BIJU PATNAIK UNIVERSITY OF TECHNOLOGY, ODISHA

Short Type

Questions and Answers

on

OPERATING SYSTEM

Prepared by,

Dr. Subhendu Kumar Rath,

BPUT, Odisha.

OPERATING SYSTEM (SHORT QUESTION AND ANSWERS)

By Dr.S.K.Rath, BPUT

1. Explain the concept of Reentrancy?

It is a useful, memory-saving technique for multiprogrammed timesharing systems. A Reentrant Procedure is one in which multiple users can share a single copy of a program during the same period. Reentrancy has 2 key aspects: The program code cannot modify itself, and the local data for each user process must be stored separately. Thus, the permanent part is the code, and the temporary part is the pointer back to the calling program and local variables used by that program. Each execution instance is called activation. It executes the code in the permanent part, but has its own copy of local variables/parameters. The temporary part associated with each activation is the activation record. Generally, the activation record is kept on the stack.

Note: A reentrant procedure can be interrupted and called by an interrupting program, and still execute correctly on returning to the procedure.

2. Explain Belady's Anomaly?

Also called FIFO anomaly. Usually, on increasing the number of frames allocated to a process virtual memory, the process execution is faster, because fewer page faults occur. Sometimes, the reverse happens, i.e., the execution time increases even when more frames are allocated to the process. This is Belady's Anomaly. This is true for certain page reference patterns.

3. What is a binary semaphore? What is its use?

A binary semaphore is one, which takes only 0 and 1 as values. They are used to implement mutual exclusion and synchronize concurrent processes.

4. What is thrashing?

It is a phenomenon in virtual memory schemes when the processor spends most of its time swapping pages, rather than executing instructions. This is due to an inordinate number of page faults.

5. List the Coffman's conditions that lead to a deadlock.

1. **Mutual Exclusion**: Only one process may use a critical resource at a time.
2. **Hold & Wait**: A process may be allocated some resources while waiting for others.
3. **No Pre-emption**: No resource can be forcible removed from a process holding it.
4. **Circular Wait**: A closed chain of processes exist such that each process holds at least one resource needed by another process in the chain.

6. What are short, long and medium-term scheduling?

Long term scheduler determines which programs are admitted to the system for processing. It controls the degree of multiprogramming. Once admitted, a job becomes a process.

Medium term scheduling is part of the swapping function. This relates to processes that are in a blocked or suspended state. They are swapped out of real-memory until they are ready to execute. The swapping-in decision is based on memory-management criteria.

Short term scheduler, also know as a dispatcher executes most frequently, and makes the finest-grained decision of which process should execute next. This scheduler is invoked whenever an event occurs. It may lead to interruption of one process by preemption.

7. What are turnaround time and response time?

Turnaround time is the interval between the submission of a job and its completion. Response time is the interval between submission of a request, and the first response to that request.

8. What are the typical elements of a process image?

User data: Modifiable part of user space. May include program data, user stack area, and programs that may be modified.

User program: The instructions to be executed.

System Stack: Each process has one or more LIFO stacks associated with it. Used to store parameters and calling addresses for procedure and system calls.

Process control Block (PCB): Info needed by the OS to control processes.

9. What is the Translation Lookaside Buffer (TLB)?

In a cached system, the base addresses of the last few referenced pages is maintained in registers called the TLB that aids in faster lookup. TLB contains those page-table entries that have been most recently used. Normally, each virtual memory reference causes 2 physical memory accesses- one to fetch appropriate page-table entry, and one to fetch the desired data. Using TLB in-between, this is reduced to just one physical memory access in cases of TLB-hit.

10. What is the resident set and working set of a process?

Resident set is that portion of the process image that is actually in real-memory at a particular instant. Working set is that subset of resident set that is actually needed for execution. (Relate this to the variable-window size method for swapping techniques.)

