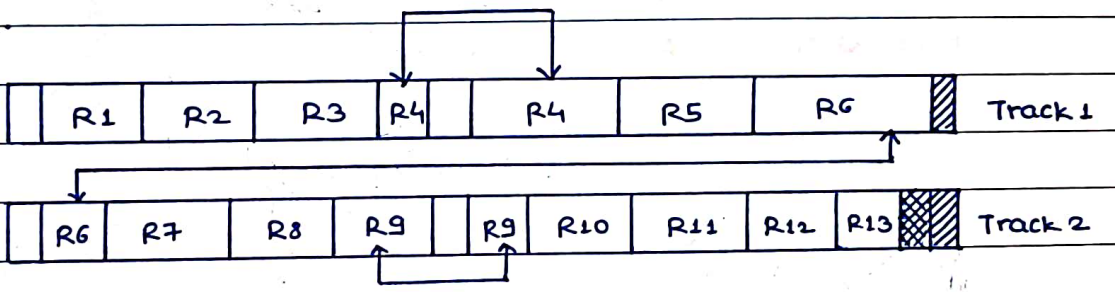
- Here, when a record is too large to fit entirely within a single block, the record is allowed to 'span' across multiple blocks. This means that part of record is stored in one block & remaining part continues in next block.
- A pointer or indicator is used to signal that the record continues in next block.
- Efficient use of space.
- complex structure.

iv) variable-length Unspanned Blocking :

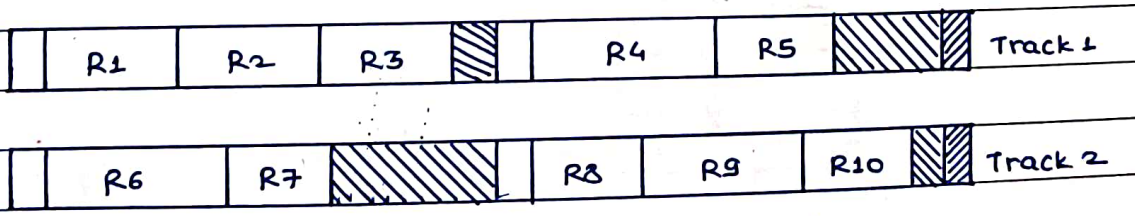
- Here, each block can hold only entire records.
- If a record is too large to fit into remaining space of block, it will be placed entirely in the next block, leaving the remaining space in current block unused.
- Simpler to implement comparatively & easier to manage.
- Wastage of space, leading to fragmentation.
- Inefficient for large records.



Fixed Blocking



Variable Blocking : Spanned



Variable Blocking : Unspanned

where,

- = Gaps due to hardware design
- = waste due to block size constraint from fixed record size.
- = waste due to record fit to block size.
- = waste due to block fit to track size.

Name : Aniruddha Shivaji Repale.

Roll No.: TEIT-10

DPU

* OS : Class Assignment – 6 *

Que.1] ⇒ List down the phases of a compiler. Explain with suitable example.

Ans :

i) The process of compiling a high-level programming language into machine code is divided into several phases, collectively called the phases of compiler.

ii) These phases ensure that the source code is correctly translated into an executable format.

iii) The phases of a compiler can be grouped into two main categories : analysis and synthesis.

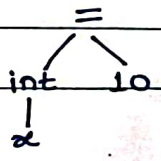
1] Lexical Analysis (Scanner)

- The Lexical Analyzer reads the source code as a stream of characters and converts it into tokens (the smallest units of meaning like keywords, operators, identifiers, etc.)
- e.g. If the input code is `int x = 10;` the lexer converts it into tokens like :
  - `int` → data type keyword
  - `x` → identifier (variable)
  - `=` → assignment operator
  - `10` → constant
  - `;` → semicolon (statement terminator).

2] Syntax Analysis (Parser)

- The parser takens the tokens from the lexical analyzer and checks them against the syntax rules of language (grammar). It constructs a syntax tree to represent structure of program.
- e.g For the code `int x = 10;`, the parser verifies that the structure is valid as per the language's syntax and builds a parse tree, showing assignment operation & declaration.

Example: `int x = 10`



3] Semantic Analysis :

- The semantic analyzer checks whether the parse tree follows the semantic rules of the language, such as type checking, scope resolution & identifier declarations.
- e.g. For int x = 10 ; , the semantic analyzer verifies that the variable x is of type int and the value 10 is a valid integer that can be assigned to it. If you had int x = "hello" ; , it would throw a semantic error because a string is being assigned to an integer.

