Last Minute Notes – Operating Systems - GeeksQuiz

Operating Systems: It is the interface between the user and the computer hardware.

Type's of OS:

- **Batch OS:** A set of similar jobs are stored in the main memory for execution. A job gets assigned to the CPU, only when the execution of the previous job completes.
- **Multiprogramming OS:** The main memory consists of jobs waiting for CPU time. The OS selects one of the processes and assigns it the CPU time. Whenever the executing process needs to wait for any other operation (like I/O), the OS selects another process from the job queue and assigns it the CPU. This way, the CPU is never kept idle and the user gets the flavor of getting multiple tasks done at once.
- **Multitasking OS:** Multitasking OS combines the benefits of Multiprogramming OS and CPU scheduling to perform quick switches between jobs. The switch is so quick that the user can interact with each program as it runs
- **Time Sharing OS:** Time sharing systems require interaction with the user to instruct the OS to perform various tasks. The OS responds with an output. The instructions are usually given through an input device like the keyboard.
- **Real Time OS :** Real Time OS are usually built for dedicated systems to accomplish a specific set of tasks within deadlines.

Threads

A thread is a light weight process and forms a basic unit of CPU utilization. A process can perform more than one task at the same time by including multiple threads.

  - A thread has its own program counter, register set, and stack
  - A thread shares with other threads of the same process the code section, the data section, files and signals.

A new thread, or a child process of a given process, can be introduced by using the fork() system call. A process with n fork() system calls generates 2^n – 1 child processes.
There are two types of threads:

  - User threads
  - Kernel threads

Example : Java thread, POSIX threads.Example : Window Solaris.

User level thread	**Kernel level thread**
User thread are implemented by users.	kernel threads are implemented by OS.
OS doesn't recognized user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complicated.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Hardware support is needed.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread can continue execution.

Process:

A process is a program under execution. The value of program counter (PC) indicates the address of the current instruction of the process being executed. Each process is represented by a Process Control Block (PCB).
Process Scheduling:

Below are different time with respect to a process.

```
Arrival Time:       Time at which the process arrives in the ready queue.
Completion Time:    Time at which process completes its execution.
Burst Time:         Time required by a process for CPU execution.
Turn Around Time:   Time Difference between completion time and arrival time.
     Turn Around Time = Completion Time - Arrival Time

Waiting Time(W.T): Time Difference between turn around time and burst time.
     Waiting Time = Turn Around Time - Burst Time
```

Why do we need scheduling?
A typical process involves both I/O time and CPU time. In a uniprogramming system like MS-DOS, time spent waiting for I/O is wasted and CPU is free during this time. In multiprogramming systems, one process can use CPU while another is waiting for I/O. This is possible only with process scheduling.

Objectives of Process Scheduling Algorithm

```
Max CPU utilization [Keep CPU as busy as possible]
Fair allocation of CPU.
Max throughput [Number of processes that complete their execution per time unit]
Min turnaround time [Time taken by a process to finish execution]
Min waiting time [Time a process waits in ready queue]
Min response time [Time when a process produces first response]
```

Different Scheduling Algorithms
First Come First Serve (FCFS): Simplest scheduling algorithm that schedules according to arrival times of processes.

Shortest Job First(SJF): Process which have the shortest burst time are scheduled first.

Shortest Remaining Time First(SRTF): It is preemptive mode of SJF algorithm in which jobs are schedule according to shortest remaining time.

Round Robin Scheduling: Each process is assigned a fixed time in cyclic way.

Priority Based scheduling (Non Preemptive): In this scheduling, processes are scheduled according to their priorities, i.e., highest priority process is schedule first. If priorities of two processes match, then schedule according to arrival time.

Highest Response Ratio Next (HRRN) In this scheduling, processes with highest response ratio is scheduled. This algorithm avoids starvation.

```
Response Ratio = (Waiting Time + Burst time) / Burst time
```

Multilevel Queue Scheduling: According to the priority of process, processes are placed in the different queues. Generally high priority process are placed in the top level queue. Only after completion of processes from top level queue, lower level queued processes are scheduled.