11. When is a system in safe state?

The set of dispatchable processes is in a safe state if there exists at least one temporal order in which all processes can be run to completion without resulting in a deadlock.

12. What is cycle stealing?

We encounter cycle stealing in the context of Direct Memory Access (DMA). Either the DMA controller can use the data bus when the CPU does not need it, or it may force the CPU to temporarily suspend operation. The latter technique is called cycle stealing. Note that cycle stealing can be done only at specific break points in an instruction cycle.

13. What is meant by arm-stickiness?

If one or a few processes have a high access rate to data on one track of a storage disk, then they may monopolize the device by repeated requests to that track. This generally happens with most common device scheduling algorithms (LIFO, SSTF, C-SCAN, etc). High-density multisurface disks are more likely to be affected by this than low density ones.

14. What are the stipulations of C2 level security?

C2 level security provides for:

1. Discretionary Access Control
2. Identification and Authentication
3. Auditing
4. Resource reuse

15. What is busy waiting?

The repeated execution of a loop of code while waiting for an event to occur is called busy-waiting. The CPU is not engaged in any real productive activity during this period, and the process does not progress toward completion.

16. Explain the popular multiprocessor thread-scheduling strategies.

1. **Load Sharing**: Processes are not assigned to a particular processor. A global queue of threads is maintained. Each processor, when idle, selects a thread from this queue. Note that load balancing refers to a scheme where work is allocated to processors on a more permanent basis.
2. **Gang Scheduling**: A set of related threads is scheduled to run on a set of processors at the same time, on a 1-to-1 basis. Closely related threads / processes may be scheduled this way to reduce synchronization blocking, and minimize process switching. Group scheduling predated this strategy.
3. **Dedicated processor assignment**: Provides implicit scheduling defined by assignment of threads to processors. For the duration of program execution, each program is allocated a set of processors equal in number to the number of threads in the program. Processors are chosen from the available pool.
4. **Dynamic scheduling**: The number of thread in a program can be altered during the course of execution.

17. When does the condition 'rendezvous' arise?

In message passing, it is the condition in which, both, the sender and receiver are blocked until the message is delivered.

18. What is a trap and trapdoor?

Trapdoor is a secret undocumented entry point into a program used to grant access without normal methods of access authentication. A trap is a software interrupt, usually the result of an error condition.

19. What are local and global page replacements?

Local replacement means that an incoming page is brought in only to the relevant process address space. Global replacement policy allows any page frame from any process to be replaced. The latter is applicable to variable partitions model only.

20. Define latency, transfer and seek time with respect to disk I/O.

Seek time is the time required to move the disk arm to the required track. Rotational delay or latency is the time it takes for the beginning of the required sector to reach the head. Sum of seek time (if any) and latency is the access time. Time taken to actually transfer a span of data is transfer time.

21. Describe the Buddy system of memory allocation.

Free memory is maintained in linked lists, each of equal sized blocks. Any such block is of size 2^k. When some memory is required by a process, the block size of next higher order is chosen, and broken into two. Note that the two such pieces differ in address only in their kth bit. Such pieces are called buddies. When any used block is freed, the OS checks to see if its buddy is also free. If so, it is rejoined, and put into the original free-block linked-list.

22. What is time-stamping?

It is a technique proposed by Lamport, used to order events in a distributed system without the use of clocks. This scheme is intended to order events consisting of the transmission of messages. Each system 'i' in the network maintains a counter Ci. Every time a system transmits a message, it increments its counter by 1 and attaches the time-stamp Ti to the message. When a message is received, the receiving system 'j' sets its counter Cj to 1 more than the maximum of its current value and the incoming time-stamp Ti. At each site, the ordering of messages is determined by the following rules: For messages x from site i and y from site j, x precedes y if one of the following conditions holds....(a) if Ti<Tj or (b) if Ti=Tj and i<j.