4] Intermediate Code Generation :

- The compiler generates an intermediate code, which is lower-level representation of source code but not yet machine code. It's often platform-independent & easy to optimize.
- e.g. For int x = 10 ; , an intermediate code in three-address form could be something like :

  x = 10

5] Code optimization :

- This phase optimizes the intermediate code to make it more efficient without altering its behaviour. This includes removing redundant code, reducing the number of operations, & optimizing memory usage.
- e.g. If there's redundant or unused code in program, like unnecessary variable assignments, they could be eliminated or simplified.

6] Code Generation :

- In this phase, the optimized intermediate code is converted into machine code or assembly code, specific to target architecture. (e.g. X86, ARM).

- e.g. The statement x=10; might be translated into machine code instructions for an x86 processor:

```
Mov AX, 10
Mov [X], AX
```

7] Code Linking and Assembly :

- The final step involves linking all the machine code & external libraries or modules, if any, to create an executable file. This step ensures that function cells & references to external resources are correctly resolved.
- once the machine code is generated, linker combines it with libraries or other parts of prog. & produces final executable, such as a.exe or a.out.

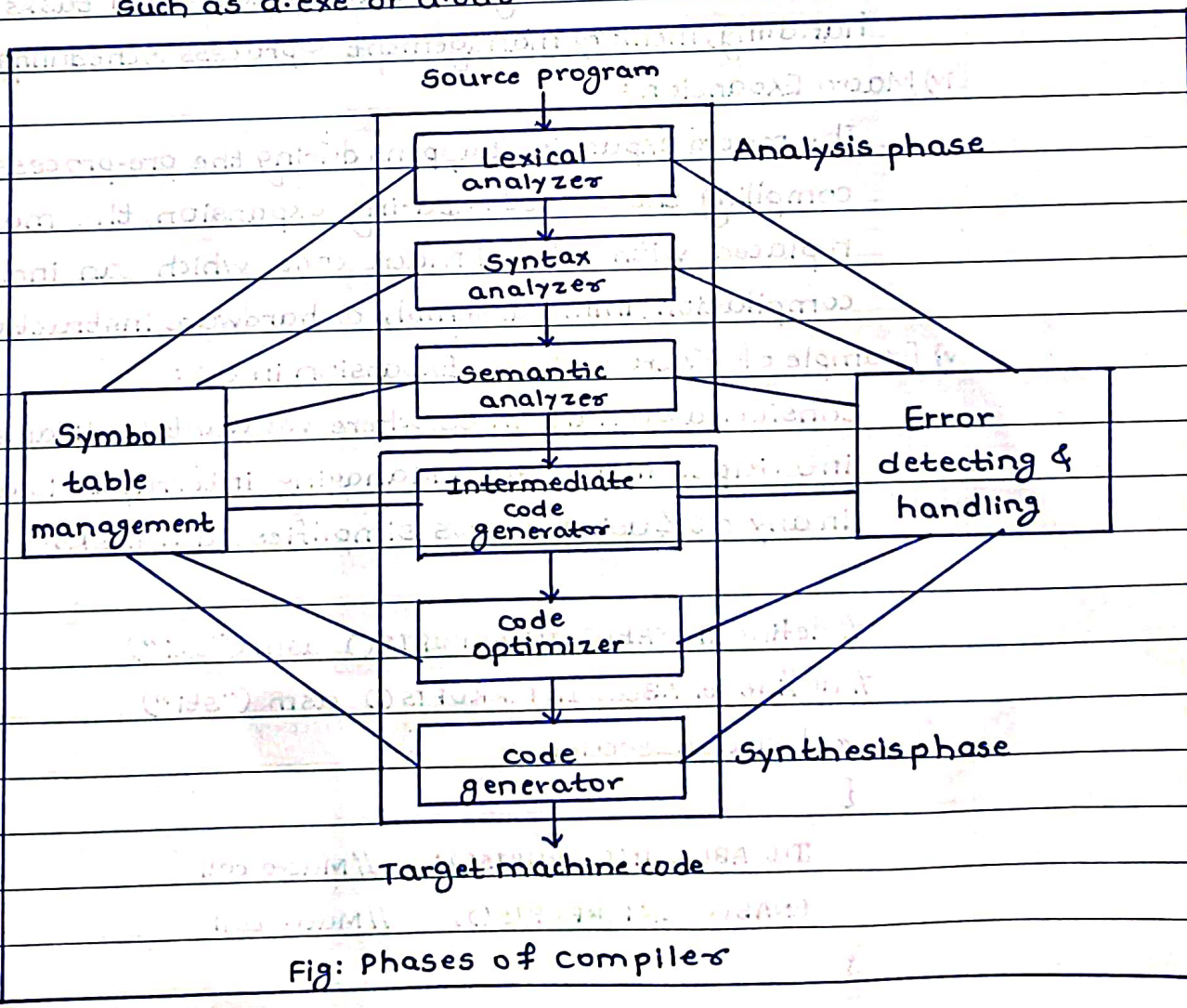


Fig: Phases of compiler

Que.2] Explain macro call and macro expansion with example.

