Victoria University
of Bangladesh

Final Assessment

Md Bakhtiar Chowdhury

ID: 2121210061

Department: CSE

Semester: Summer -2022

Batch: 21th

Course Title: Operating System Concepts

Course Code: CSI 231

Submitted To:

Renea Chowdhury

Lecturer, Department of Computer Science & Engineering

Victoria University of Bangladesh

Submission Date: 08 October, 2022

Answer to the question no 1(a)

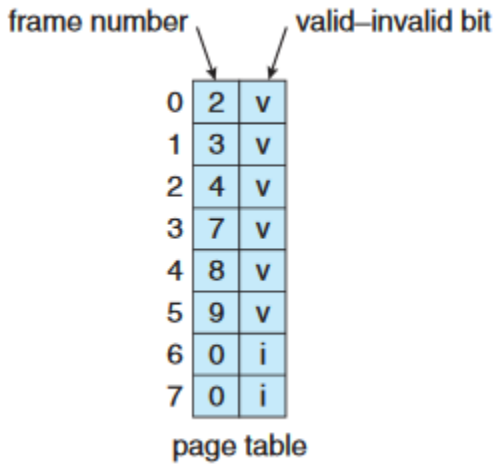
 # What is valid bit and invalid bit?

Answer:

valid bit : - A bit of information that indicates whether the data in a block is valid (1) or not (0).

a bit used in caches and virtual memories that records whether the cached item or page contains valid data.

indicates that the associated page is in the process' logical address space, and is thus a legal page



invalid bit: "invalid" indicates that the page is not in the process' logical address space

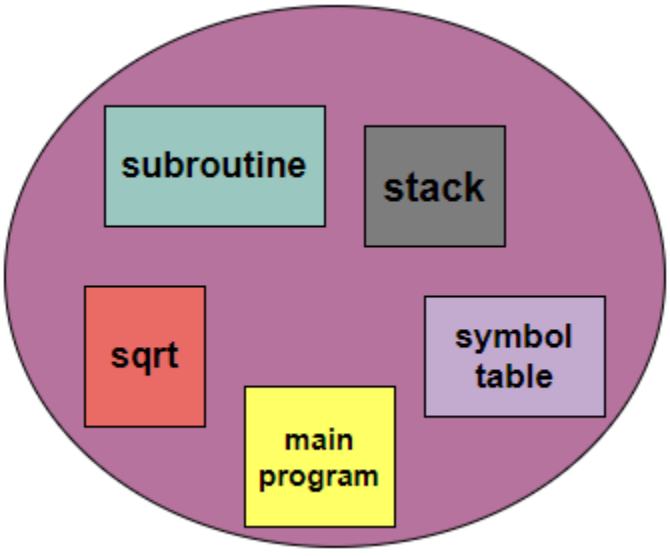
- With each page table entry, a valid–invalid bit is associated
- (1 is in-memory, 0 is not-in-memory)
- Initially, valid-invalid but is set to 0 on all entries.
- Example of a page table snapshot.

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

Answer to the question no 1(b)

 # Write down the architecture of Segmentation with an example.

Answer:



A Table that is used to store the information of all segments of the process is commonly known as Segment Table. Generally, there is no simple relationship between logical addresses and physical addresses in this scheme.

- The mapping of a two-dimensional Logical address into a one-dimensional Physical address is done using the segment table.
- This table is mainly stored as a separate segment in the main memory.
- The table that stores the base address of the segment table is commonly known as the Segment table base register (STBR)

In the segment table each entry has :

1. **Segment Base/base address:** The segment base mainly contain**s** the starting physical address where the segments reside in the memory.
2. **Segment Limit:** The segment limit is mainly used to specify the length of the segment.

Segment Table Base Register(STBR) The STBR register is used to point the segment table's location in the memory.