23. How are the wait/signal operations for monitor different from those for semaphores?

If a process in a monitor signal and no task is waiting on the condition variable, the signal is lost. So this allows easier program design. Whereas in semaphores, every operation affects the value of the semaphore, so the wait and signal operations should be perfectly balanced in the program.

24. In the context of memory management, what are placement and replacement algorithms?

Placement algorithms determine where in available real-memory to load a program. Common methods are first-fit, next-fit, best-fit. Replacement algorithms are used when memory is full, and one process (or part of a process) needs to be swapped out to accommodate a new program. The replacement algorithm determines which are the partitions to be swapped out.

25. In loading programs into memory, what is the difference between load-time dynamic linking and run-time dynamic linking?

For **load-time dynamic linking**: Load module to be loaded is read into memory. Any reference to a target external module causes that module to be loaded and the references are updated to a relative address from the start base address of the application module.

With **run-time dynamic loading**: Some of the linking is postponed until actual reference during execution. Then the correct module is loaded and linked.

26. What are demand-paging and pre-paging?

With demand paging, a page is brought into memory only when a location on that page is actually referenced during execution. With pre-paging, pages other than the one demanded by a page fault are brought in. The selection of such pages is done based on common access patterns, especially for secondary memory devices.

27. Paging a memory management function, while multiprogramming a processor management function, are the two interdependent?

Yes.

28. What is page cannibalizing?

Page swapping or page replacements are called page cannibalizing.

29. What has triggered the need for multitasking in PCs?

1. Increased speed and memory capacity of microprocessors together with the support fir virtual memory and
2. Growth of client server computing

30. What are the four layers that Windows NT have in order to achieve independence?

1. Hardware abstraction layer
2. Kernel
3. Subsystems
4. System Services.

31. What is SMP?

To achieve maximum efficiency and reliability a mode of operation known as symmetric multiprocessing is used. In essence, with SMP any process or threads can be assigned to any processor.

32. What are the key object oriented concepts used by Windows NT?

Encapsulation, Object class and instance.

33. Is Windows NT a full blown object oriented operating system? Give reasons.

No Windows NT is not so, because its not implemented in object oriented language and the data structures reside within one executive component and are not represented as objects and it does not support object oriented capabilities.

34. What is a drawback of MVT?

It does not have the features like

1. ability to support multiple processors
2. virtual storage
3. source level debugging

35. What is process spawning?

When the OS at the explicit request of another process creates a process, this action is called process spawning.

36. How many jobs can be run concurrently on MVT?

15 jobs.

37. List out some reasons for process termination.

1. Normal completion
2. Time limit exceeded
3. Memory unavailable
4. Bounds violation
5. Protection error
6. Arithmetic error
7. Time overrun
8. I/O failure
9. Invalid instruction
10. Privileged instruction
11. Data misuse
12. Operator or OS intervention
13. Parent termination.

38. What are the reasons for process suspension?

1. swapping
2. interactive user request
3. timing
4. parent process request

39. What is process migration?

It is the transfer of sufficient amount of the state of process from one machine to the target machine.

40. What is mutant?

In Windows NT a mutant provides kernel mode or user mode mutual exclusion with the notion of ownership.

41. What is an idle thread?

The special thread a dispatcher will execute when no ready thread is found.

42. What is FtDisk?

It is a fault tolerance disk driver for Windows NT.

43. What are the possible threads a thread can have?

1. Ready
2. Standby
3. Running
4. Waiting
5. Transition
6. Terminated

44. What are rings in Windows NT?

Windows NT uses protection mechanism called rings provides by the process to implement separation between the user mode and kernel mode.

45. What is Executive in Windows NT?

In Windows NT, executive refers to the operating system code that runs in kernel mode.

46. What are the sub-components of I/O manager in Windows NT?

1. Network redirector/ Server
2. Cache manager.
3. File systems
4. Network driver
5. Device driver

47. What are DDks? Name an operating system that includes this feature.

DDks are device driver kits, which are equivalent to SDKs for writing device drivers. Windows NT includes DDks.