⇒ Ans :

i) In the context of os, macro plays a crucial role in simplifying the management of system code & making code more modular and reusable.

ii) Macros in os are often used for :

1. System call interface management.
2. Handling repetitive operations.
3. Abstracting hardware-specific instructions.

iii) Macro call :

A macro call in an os is when the os kernel or other system-level code uses a predefined macro in place of actual code or logic. Macros can help manage various system-level tasks like interrupt handling, memory management & process scheduling.

iv) Macro Expansion :

The macro expansion happens during the pre-processing phase of compiling the os code. During expansion, the macro call is replaced with defined macro code, which can include conditional compilation, inline assembly or hardware instructions.

v) Example of Macro call and Expansion in os :

Consider a scenario in os where we want to disable & enable interrupts in a system. Managing interrupts is a key task in any os & using macros simplifies such tasks.

```
#define DISABLE_INTERRUPTS() asm("cli")
#define ENABLE_INTERRUPTS() asm("sti")
void critical_section()
{
    DISABLE_INTERRUPTS();    //Macro call
    ENABLE_INTERRUPTS();     //Macro call
}
```

DPU

Explanation :

- #define DISABLE_INTERRUPTS() asm("cli") : This macro issues the cli instruction, which disables interrupts on CPU.
- #define ENABLE_INTERRUPTS() asm("sti") : This macro issues the sti instruction, which re-enables interrupts.
- The macro DISABLE_INTERRUPTS() is invoked in the function critical_section(). This is macro call where interrupts are disabled.
- Similarly, ENABLE_INTERRUPTS() is called at the end of critical section to re-enable interrupts.
- During the preprocessing phase, DISABLE_INTERRUPTS() is expanded to asm("cli") & ENABLE_INTERRUPTS() is expanded to asm("sti"). This results in:

```
void critical_section() {
    asm("cli");    //disable interrupts
    asm("sti");    //enable interrupts
}
```

vi) Real world applications :

1] Interrupt Handling : Many os kernels use macros to manage critical sections where interrupts need to be disabled temporarily to ensure data consistency & atomicity of operations.

2] Memory management : Macros are often used to allocate or deallocate memory & handle page tables.

3] Task Scheduling : In context switches (switching from one process to another), macros are often used to save or restore the state of process, making it easier to manage code complexity.

vii) Advantages :

1) code simplification
2) Hardware abstraction.
3) Efficiency.

viii) Disadvantages :

1) Debugging difficulty
2) Lack of Type safety.

iv) Example in Memory Management:

In OS, especially those with virtual memory support, macros are used to convert between physical and virtual memory addresses.

e.g.

```
#define PAGE_SIZE 4096
#define PAGE_ALIGN(addr) ((addr+PAGE_SIZE-1) & ~(PAGE_SIZE-1))
void allocate_page (void *address)
{
    void *aligned_address = (void *)PAGE_ALIGN((unsigned long)address);
}
```

Here,

PAGE_ALIGN (addr) is a macro that aligns an address to the nearest page boundary.

When allocate_page() is called, the macro call is expanded into the alignment logic for memory management purposes.

Que.3] Explain with example imperative statements, declarative statement, & assembly directives of assembly language programming.

⇒ Ans:

i) Imperative statements:

- These are instructions that direct the processor to perform specific operations or actions. These include instructions like arithmetic operations, data transfer or control flow.
- Imperative statements directly translate to machine code & are executed by processor.
- e.g.

```
MOV AX, 5   ; Move the value 5 into register AX.
ADD AX, 2   ; Add 2 to the value in register AX.
```

ii) Declarative Statements :

- They are used to define data or allocate memory but do not generate executable code.
- These statements define variables, constants & storage locations.
- They allow the programmer to specify data types and values that will be used by program.
- e.g.

```
DATA_SEG SEGMENT
    NUM1 DB 10 ; Declare a byte with value 10
    NUM2 DW 20 ; Declare a word (2 bytes) with value 20
DATA_SEG ENDS
```

Here,

NUM1 DB 10 : This declarative statement defines a variable NUM1 that stores a byte (DB) with value 10.

