**Operating Systems**

**Assignment 1**

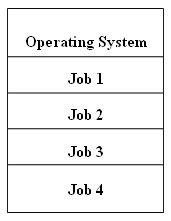
Sarthak Bhatia

07714803118

**Ques.1 How does multiprogramming increase CPU utilization?**

Early computers ran one process at a time. While the process waited for servicing by another device, the CPU was idle. In an I/O intensive process, the CPU could be idle as much as 80% of the time. Advancements in operating systems led to computers that load several independent processes into memory and switch the CPU from one job to another when the first becomes blocked while waiting for servicing by another device. This idea of multiprogramming reduces the idle time of the CPU. Multiprogramming accelerates the throughput of the system by efficiently using the CPU time.

Multiprogramming increases CPU utilization by organizing jobs (code and data) so that the CPU always has one to execute. The idea is as follows: The operating system keeps several jobs in memory simultaneously (Figure).



**Ques. 2 What are the two basic goal that must be considered when designing an operating system?**

Two types of goals of an Operating System i.e. Primary Goals and Secondary Goal.

* **Primary Goal:** The primary goal of an Operating System is to provide a user-friendly and convenient environment. We know that it is not compulsory to use the Operating System, but things become harder when the user has to perform all the process scheduling and converting the user code into machine code is also very difficult. So, we make the use of an Operating System to act as an intermediate between us and the hardware. All you need to do is give commands to the Operating System and the Operating System will do the rest for you. So, the Operating System should be convenient to use.
* **Secondary Goal:** The secondary goal of an Operating System is efficiency. The Operating System should perform all the management of resources in such a way that the resources are fully utilised and no resource should be held idle if some request to that resource is there at that instant of time.

**Ques. 3 In what circumstances are Overlays useful? When may a section of memory be overlayed? How does a overlaying effect program development time?**

Overlay is a technique to run a program that is bigger than the size of the physical memory by keeping only those instructions and data that are needed at any given time.

The main problem in Fixed partitioning is the size of a process has to be limited by the maximum size of the partition, which means a process can never be span over another. In order to solve this problem, earlier people have used some solution which is called as Overlays.

The concept of overlays is that whenever a process is running it will not use the complete program at the same time, it will use only some part of it. Then overlays concept says that whatever part you required, you load it an once the part is done, then you just unload it, means just pull it back and get the new part you required and run it.

Formally, “The process of transferringablock of program code or other data into internal memory, replacing what is already stored”. Sometimes it happens that compare to the size of the biggest partition, the size of the program will be even more, then, in that case, you should go with overlays.

So overlay is a technique to run a program that is bigger than the size of the physical memory by keeping only those instructions and data that are needed at any given time. Divide the program into modules in such a way that not all modules need to be in the memory at the same time.

**Ques.4 Dekker’s Algorithm, test And Set, Swap and the Semaphore P and V may all be used to enforce mutual exclusion. Compare and contrast their respective advantages and disadvantages.**

**Dekker's algorithm** was the first probably-correct solution to the critical section issue. It allows two threads to share a single-use resource without conflict, using just shared memory for communication. It avoids the strict alternation of a credulous turn-taking algorithm and was one of the first mutual exclusion algorithms to be invented. Although there are numerous versions of Dekker's Solution, the last or 5th version is the one that satisfies the entirety of the above conditions and is the most efficient of all. In this version, seasoned thread motion is used to determine entry to critical section. It provides mutual exclusion and dodging deadlock, indefinite postponement or lockstep synchronization by resolving the conflict that which thread should execute first. This version of Dekker's algorithm provides the complete solution of critical section problems.

**Test and Set Lock (TSL)** is a synchronization mechanism. It uses a test and set instruction to give the synchronization among the processes executing concurrently. It is an instruction that returns the old value of a memory location and sets the memory location value to 1 as a single atomic operation.

**Swapping i**s a mechanism in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason Swapping is also known as a technique for memory compaction.

