2022

Victoria University of Bangladesh

58/11/A, Panthapath, Dhaka, Bangladesh

Operating System Concepts

CSI-231

Final Assessment

Submitted By:

ARS Nuray Alam Parash

ID # 2120180041

18th Batch, BSc in CSE

E-mail: chatokparash@gmail.com

Mobile: 01922339393

Submission Date: 6 October 2022

Submitted To:

Renea Chowdhury Shormi

Lecturer

Department of Computer Science & Engineering

Victoria University of Bangladesh (VUB)

Answer to the Question No- 1 (a)

Valid Bit- It indicates that the associated page is in the process logical address space, and is thus a legal page.

Invalid Bit- It is indicating that the page is not in the process logical address space

- With each page table entry, a valid–invalid bit is associated
- (1 is in-memory, 0 is not-in-memory)
- Initially, valid-invalid but is set to 0 on all entries.
- Example of a page table snapshot.

Frame #	valid-invalid bit
	1
	1
	1
	1
	0
⋮	
	0
	0

- During address translation, the valid-invalid bit in page table entry is 0 is page fault.

Answer to the Question No- 1 (b)

Architecture of Segmentation: Segmentation gives a memory-management scheme that increases the support of the user's perspective on memory.

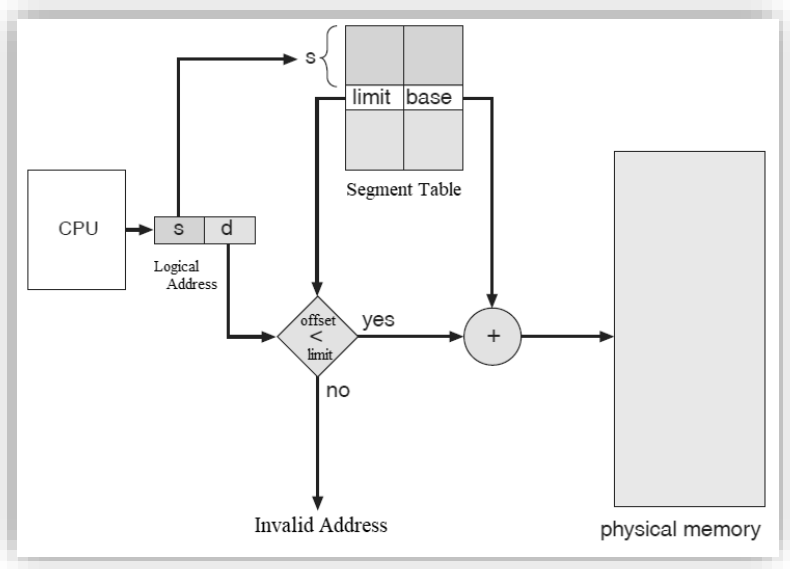
The segment table mainly contains two types of data about the segment:

- **Segment Base Address:** Addresses specify both the name of the segment and the offset inside the segment. The user, thus, determines each address by two amounts: a segment name and an offset.
- **Segment Limit:** This indicates the length of the segment.

The Architecture of Segmentation: The hardware architecture of segmentation and the use of a segment table is illustrated in Figure. A logical address consists of two parts:

- **A Segment Number:** The segment number is used as an index to the segment table. It is represented by s.
- **An Offset into that Segment:** It is represented by d. The offset d of the logical address must be between 0 and the segment limit. If it is not, we trap to the operating system (logical addressing attempt beyond the end of the segment). When an offset is legal, it is added to the segment base to produce the address in the physical memory of the desired byte. The segment table is thus essentially an array of base–limit register pairs.

Hardware Architecture of Segmentation:



Advantages of Segmentation

- Segmentation provides a powerful memory management mechanism.
- It allows user to partition their programs into modules that operate independently of one another.
- The segment table is of lesser size as compared to the page table in paging.
- Segments allow two processes to easily share data.
- The average Segment Size is larger than the actual page size.
- It is easier to relocate segments than the entire address space.
- There is no internal fragmentation
- Less overhead

Disadvantages

- It can have external fragmentation.
- It is difficult to allocate contiguous memory to a variable-sized partition.
- Costly memory management algorithms.
- Segmentation: find free memory area big enough.
- Paging: keep a list of free pages, any page is ok.
- Segments of inconsistent sizes are also not fit for swapping.