NUM2 DW 20 : This declares a word (DW, 2 bytes) named NUM2 with value 20.

iii) Assembly Directives :

- They provide instructions to assembler on how to process the prog.
- They are not converted into machine code but are used to control organization & structure of assembly code, such as defining segments, managing memory or controlling conditional assembly.
- Directives are essentially commands for assembler that affect assembly process, but they do not result in executable code.
- e.g.

```
ORG 100h        ; set the origin to address 100h
SECTION .text   ; start a section for the code.
```

-

- Common Assembly Directives:

1] SEGMENT/ENDS : Used to define and close a segment.

e.g:

```
DATA_SEG SEGMENT
  ; data declarations
DATA_SEG ENDS
```

2] DB, DW, DD : Used to define bytes, words or double words of data

e.g:

```
MSG DB 'Hello!',0 ; Declare a string.
```

3] EQU : Used to assign a constant value to symbol.

e.g:

```
MAXLEN EQU 256 ; Define a const. MAXLEN with value 256.
```

Que.4] What is System software? Explain any 4 system s/w in brief.

⇒ Ans:

i) System software consists of a set of programs that support the operation of a computer system and help the programmer to simplify the programming process & create a environment to run application s/w efficiently.

ii) It acts as an intermediary between hardware and end users or the application programs.

iii) Key Functions of System software:

1] Manages hardware resources (e.g. CPU, Memory, Storage)

2] Provide a UI or CLI to interact with system.

3] Ensures efficient operation of system.

4] Provides platform for running application s/w.

iv) Examples of system s/w:

① Operating System (OS):-

- It manages H/W resources of computer, provides a user interface

and offers services for application software.
- It is responsible for managing memory, processes, I/O devices & files.
- e.g. Windows, macOS, Linux, Android.

② Device drivers :-
- They are specialized system s/w programs that allow OS to communicate with hardware devices like printers, network adapters or video cards.
- Without drivers, OS would not be able to control hardware components properly.
- It translates OS instructions into commands that H/w can understand.

③ Utility Software :-
- Provides specialized tools for system maintenance, performance monitoring & optimization.
- These tools help manage, optimize & analyze the overall functioning of computer.
- e.g. Antivirus prog., backup tools, file compressors, etc.

④ Compiler :-
- It is a system s/w tool that translates code written in high level prog. languages (like C, C++, Java) into machine code (binary) that the processor can execute.
- It performs several phases, including lexical analysis, syntax analysis & code generation.

⑤ Linker :-
System s/w that combines various object code files produced by compiler into a single executable program.

⑥ Loader :-
System s/w that loads executable prog. into memory for execution.

Que.5] Discuss with example what is forward reference problem.

⇒ Ans:

i) The forward reference problem occurs in programming or assembly language when a symbol (such as variable, label or function) is used before it is defined or declared.

ii) This is a common issue in low-level programming, particularly in assembly language & compilers, where all symbols need to be defined before they are referenced.

iii) Assemblers or compilers that process code in a single pass struggle with forward references because they encounter an undefined symbol before its defination.

iv) In assembly language, this problem occurs when the assembler encounters an instruction that refers to a label or memory location that hasn't been defined yet.

v) Example:

```
JMP END          ; Forward reference to END label (not yet defined)

START:           ; Defination of START Label
  MOV AX, 5

END:             ; Defination of END Label
  MOV BX, AX
  HLT
```

In this example,

The instruction JMP END refers to a label END, but the END label is defined later in the code.

If assembler processes code line by line in single pass, it will encounter JMP END instruction before it has encountered END label defination, causing an error.

vi) Solutions to Forward Reference Problem :

1] Two-Pass Assembler :-

- First Pass : The assembler scans the entire program and records the locations of all labels and symbols in a symbol table. It doesn't generate machine code at this stage.
- Second Pass : The assembler processes the code again, this time using the info. in symbol table to resolve all forward references.

So in given example, the assembler is able to resolve JMP END instruction because symbol table will contain address of END label by the time the second pass is made.

2] Backpatching :-

The assembler generates incomplete instructions (with placeholders) during first pass & goes back to 'patch' them once the symbol is defined. This allows assembler to fill in the correct addresses for forward references after definations are found.

Que.6] Explain ORIGIN, EQU and LTRORG with an example.