48. What level of security does Windows NT meets?

C2 level security.

Previous Year Question and Answer
Subject- Operating System

Q.1)(a) What is Throughput, Turnaround time, Waiting time and Response time?

Ans: **Throughput** is defined as the number of processes that complete their execution per unit time.

Turnaround time is the amount of time to execute a particular process.

Waiting time is the amount of time a process has been waiting in the ready queue.

Response time is the amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

(b) What is Reentrancy?

Ans: A computer program or subroutine is called **reentrant** if it can be interrupted in the middle of its execution and then safely called again ("re-entered") before its previous invocations complete execution. The interruption could be caused by an internal action such as a jump or call, or by an external action such as a hardware interrupt or signal. Once the reentered invocation completes, the previous invocations will resume correct execution.

(c) What is the difference between Hard and Soft real time Systems?

Ans: A **hard real-time** system guarantees that critical tasks be completed on time. This goal requires that all delays in the system be bounded, from the retrieval of stored data to the time that it takes the operating system to finish any request made of it. Such time constraints dictate the facilities that are available in hard real-time systems.

A **soft real-time** system is a less restrictive type of real-time system. Here, a critical real-time task gets priority over other tasks and retains that priority until it completes. Soft real time system can be mixed with other types of systems. Due to less restriction, they are risky to use for industrial control and robotics.

(d) What resources are used when a thread created? How do the differ from those when a process is created?

Ans: Because a thread is smaller than a process, thread creation typically uses fewer resources than process creation. Creating a process requires allocating a process control block (PCB), a rather large data structure. The PCB includes a memory map, list of open files, and environment variables. Allocating and managing the memory map is typically the most time-consuming activity. Creating either a user or kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

(e) What is a binary semaphore? What is its use?

Ans: A binary semaphore is a semaphore with an integer value that can range only between 0 and 1. A binary semaphore can be simpler to implement than a counting semaphore, depending on the underlying hardware architecture.

(f) What is the difference between synchronization and mutual exclusion?

Ans: If one process is executing in its critical section then no other process is allowed to enter it critical section. This is called **mutual exclusion**. **Synchronization** refers to one of two distinct but related concepts: synchronization of processes, and synchronization of data. **Process synchronization** refers to the idea that multiple processes are to join up or handshake at a certain point, in order to reach an agreement or commit to a certain sequence of action. **Data synchronization** refers to the idea of keeping multiple copies of a dataset in coherence with one another, or to maintain data integrity. Process synchronization primitives are commonly used to implement data synchronization.

g) List the Coffman's conditions that lead to a deadlock.

Ans: A deadlock situation can arise if and only if all of the following conditions hold simultaneously in a system:

1. **Mutual Exclusion**: At least one resource must be non-shareable. Only one process can use the resource at any given instant of time.
2. **Hold and Wait or Resource Holding**: A process is currently holding at least one resource and requesting additional resources which are being held by other processes.
3. **No Preemption**: The operating system must not de-allocate resources once they have been allocated; they must be released by the holding process voluntarily.
4. **Circular Wait**: A process must be waiting for a resource which is being held by another process, which in turn is waiting for the first process to release the resource. In general, there is a set of waiting processes, P = {P1, P2, ..., PN}, such that P1 is waiting for a resource held by P2, P2 is waiting for a resource held by P3 and so on till PN is waiting for a resource held by P1.

These four conditions are known as the Coffman conditions from their first description in a 1971 article by Edward G. Coffman.

h)Explain Belady's Anomaly.

Ans:

2)(a) Name the three types of schedulers and give functions of each.

Ans: Three types of CPU schedulers are FCFS, SJF and Priority scheduling.