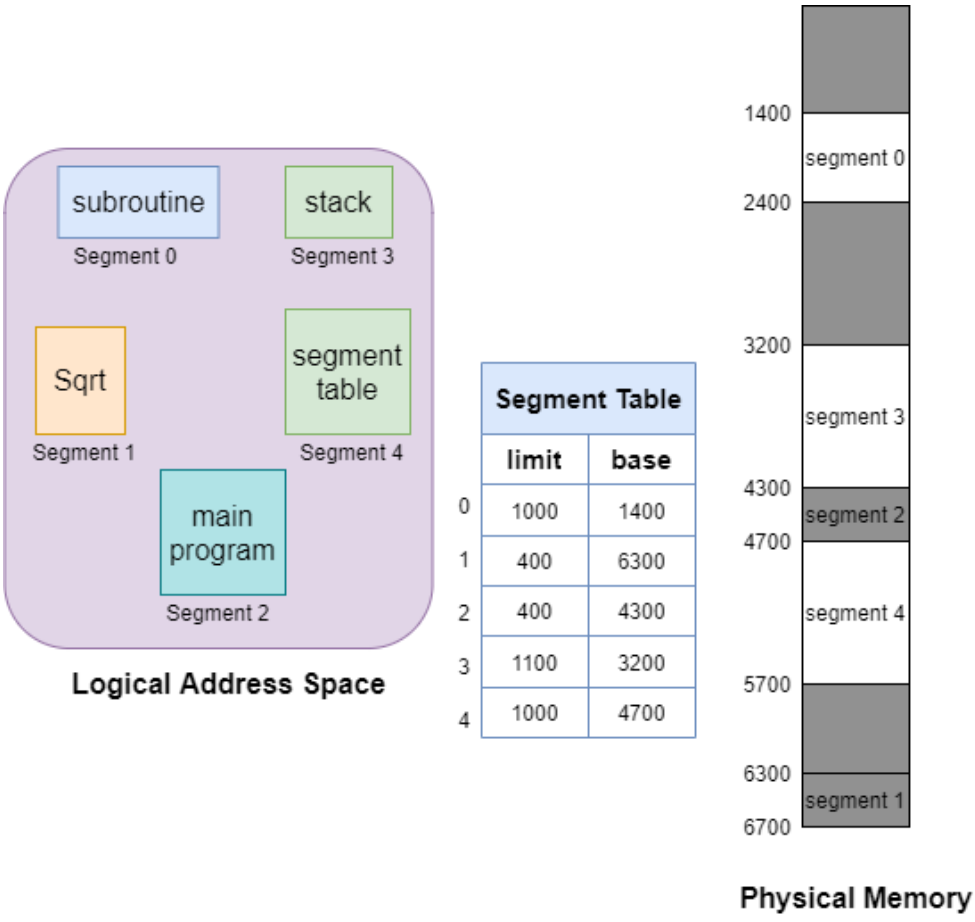
Segment Table Length Register(STLR) This register indicates the number of segments used by a program. The segment number s is legal if **s<STLR**

- Protection
- With each entry in segment table associate:
    - validation bit = 0 => illegal segment
    - read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level

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

Example of Segmentation

Given below is the example of the segmentation, There are five segments numbered from 0 to 4. These segments will be stored in Physical memory as shown. There is a separate entry for each segment in the segment table which contains the beginning entry address of the segment in the physical memory(denoted as the base) and also contains the length of the segment(denoted as limit).



Segment 2 is 400 bytes long and begins at location 4300. Thus in this case a reference to byte 53 of segment 2 is mapped onto the location 4300 (4300+53=4353). A reference to segment 3, byte 85 is mapped to 3200(the base of segment 3)+852=4052.

A reference to byte 1222 of segment 0 would result in the trap to the OS, as the length of this segment is 1000 bytes.

Answer to the question no 1(c)

 # What is Physical address?

Answer:

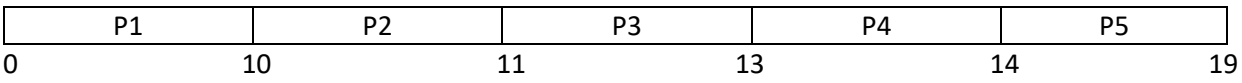
Physical Address: In computing, physical address refers to a memory address or the location of a memory cell in the main memory. It is used by both hardware and software for accessing data. Software, however, does not use physical addresses directly; instead, it accesses memory using a virtual address.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