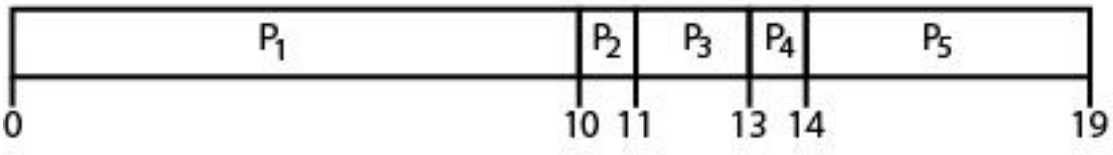
Answer to the Question No- 1 (c)

Physical Address: In computing, physical address refers to a memory address or the location of a memory cell in the main memory. It is used by both hardware and software for accessing data. Software, however, does not use physical addresses directly; instead, it accesses memory using a virtual address. A hardware component known as the memory management unit (MMU) is responsible for translating a virtual address to a physical address.

In networking, physical address refers to a computer's MAC address, which is a unique identifier associated with a network adapter that is used for identifying a computer in a network.

Answer to the Question No- 2 (a)

i. **The Gantt Chart for the FCFS schedule is:**



Completion Time:

P_1 = 10 ms, P_2 = 11 ms, P_3 = 13 ms, P_4 = 14 ms, P_5 = 19 ms

So, The FCFS Schedule's Average Completion Time:

(10 + 11 + 13 + 14 + 19) ms = 67 ms = (67 / 5) ms **= 13.40 ms**

Waiting Time:

P_1 = 0 ms, P_2 = 10 ms, P_3 = 11 ms, P_4 = 13 ms, P_5 = 14 ms

So, The FCFS Schedule's Average Waiting Time:

(0 + 10 + 11 + 13 + 14) ms = 48 ms = (48 / 5) ms **= 9.60 ms**

Turnaround Time: (Turnaround Time= Burst Time + Waiting Time)

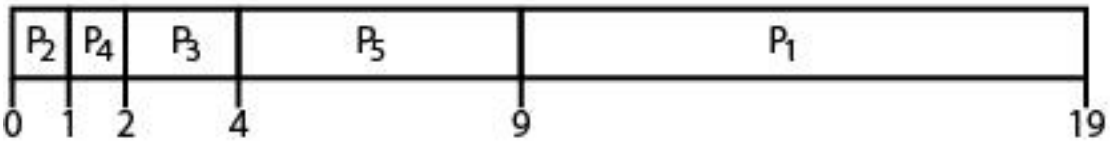
P_1 = (10 + 0) = 10 ms, P_2 = (1 + 10) = 11 ms, P_3 = (2 + 11) = 13 ms

P_4 = (1 + 13) = 14 ms, P_5 = (5 + 14) = 19 ms

So, The FCFS Schedule's Average Turnaround Time:

(10 + 11 + 13 + 14 + 19) ms = 67 ms = (67 / 5) ms **= 13.4 ms**

ii. **<u>The Gantt Chart for the SJF Non-Preemptive schedule is:</u>**



Completion Time:

P_1 = 21 ms, P_2 = 1 ms, P_3 = 7 ms, P_4 = 3 ms, P_5 = 12 ms

So, The SJF- Non-preemptive Schedule's Average Completion Time:

(21 + 1 + 7 + 3 + 12) ms = 44 ms = (44 / 5) ms **= 8.80 ms**

Waiting Time:

P_1 = 12 ms, P_2 = 0 ms, P_3 = 3 ms, P_4 = 1 ms, P_5 = 7 ms

So, The SJF- Non-preemptive Schedule's Average Waiting Time:

(12 + 0 + 3 + 1 + 7) ms = 23 ms = (23 / 5) ms **= 4.60 ms**

Turnaround Time: (Turnaround Time= Burst Time + Waiting Time)

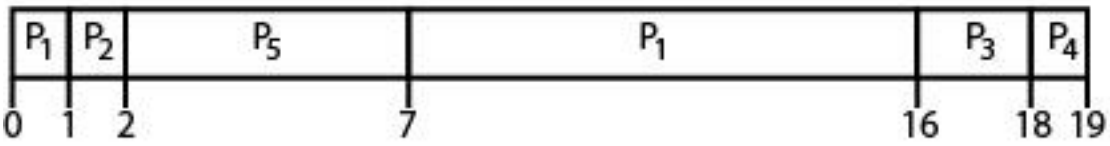
P_1 = (9 + 12) = 21 ms, P_2 = (1 + 0) = 1 ms, P_3 = (4 + 3) = 7 ms