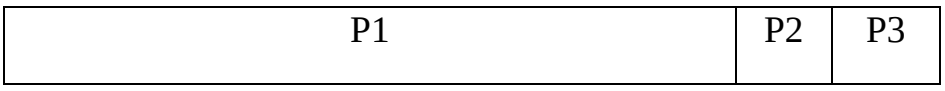
First-Come, First-Served Scheduling

By far the simplest CPU-scheduling algorithm is the first-come, first-served (FCFS) scheduling algorithm. With this scheme, the process that requests the CPU first is allocated the CPU first. The implementation of the FCFS policy is easily managed with a FIFO queue. When a process enters the ready queue, its PCB is linked onto the tail of the queue. When the CPU is free, it is allocated to the process at the head of the queue. The running process is then removed from the queue. The code for FCFS scheduling is simple to write and understand. The average waiting time under the FCFS policy, however, is often quite long.

Consider the following set of processes that arrive at time 0, with the length of the CPU-burst time given in milliseconds:

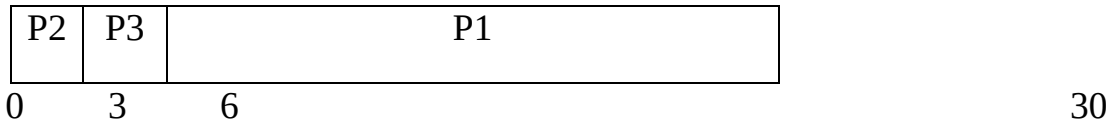
Process	Burst Time
PI	24
P2	3
P3	3

If the processes arrive in the order PI, P2, P3, and are served in FCFS order, we get the result shown in the following Gantt chart:



0 24 27 30

The waiting time is 0 milliseconds for process PI, 24 milliseconds for process PZ, and 27 milliseconds for process P3. Thus, the average waiting time is (0 + 24 + 27)/3 = 17 milliseconds. If the processes arrive in the order P2, P3, Pl, however, the results will be as shown in the following Gantt chart:



The average waiting time is now (6 + 0 + 3)/3 = 3 milliseconds. This reduction is substantial. Thus, the average waiting time under a FCFS policy is generally not minimal, and may vary substantially if the process CPU-burst times vary greatly.

In addition, consider the performance of FCFS scheduling in a dynamic situation. Assume we have one CPU-bound process and many I/O-bound processes. As the processes flow around the system, the following scenario may result. The CPU-bound process will get the CPU and hold it. During this time, all the other processes will finish their I/O and move into the ready queue, waiting for the CPU. While the processes wait in the ready queue, the I/O devices are idle. Eventually, the CPU-bound process finishes its CPU burst and moves to an I/O device. All the I/O-bound processes, which have very short CPU bursts, execute quickly and move, back to the I/O queues. At this point, the CPU sits idle.

The CPU-bound process will then move back to the ready queue and be allocated the CPU. Again, all the I/O processes end up waiting in the ready queue until the CPU-bound process is done. There is a convoy effect, as all the other processes wait for the one big process to get off the CPU. This effect results in lower CPU and device utilization than might be possible if the shorter processes were allowed to go first.

The FCFS scheduling algorithm is non-preemptive. Once the CPU has been allocated to a process, that process keeps the CPU until it releases the CPU, either by terminating or by requesting I/O. The FCFS algorithm is particularly troublesome for time-sharing systems, where each user needs to get a share of the CPU at regular intervals. It would be disastrous to allow one process to keep the CPU for an extended period.

Shortest-Job-First Scheduling

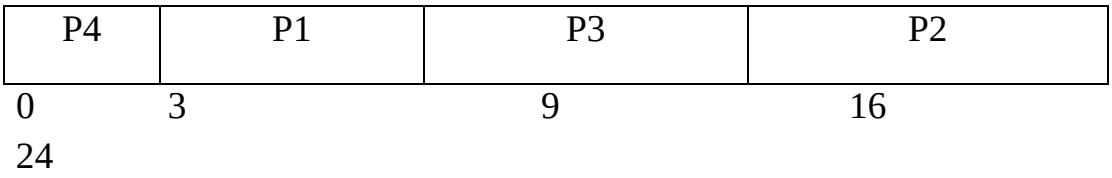
A different approach to CPU scheduling is the shortest-job-first (SJF) scheduling algorithm. This algorithm associates with each process the length of the latter's next CPU burst. When the CPU is available, it is assigned to the process that has the smallest next CPU burst. If two processes have the same length next