**Semaphore P and V** was proposed by Dijkstra in 1965 which is a very significant technique to manage concurrent processes by using a simple integer value, which is known as a semaphore. Semaphore is simply a variable that is non-negative and shared between threads. This variable is used to solve the critical section problem and to achieve process synchronization in the multiprocessing environment.

**Limitations:**

1. One of the biggest limitations of semaphore is priority inversion.
2. Deadlock, suppose a process is trying to wake up another process which is not in a sleep state. Therefore, a deadlock may block indefinitely.
3. The operating system has to keep track of all calls to wait and to signal the semaphore.

**Ques. 5 Consider the 3 processes, P1, P2 and P3 shown in the table.**

**Process Arrival time Time Units Required**

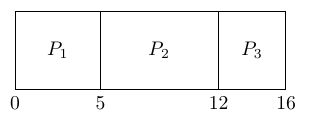
**P1 0 5**

**P2 1 7**

**P3 3 4**

**Write the completion order of the 3 processes under the policies FCFS and RR2 (round robin scheduling with CPU quantum of 2 time units) .**

**FCFS First Come First Server**



**RR2**

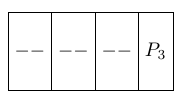
In Round Robin We are using the concept called Ready Queue.

Note at t=2,

P1 finishes and sent to Ready Queue.

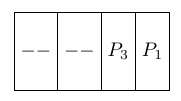
P2 arrives and schedules P2P2.

This is the Ready Queue



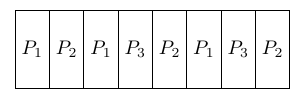
At t=3

P3 arrives at ready queue.



At t=4

P1 is scheduled as it is the first process to arrive at Ready Queue



FCFS: P1, P3, P2

RR2: P1, P2, P3

**Ques. 6 Consider the following table of arrival time and burst time for three processes P0, P1 and P2.**

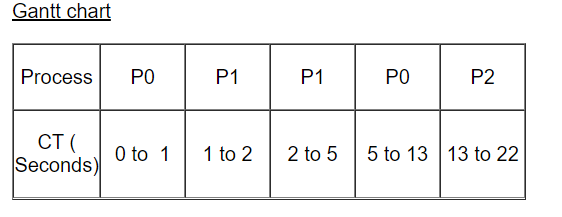
**Process Arrival time Burst Time**

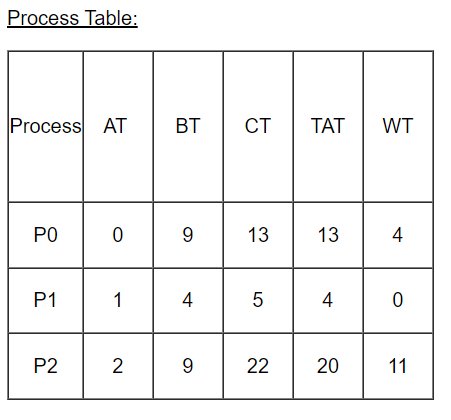
**P0 0 ms 9 ms**

**P1 1 ms 4 ms**

**P2 2 ms 9 ms**

**The pre-emptive shortest job first scheduling algorithm is used. Scheduling is carried out only at arrival or completion of processes. What is the average waiting time for the three processes?**





So average waiting time is (0+4+11) / 3 = 5ms.

**Ques. 7 What is the cause of thrashing? How does the system detect thrashing? Once it detects thrashing, what can the system do to eliminate the problem?**

Thrashing is caused by under allocation of the minimum number of pages required by a process, forcing it to continuously page fault.

The system can detect thrashing by evaluating the level of CPU utilization as compared to the level of multiprogramming. It can be eliminated by reducing the level of multiprogramming.