⇒ Ans :

i) In the assembly language, ORIGIN, EQU & LTORG are commonly used assembler directives that guide the assembler on how to organize & allocate memory, define constants & manage literal values.

ii) ORIGIN (ORG) Directive :

- It is used to specify the starting address for the assembler to place subsequent instructions/data in memory. This is useful when programmer wants to load code/data at specific memory address.
- e.g.

```
ORG 100h ; Set the starting address to 100h (256 in decimal)
MOV AX, 5  ; This instruction is placed at address 100h
MOV BX, 10 ; This instruction follows the previous one.
```

iii) EQU Directive :

- The Equate (EQU) directive is used to define a constant.
- It assigns a symbolic name to a numeric value or expression, making the code more readable & easier to maintain.
- The value defined with EQU does not occupy memory - it simply serves as a substitution for constant value in code.
- e.g.

```
MAXLEN  EQU 100     ; Define a constant MAXLEN with value of 100.
MOV CX,MAXLEN       ; Move the value of MAXLEN(100) into register CX.
ADD CX,MAXLEN       ; Add the value of MAXLEN(100) to CX.
```

iv) LTORG Directive :

- The LTORG (Literal Pool origin) directive instructs the assembler to assemble all the literals (constant values used in program) at the current pt. in code.
- Literals are usually stored in a seperate literal pool & LTORG is used to force the placement of those literals into memory at a specific location in program.
- This is useful to control where literals are placed in memory, especially in long programs.
- e.g.

```
MOV R1, =5      ; Use a literal value 5
MOV R2, =10     ; Use another literal value 10
LTORG           ; Place the literals into memory at this point.
MOV R3, =15     ; Another literal.
```

These directives provide important functionality for managing memory, defining constants & controlling the placement of literal values in assembly lang. programs.

DPU

Que.7] Explain the data structures required for two PASS Assembler.

⇒ Ans:

i) In a two-pass assembler, the assembly program is processed in two distinct phases to handle forward references and to generate machine code. During first pass, assembler builds necessary data structures to gather info. about symbols & addresses & in the second pass, it resolves these addresses & generates final machine code.

ii) Data structures required for Two pass assembler:

1] Symbol Table (SYMTAB):

- It stores info. about labels and variables used in the program. It associates each symbol with its memory address & attributes.
- e.g.

Symbol	Address
START	0000
LOOP	0010
VALUE	0030

2] Literal Table (LITTAB):

- It stores literal constants used in program (e.g. =5, ='A') and their corresponding memory locations.
- e.g.

Literal	Address
=5	0100
='A'	0104

3] Location Counter (LOCCTR):

- It keeps track of current memory locations during the assembly process.
- e.g.

Initial LOCCTR : 0000

After processing an instruction (MOV AX, 5) : LOCCTR updated to 0004.

4] Intermediate Code (Intermediate File) :

- It stores a partial representation of assembly program after the first pass. It includes symbolic references (for later resolution) & intermediate form of instructions.
- e.g.

Line Number	Opcode	Operand	Address
1	MOV	AX,5	Unresolved
2	JMP	LOOP	Unresolved

5] Opcode Table (OPTAB) :

- It stores info. about mnemonics of assembly language, their corresponding machine code & format of instruction.
- e.g.

Mnemonic	opcode	Length
MOV	89	2
ADD	01	2
JMP	E9	3

6] Pool Table (POOLTAB) :

- It is used to keep track of literal pools, which are areas in memory where literal values are stored.
- It helps in managing placement of literals when LTORG directive is used or when literals are stored at end of prog.
- e.g.

Pool Number	Start Address
1	0100
2	0200

7] Program counter (PC) :

It holds the memory address of next instruction to be executed or assembled.

DPU

Que.8] Differentiate between literal and immediate operand.

⇒ Ans :

Criteria	Literal operand	Immediate operand
i) Defination	A constant value that is stored in memory at runtime.	A const. value that is embedded directly in the instruction.
ii) Storage	The value is placed in a literal pool (memory).	The value is part of the instruction itself, stored in instruction code.
iii) Access	The operand refers to a memory address where literal is stored.	The operand value is directly accessed from the instruction.
iv) Example (Assembly)	MOV R1, =5 (literal 5 stored in memory)	MOV R1, #5 (5 is embedded in instruction)
v) Memory Usage	Requires memory location to store literal value.	Doesn't require extra memory for operand (apart from the instruction size).
vi) Efficiency	slightly slower due to the memory access for literal.	Faster since the value is already part of instruction.
vii) Use	Typically used when large, less frequent constants are needed	used when small constants need to be quickly accessed.
viii) Addressing mode	Refers to direct or indirect memory addressing	Uses the immediate addressing mode.