P_4 = (2 + 1) = 3 ms, P_5 = (5 + 7) = 12 ms

So, The SJF- Non-preemptive Schedule's Average Turnaround Time:

(21 + 1 + 7 + 3 + 12) ms = 44 ms = (44 / 5) ms **= 8.80 ms**

iii. **The Gantt Chart for the Priority-Preemptive schedule is:**



Completion Time:

P_1 = 7 ms, P_2 = 1 ms, P_3 = 16 ms, P_4 = 18 ms, P_5 = 2 ms

So, The Priority- (Preemptive) Schedule's Average Completion Time:

(7 + 1 + 16 + 18 + 2) ms = 44 ms = (44 / 5) ms **= 8.80 ms**

Waiting Time:

P_1 = (7 - 1 - 0) = 6 ms, P_2 = (1 - 0 - 1) = 0 ms, P_3 = (16 - 0 - 2) = 14 ms

P_4 = (18 - 0 - 3) = 15 ms, P_5 = (2 - 0 - 4) = -2 ms

So, The Priority- (Preemptive) Schedule's Average Waiting Time:

(6 + 0 + 14 + 15 + (-2)) ms = 33 ms = (33 / 5) ms **= 6.60 ms**

Turnaround Time: (Turnaround Time= Completion Time - Arrival Time)

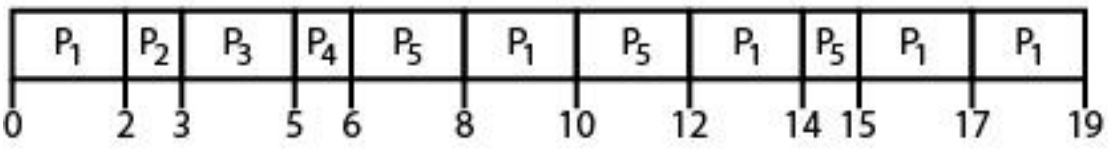
P_1 = (16 - 0) = 16 ms, P_2 = (2 - 1) = 1 ms, P_3 = (18 - 2) = 16 ms

P_4 = (19 - 3) = 16 ms, P_5 = (7 - 4) = 3 ms

So, The Priority- (Preemptive) Schedule's Average Turnaround Time:

(16 + 1 + 16 + 16 + 3) ms = 132 ms = (132 / 5) ms **= 26.40 ms**

iv. **<u>The Gantt Chart for the Round Robin with Time Quantum= 2 schedule is:</u>**



Completion Time:

P_1 = 19 ms, P_2 = 3 ms, P_3 = 5 ms, P_4 = 6 ms, P_5 = 15 ms

So, The Round Robin Schedule's Average Completion Time:

(19 + 3 + 5 + 6 + 15) ms = 48 ms = (48 / 5) ms **= 9.60 ms**

Waiting Time:

P_1 = (8 - 2) + (12 - 10) + (15 - 14) = 9 ms,

P_2 = (2 - 0) = 2 ms, P_3 = (3 - 0) = 3 ms,

P_4 = (5 - 0) = 5 ms, P_5 = (6 - 0) + (10 - 8) + (14 -12) = 10 ms

So, The Round Robin Schedule's Average Waiting Time:

(9 + 2 + 3 + 5 + 10) ms = 29 ms = (29 / 5) ms **= 5.80 ms**

Turnaround Time: (Turnaround Time= Burst Time + Waiting Time)

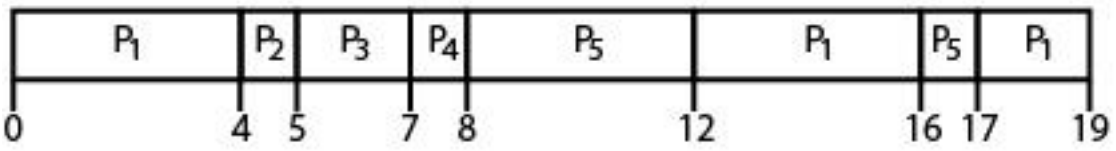
P_1 = (10 + 9) = 19 ms, P_2 = (1 + 2) = 3 ms, P_3 = (2 + 3) = 5 ms