**Ques. 8 Explain Internal and external fragmentations. If memory partitions of 100k, 500k, 200k, 300K and 600K(in order) are given, how would each of the first fit, best fit, worst fit algorithm place the process of 212K,417k,112k and 426K(in order)? Which algorithm makes the most efficient use of memory?**

**Internal Fragmentation:**

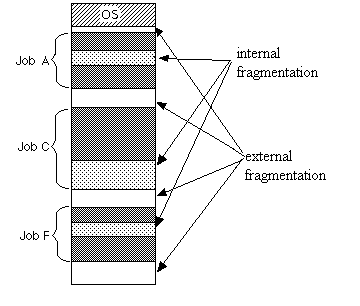
Internal fragmentation happens when the memory is split into mounted sized blocks. Whenever a method request for the memory, the mounted sized block is allotted to the method. just in case the memory allotted to the method is somewhat larger than the memory requested, then the distinction between allotted and requested memory is that the Internal fragmentation.

The above diagram clearly shows the internal fragmentation because the difference between memory allocated and required space or memory is called Internal fragmentation.

**External Fragmentation:** 

External fragmentation happens when there’s a sufficient quantity of area within the memory to satisfy the memory request of a method. however the process’s memory request cannot be fulfilled because the memory offered is during a non-contiguous manner. Either you apply first-fit or best-fit memory allocation strategy it’ll cause external fragmentation.

In above diagram, we can see that, there is enough space (55 KB) to run a process-07 (required 50 KB) but the memory (fragment) is not contiguous. Here, we use compaction, paging or segmentation to use the free space to run a process.



**First-Fit:**

212K is put in 500K partition.

417K is put in 600K partition.

112K is put in 288K partition (new partition 288K = 500K - 212K).

426K must wait.

**Best-Fit:**

212K is put in 300K partition.

417K is put in 500K partition.

112K is put in 200K partition.

426K is put in 600K partition.

**Worst-Fit:**

212K is put in 600K partition.

417K is put in 500K partition.

112K is put in 388K partition.

426K must wait.

**Best-Fit** is more efficient for this example in the sense that in this case Best-Fit is able to satisfy all requests without waiting while the other algorithms cannot.

**Ques. 9 Assume an average page-fault service time is 25 milliseconds and a memory access time is 100 nanoseconds. Find the Effective Access Time?**

Effective Access Time (EAT)= (1 – p) \* (ma) + p \* (page fault time)

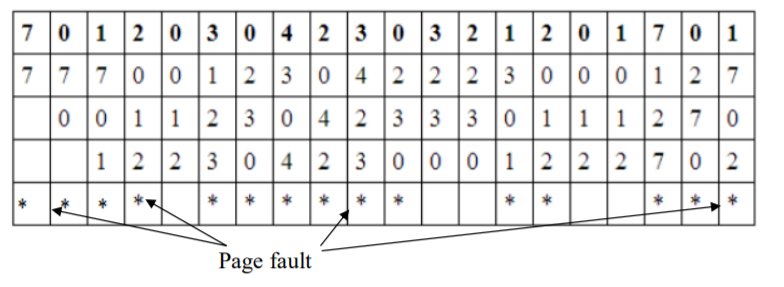
= (1 – p) \* 100 + p \* 25,000,000

= 100 – 100 \* p + 25,000,000 \* p

= 100 + 24,999,900 \*p

**Ques. 10 Consider the following page reference using three frames that are initially empty. Find the page faults using FIFO algorithm, where the page reference sequence:**

**7,0,1, 2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1?**



The page fault = 15.

**Ques. 11 How can measure the performance of demand paging?**

The performance of demand paging is often measured in terms of the effective access time. Effective access time is the amount of time it takes to access memory, if the cost of page faults are amortized over all memory accesses. In some sense it is an average or expected access time.

ea = (1 - p) \* ma + p\*pft

where

ea = effective access time  
ma = physical memory (core) access time  
pft = page fault time  
p = probability of a page fault occuring  
(1-p) = the probability of accessing memory in an available frame

The page fault time is the sum of the additional overhead associated with accessing a page in the backing store. This includes additional context switches, disk latency and transfer time associated with page-in and page-out operations, the overhead of executing an operating system trap, &c.

**Ques. 12 How can the system distinguish between the pages that are in main memory from the pages that are on the disk?**

The system uses valid-invalid bit is used. This bit is set to "valid" when the page in memory, while it set to "invalid" when the page either not valid or is the page is valid but is on the disk, as in the following figure.