 # Consider the following set of processes, with the length of CPU burst time given in 7 milliseconds.

1) FCFS



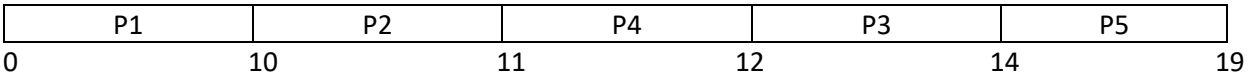
Job	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	10	10	10	0
P2	1	1	11	10	9
P3	2	2	13	11	9
P4	3	1	14	11	10
P5	4	5	19	15	10
Average				57 / 5 = 11.4	38 / 5 = 7.6

Average Waiting Time: (0+9+9+10+10)/5= 7.6

Average Finish/Completion Time: (10+11+13+14+19)/5=13.4

Average Turnaround Time: (10+10+11+11+15/5= 11.4

2) SJF- Non-preemptive



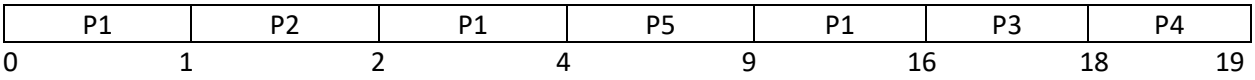
Job	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	10	10	10	0
P2	1	1	11	10	9
P3	2	2	14	12	10
P4	3	1	12	9	8
P5	4	5	19	15	10
Average				56 / 5 = 11.2	37 / 5 = 7.4

Average Waiting Time: (0+9+10+8+10)/5= 7.4

Average Finish/Completion Time: (10+11+14+12+19)/5=13.2

Average Turnaround Time: (10+10+12+9+15)/5= 11.2

3) Priority- Preemptive



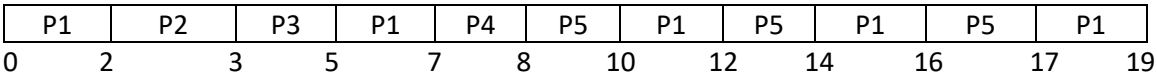
Job	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	10	16	16	6
P2	1	1	2	1	0
P3	2	2	18	16	14
P4	3	1	19	16	15
P5	4	5	9	5	0
Average				54 / 5 = 10.8	35 / 5 = 7

Average Waiting Time: (6+0+14+15+0)/5= 7

Average Finish/Completion Time: (16+2+18+19+9)/5=12.8

Average Turnaround Time: (16+1+16+16+5)/5= 10.8

4.1)**Round Robin with Time Quantum 2**



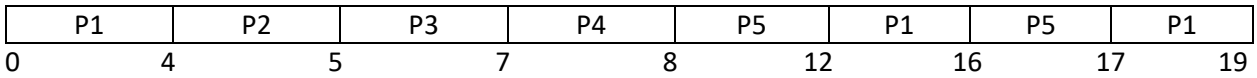
Job	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	10	19	19	9
P2	1	1	3	2	1
P3	2	2	5	3	1
P4	3	1	8	5	4
P5	4	5	17	13	8
Average				42 / 5 = 8.4	23 / 5 = 4.6

Average Waiting Time: (9 + 1 + 1+4+8)/5= 4.6

Average Finish/Completion Time: (19+3+5+8+17)/5=10.4

Average Turnaround Time: (19+2+3+5+13)/5= 8.4

4.2) **Round Robin with Time Quantum 4**



Job	Arrival Time	Burst Time	Completion Time	Turnaround Time	Waiting Time
P1	0	10	19	19	9
P2	1	1	5	4	3
P3	2	2	7	5	3
P4	3	1	8	5	4
P5	4	5	17	13	8
Average				46/ 5 = 9.2	27 / 5 = 5.4

Average Waiting Time: (9 + 3 + 3+4+8)/5= 5.4

Average Finish/Completion Time: (19+5+7+8+17)/5=11.2

Average Turnaround Time: (9+3+3+4+8)/5= 5.4

Answer to the question no 2(b)

 # Discuss the Multilevel feedback queue scheduler

Answer:

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.