P_4 = (1 + 5) = 6 ms, P_5 = (5 + 10) = 15 ms

So, The Round Robin Schedule's Average Turnaround Time:

(19 + 3 + 5 + 6 + 15) ms = 48 ms = (48 / 5) ms **= 9.60 ms**

v. **<u>The Gantt Chart for the Round Robin with Time Quantum= 4 schedule is:</u>**



Completion Time:

P_1 = 19 ms, P_2 = 5 ms, P_3 = 7 ms, P_4 = 8 ms, P_5 = 17 ms

So, The Round Robin Schedule's Average Completion Time:

(19 + 5 + 7 + 8 + 17) ms = 56 ms = (56 / 5) ms **= 11.20 ms**

Waiting Time:

P_1 = (12 - 4) + (17 - 16) = 9 ms, P_2 = (4 - 0) = 4 ms,

P_3 = (5 - 0) = 5 ms, P_4 = (7 - 0) = 7 ms, P_5 = (8 - 0) + (16 - 12) = 12 ms

So, The Round Robin Schedule's Average Waiting Time:

(9 + 4 + 5 + 7 + 12) ms = 37 ms = (37 / 5) ms **= 7.40 ms**

Turnaround Time: (Turnaround Time= Burst Time + Waiting Time)

P_1 = (10 + 9) = 19 ms, P_2 = (1 + 4) = 5 ms, P_3 = (2 + 5) = 7 ms

P_4 = (1 + 7) = 8 ms, P_5 = (5 + 12) = 17 ms

So, The Round Robin Schedule's Average Turnaround Time:

(19 + 5 + 7 + 8 + 17) ms = 56 ms = (56 / 5) ms **= 11.20 ms**

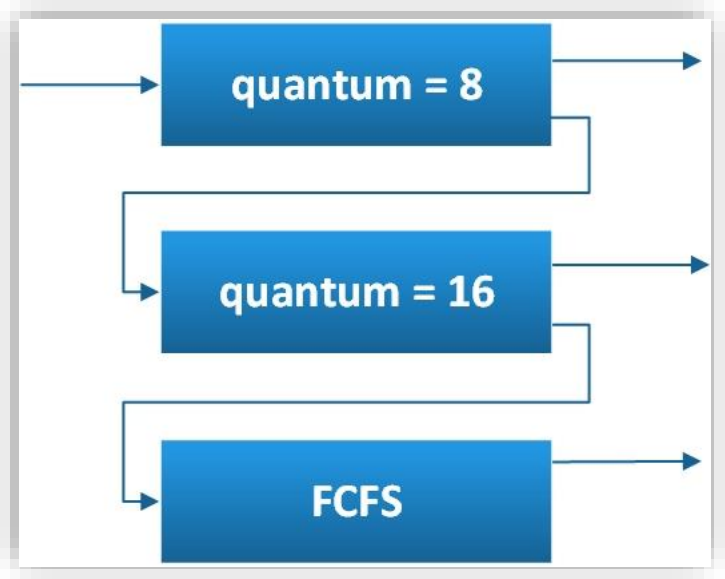
Answer to the Question No- 2 (b)

Multilevel Feedback Queue Scheduling: Multilevel Feedback Queue Scheduling (MLFQ) CPU Scheduling is like Multilevel Queue (MLQ) Scheduling but in this process can move between the queues. And thus, much more efficient than multilevel queue scheduling.

In general, a multilevel feedback queue scheduler is defined by the following parameters:

- The number of queues.
- The scheduling algorithm for each queue.
- The method used to determine when to upgrade a process to a higher-priority queue.
- The method used to determine when to demote a process to a lower-priority queue.
- The method used to determine which queue a process will enter when that process needs service.

An example of a multilevel feedback queue can be seen in the below figure-



Explanation: First of all, suppose that queues 1 and 2 follow round robin with time quantum 8 and 16 respectively and queue 3 follows FCFS. One of the implementations of Multilevel Feedback Queue Scheduling is as follows:

- If any process starts executing then firstly it enters queue 1.
- In queue 1, the process executes for 8 unit and if it completes in these 8 units or it gives CPU for I/O operation in these 8 units unit than the priority of this process does not change, and if for some reasons it again comes in the ready queue than it again starts its execution in the Queue 1.
- If a process that is in queue 1 does not complete in 8 units then its priority gets reduced and it gets shifted to queue 2.