Multi level Feedback Queue Scheduling: It allows the process to move in between queues. The idea is to separate processes according to the characteristics of their CPU bursts. If a process uses too much CPU time, it is moved to a lower-priority queue.

Some useful facts about Scheduling Algorithms:
1) FCFS can cause long waiting times, especially when the first job takes too much CPU time.

2) Both SJF and Shortest Remaining time first algorithms may cause starvation. Consider a situation when long process is there in ready queue and shorter processes keep coming.

3) If time quantum for Round Robin scheduling is very large, then it behaves same as FCFS scheduling.

4) SJF is optimal in terms of average waiting time for a given set of processes. SJF gives minimum average waiting time, but problems with SJF is how to know/predict time of next job.

The Critical Section Problem

Critical Section: The portion of the code in the program where shared variables are accessed and/or updated.
Remainder Section: The remaining portion of the program excluding the Critical Section.

Race around Condition: The final output of the code depends on the order in which the variables are accessed. This is termed as the race around condition.

A solution for the critical section problem must satisfy the following three conditions:

1. **Mutual Exclusion:** If a process Pi is executing in its critical section, then no other process is allowed to enter into the critical section.
2. **Progress:** If no process is executing in the critical section, then the decision of a process to enter a critical section cannot be made by any other process that is executing in its remainder section. The selection of the process cannot be postponed indefinitely.
3. **Bounded Waiting:** There exists a bound on the number of times other processes can enter into the critical section after a process has made request to access the critical section and before the requested is granted.

Synchronization Tools

Semaphores: A semaphore is an integer variable that is accessed only through two atomic operations, wait () and signal (). An atomic operation is executed in a single CPU time slice without any pre-emption.
Semaphores are of two types:

1. **Counting Semaphore:** A counting semaphore is an integer variable whose value can range over an unrestricted domain.
2. **Mutex:** Binary Semaphores are called Mutex. These can have only two values, 0 or 1. The operations wait () and signal () operate on these in a similar fashion.

   Deadlock
   A situation where a set of processes are blocked because each process is holding a resource and waiting for another resource acquired by some other process.
   Deadlock can arise if following four conditions hold simultaneously (Necessary Conditions)
   Mutual Exclusion: One or more than one resource are non-sharable (Only one process can use at a time)
   Hold and Wait: A process is holding at least one resource and waiting for resources.

No Preemption: A resource cannot be taken from a process unless the process releases the resource.
Circular Wait: A set of processes are waiting for each other in circular form.

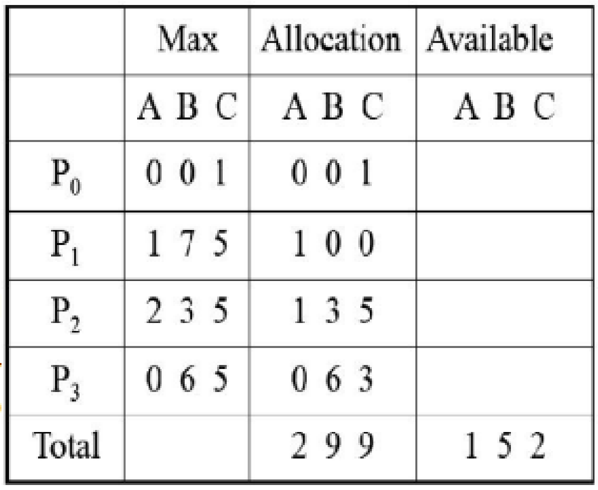
Methods for handling deadlock
There are three ways to handle deadlock
1) Deadlock prevention or avoidance: The idea is to not let the system into deadlock state.

2) Deadlock detection and recovery: Let deadlock occur, then do preemption to handle it once occurred.