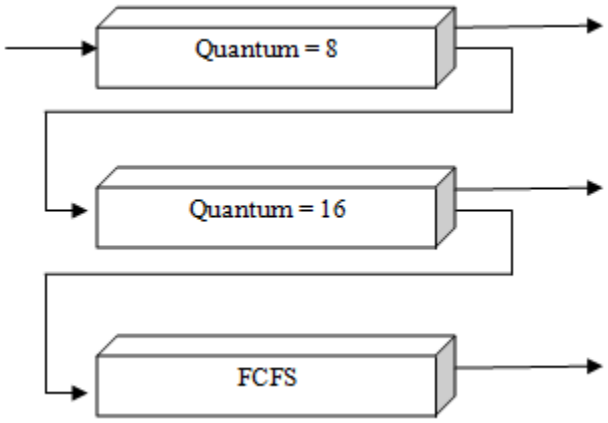
Example of Multilevel Feedback Queue

Three queues:

– Q0 – RR with time quantum 8 milliseconds

–Q1 – RR time quantum 16 milliseconds

–Q2 – FCFS



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)

 # Write down the characterization of Deadlock.

Answer:

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 $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that-

  P_0 is waiting for a resource that is held by P_1,

  P_1 is waiting for a resource that is held by P_2, …, P_n–1is waiting for a resource that is held by P_n, and P0is waiting for a resource that is held by P_0.

Answer to the question no 3(b)

 # Write down the steps to prevent Deadlock.

Answer:

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)

 # What is Safe state?

Answer:

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.

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

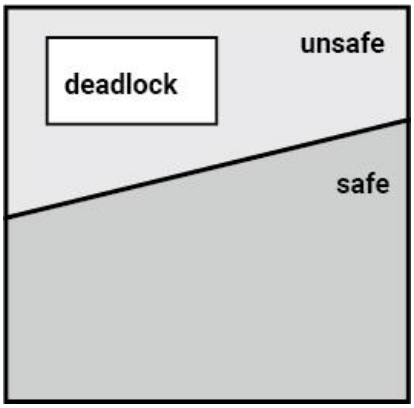
- System is in safe state if there exists a safe sequence of all processes.

- Sequence is safe if for each Pi , the resources that Pi can still request can be satisfied by currently available resources + resources held by all the Pj , with j<I

    - If Pi resource needs are not immediately available, then Pi can wait until all Pj have finished.

    - When Pj is finished, Pi can obtain needed resources, execute, return allocated resources, and terminate.

    - When Pi terminates, Pi+1 can obtain its needed resources, and so on.



Answer to the question no 5(a)

 # What are the issues of Multithreading?

Answer:

Multithreading is a model of program execution that allows for multiple threads to be created within a process, executing independently but concurrently sharing process resources. Depending on the hardware, threads can run fully parallel if they are distributed to their own CPU core.

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:

Yes, 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)

 # What is Semaphore? Write down the properties of Semaphore.

Answer:

Semaphore is simply a variable that is non-negative and shared between threads. A semaphore is a signaling mechanism, and a thread that is waiting on a semaphore can be signaled by another thread. It uses two atomic operations,

1) Wait

2) Signal for the process synchronization.

A semaphore either allows or disallows access to the resource, which depends on how it is set up.

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 always has a non-negative integer value (i.e., 0, 1, ...).
- It can be initialized to a non-negative integer value.
- A process can do a P operation (or "wait") on S, denoted P(S). P(S) atomically decrements S by 1 only when S>0 holds; the process waits until S>0 holds.
- A process can do a V operation (or "signal") on S, denoted V(S). V(S) atomically increments S by 1.

- Starvation-freedom: If a process is at P(S), then it eventually completes execution of the P(S) provided
    - S>0 holds continuously after some point in time, or
    - S>0 holds repeatedly (because V(S) is repeatedly done).
- 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)

 # What is Worst Fit?

Answer:

Worst Fit : Worst Fit allocates a process to the partition which is largest sufficient among the freely available partitions available in the main memory. If a large process comes at a later stage, then memory will not have space to accommodate it.

First-fit and best-fit better than worst-fit in terms of speed and storage utilization (according to simulations)

Worst fit works in the following way, for any given process P_n

The algorithms search sequentially starting from first memory block and searches for the memory block that fulfills the following condition –

- Can accommodate the process size
- Leaves the largest wasted space (fragmentation) after the process is allocated to given memory block

>>>>>END<<<<<