- Above points 2 and 3 are also true for processes in queue 2 but the time quantum is 16 units. Generally, if any process does not complete in a given time quantum then it gets shifted to the lower priority queue.
- After that in the last queue, all processes are scheduled in an FCFS manner.
- It is important to note that a process that is in a lower priority queue can only execute only when the higher priority queues are empty.
- Any running process in the lower priority queue can be interrupted by a process arriving in the higher priority queue.

Also, the above implementation may differ for the example in which the last queue will follow Round-robin Scheduling.

In the above Implementation, there is a problem and that is; Any process that is in the lower priority queue has to suffer starvation due to some short processes that are taking all the CPU time.

And the solution to this problem is : There is a solution that is to boost the priority of all the process after regular intervals then place all the processes in the highest priority queue.

Answer to the Question No- 3 (a)

Deadlock characterization describes the distinctive features that are the cause of deadlock occurrence. Deadlock is a condition in the multiprogramming environment where the executing processes get stuck in the middle of execution waiting for the resources that have been held by the other waiting processes thereby preventing the execution of the processes.

Deadlock can arise if four conditions hold simultaneously:

- **Mutual Exclusion:** Only one process at a time can use a resource. If another process requests that resource, the requesting process must be delayed until the resource has been released.
- **Hold and Wait:** A process that holding at least one resource is waiting to acquire additional resources held by other processes.
- **No Preemption:** A resource can be released only voluntarily by the process holding it, after that process has completed its task.
- **Circular Wait:** there exists a set {P0, P1, …, P0} of waiting processes such that-

  P0 is waiting for a resource that is held by P1,

  P1is waiting for a resource that is held by P2, …, Pn–1is waiting for a resource that is held by Pn, and P0is waiting for a resource that is held by P0.

Answer to the Question No- 3 (b)

Deadlock Prevention: Deadlocks can be prevented by preventing at least one of the four required conditions-

Mutual Exclusion:

- Shared resources such as read-only files do not lead to deadlocks.
- Unfortunately, some resources, such as printers and tape drives, require exclusive access by a single process.

Hold and Wait: To prevent this condition processes must be prevented from holding one or more resources while simultaneously waiting for one or more others. There are several possibilities for this-

- Require that all processes request all resources at one time. This can be wasteful of system resources if a process needs one resource early in its execution and doesn't need some other resource until much later.
- Require that processes holding resources must release them before requesting new resources, and then re-acquire the released resources along with the new ones in a single new request. This can be a problem if a process has partially completed an operation using a resource and then fails to get it re-allocated after releasing it.
- Either of the methods described above can lead to starvation if a process requires one or more popular resources.

No Preemption: Preemption of process resource allocations can prevent this condition of deadlocks, when it is possible.

- One approach is that if a process is forced to wait when requesting a new resource, then all other resources previously held by this process are implicitly released, (preempted), forcing this process to re-acquire the old resources along with the new resources in a single request, similar to the previous discussion.
- Another approach is that when a resource is requested and not available, then the system looks to see what other processes currently have those resources and are themselves blocked waiting for some other resource. If such a process is found, then some of their resources may get preempted and added to the list of resources for which the process is waiting.
- Either of these approaches may be applicable for resources whose states are easily saved and restored, such as registers and memory, but are generally not applicable to other devices such as printers and tape drives.

Circular Wait:

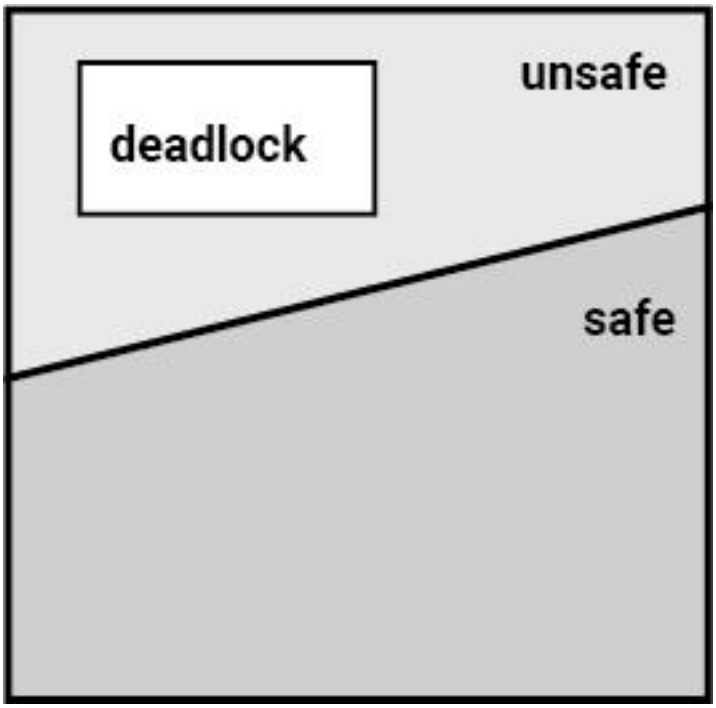
- One way to avoid circular wait is to number all resources, and to require that processes request resources only in strictly increasing (or decreasing) order.
- In other words, in order to request resource Rj, a process must first release all Ri such that i >= j.
- One big challenge in this scheme is determining the relative ordering of the different resources