3) Ignore the problem all together: If deadlock is very rare, then let it happen and reboot the system. This is the approach that both Windows and UNIX take.

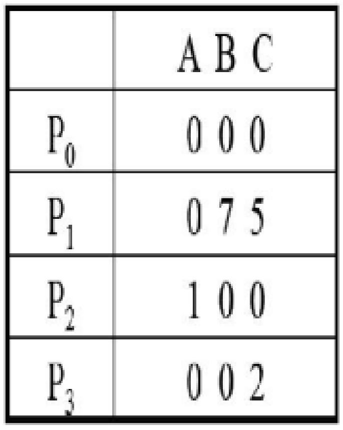
Banker's Algorithm:
This algorithm handles multiple instances of the same resource.

Example: The snapshot of the system at a given instant:



	Max	Allocation	Available
	A B C	A B C	A B C
P_0	0 0 1	0 0 1	
P_1	1 7 5	1 0 0	
P_2	2 3 5	1 3 5	
P_3	0 6 5	0 6 3	
Total		2 9 9	1 5 2

Need = Max - Allocation.



	A B C
P_0	0 0 0
P_1	0 7 5
P_2	1 0 0
P_3	0 0 2

Memory Management:
These techniques allow the memory to be shared among multiple processes.

Overlays: The memory should contain only those instructions and data that are required at a given time.

Swapping: In a multiprogramming program, the instructions that have used the time slice are swapped out from the memory.

Memory Management Techniques:
1: Single Partition Allocation Schemes: The memory is divided into two parts. One part is kept for use by the OS and the other for use by the users.

2: Multiple Partition Schemes:
Fixed Partition: The memory is divided into fixed size partitions.
Variable Partition: The memory is divided into variable sized partitions.

Variable partition allocation schemes:
First Fit: The arriving process is allotted the first hole of memory in which it fits completely.
Best Fit: The arriving process is allotted the hole of memory in which it fits the best by leaving the minimum memory empty.

Worst Fit: The arriving process is allotted the hole of memory in which it leaves the maximum gap. Note: Best fit does necessarily give the best results for memory allocation.

1. Paging: The physical memory is divided into equal sized frames. The main memory is divided into fixed size pages. The size of a physical memory frame is equal to the size of a virtual memory frame.

2. Segmentation: Segmentation is implemented to give users view of memory. The logical address space is a collection of segments. Segmentation can be implemented with or without the use of paging.

Page Fault

A page fault is a type of interrupt, raised by the hardware when a running program accesses a memory page that is mapped into the virtual address space, but not loaded in physical memory.

Page Replacement Algorithms

First In First Out

This is the simplest page replacement algorithm. In this algorithm, operating system keeps track of all pages in the memory in a queue, oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

For example, consider page reference string 1, 3, 0, 3, 5, 6 and 3 page slots.

Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**
when 3 comes, it is already in memory so —> **0 Page Faults.**
Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.**
Finally 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.**

Belady's anomaly

Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3 2 1 0 3 2 4 3 2 1 0 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

Optimal Page replacement

In this algorithm, pages are replaced which are not used for the longest duration of time in the future.

Let us consider page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 and 4 page slots.

Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already there so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**
0 is already there so —> **0 Page fault.**.
4 will takes place of 1 —> **1 Page Fault.**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

Least Recently Used
In this algorithm page will be replaced which is least recently used.

Let say the page reference string 7 0 1 2 0 3 0 4 2 3 0 3 2 . Initially we have 4 page slots empty.
Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
0 is already their so —> **0 Page fault.**
when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
0 is already in memory so —> **0 Page fault**.
4 will takes place of 1 —> **1 Page Fault**
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

This article has been contributed by Pushp Sra.

Please comment below if you find anything wrong in the above post or you wish to add something related to the topic.

See Placement Course for placement preparation, GATE Corner for GATE CS Preparation and Quiz Corner for all Quizzes on GeeksQuiz.