Answer to the Question No- 3 (c)

Safe State:

- A state is safe if the system can allocate resources to each process (up to its maximum) in some order and still avoid a deadlock.
- More formally, a system is in a safe state only if there exists a safe sequence.
    - A sequence of processes $< P_1, P_2, P_3 \ldots\ldots, P_n >$ is a safe sequence for the current allocation state if, for each P_i the resource requests that P_i can still make can be satisfied by the currently available resources plus the resources held by all P_j, with j < i.
    - In this situation, if the resources that P_i needs are not immediately available, then P_i can wait until all P_j have finished.
    - When they have finished, P_i can obtain all of its needed resources, complete its designated task, return its allocated resources, and terminate.
    - When P_i terminates, P_{i+1} can obtain its needed resources, and so on.
    - If no such sequence exists, then the system state is said to be unsafe.
- A safe state is not a deadlocked state. Conversely, a deadlocked state is an unsafe state. Not all unsafe states are deadlocks.

Safe, unsafe, and deadlock state spaces-



Answer to the Question No- 5 (a)

Issues of Multithreading: Below we have mentioned a few issues related to multithreading. Well, it's an old saying, all good things, come at a price.

Thread Cancellation: Thread cancellation means terminating a thread before it has finished working. There can be two approaches for this, one is Asynchronous cancellation, which terminates the target thread immediately. The other is Deferred cancellation allows the target thread to periodically check if it should be cancelled.

Signal Handling: Signals are used in UNIX systems to notify a process that a particular event has occurred. Now in when a Multithreaded process receives a signal, to which thread it must be delivered? It can be delivered to all, or a single thread.

fork() System Call: fork() is a system call executed in the kernel through which a process creates a copy of itself. Now the problem in Multithreaded process is, if one thread forks, will the entire process be copied or not?

Security Issues: there can be security issues because of extensive sharing of resources between multiple threads.

There are many other issues that you might face in a multithreaded process, but there are appropriate solutions available for them. Pointing out some issues here was just to study both sides of the coin.

Answer to the Question No- 5 (b)

Semaphore:

- Synchronization tool that does not require busy waiting.
- Semaphore S–integer variable.
- Two standard operations modify S: wait() and signal()
    - Originally called P() and V()
- Less complicated.

- Can only be accessed via two indivisible (atomic) operations.

```
wait (S) {
while S <= 0
; // no-op
S--;
}

signal (S) {
S++;
}
```

The classical definitions of wait and signal are:

- **Wait:** Decrements the value of its argument S, as soon as it would become non-negative (greater than or equal to 1).
- **Signal:** Increments the value of its argument S, as there is no more process blocked on the queue.

Properties of Semaphore:

- It's simple and always have a non-negative Integer value.
- Works with many processes.
- Can have many different critical sections with different semaphores.
- Each critical section has unique access semaphores.
- Can permit multiple processes into the critical section at once, if desirable.

Answer to the Question No- 5 (c)

Worst-Fit: In this allocation technique, the process traverses the whole memory and always search for the largest hole/partition, and then the process is placed in that hole/partition. It is a slow process because it has to traverse the entire memory to search the largest hole.

Advantages of Worst-Fit Allocation: Since this process chooses the largest hole/partition, therefore there will be large internal fragmentation. Now, this internal fragmentation will be quite big so that other small processes can also be placed in that leftover partition.

Disadvantages of Worst-Fit Allocation: It is a slow process because it traverses all the partitions in the memory and then selects the largest partition among all the partitions, which is a time-consuming process.

>> END <<