CPU burst, FCFS scheduling is used to break the tie. Note that a more appropriate term would be the shortest next CPU burst, because the scheduling is done by examining the length of the next CPU burst of a process, rather than its total length. We use the term SJF because most people and textbooks refer to this type of scheduling discipline as SJF.

As an example, consider the following set of processes, with the length of the CPU-burst time given in milliseconds:

Process	Burst Time
PI	6
p2	8
p3	7
p4	3

Using SJF scheduling, we would schedule these processes according to the following Gantt chart:



The waiting time is 3 milliseconds for process PI, 16 milliseconds for process P2,9 milliseconds for process PS, and 0 milliseconds for process Pq. Thus, the average waiting time is (3 + 16 + 9 + 0)/4 = 7 milliseconds. If we were using the FCFS scheduling scheme, then the average waiting time would be 10.25 milliseconds.

The SJF scheduling algorithm is provably optimal, in that it gives the minimum average waiting time for a given set of processes. By moving a short process before a long one, the waiting time of the short process decreases more than it increases the waiting time of the long process. Consequently, the average waiting time decreases.

The real difficulty with the SJF algorithm is knowing the length of the next CPU request. For long-term (or job) scheduling in a batch system, we can use as the length the process time limit that a user specifies when he submits the job.

Thus, users are motivated to estimate the process time limit accurately, since a lower value may mean faster response. (Too low a value will cause a time-limit exceeded error and require resubmission.) SJF scheduling is used frequently in long-term scheduling.

Although the SJF algorithm is optimal, it cannot be implemented at the level of short-term CPU scheduling. There is no way to know the length of the next CPU burst. One approach is to try to approximate SJF scheduling. We may not know the length of the next CPU burst, but we may be able to predict its value.We expect that the next CPU burst will be similar in length to the previous ones.

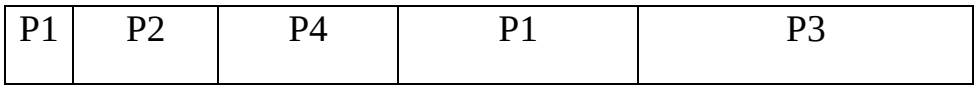
Thus, by computing an approximation of the length of the next CPU burst, we can pick the process with the shortest predicted CPU burst.

The SJF algorithm may be either preemptive or nonpreemptive. The choice arises when a new process arrives at the ready queue while a previous process is executing. The new process may have a shorter next CPU burst than what is left of the currently executing process. A preemptive SJF algorithm will preempt the currently executing process, whereas a nonpreemptive SJF algorithm will allow the currently running process to finish its CPU burst. Preemptive SJF scheduling is sometimes called shortest-remaining-time-first scheduling.

As an example, consider the following four processes, with the length of the CPU-burst time given in milliseconds:

Process	Arrival Time	Burst Time
P1	0	8
P2	1	4
P3	2	9
p4	3	5

If the processes arrive at the ready queue at the times shown and need the indicated burst times, then the resulting preemptive SJF schedule is as depicted in the following Gantt chart:



0 1 5 10 17 26

Process PI is started at time 0, since it is the only process in the queue. Process PZ arrives at time 1. The remaining time for process P1 (7 milliseconds) is larger than the time required by process P2 (4 milliseconds), so process P1 is preempted, and process P2 is scheduled. The average waiting time for this example is ((10 - 1) + (1 - 1) + (17 - 2) + (5 - 3))/4 = 26/4 = 6.5 milliseconds. A nonpreemptive SJF scheduling would result in an average waiting time of 7.75 milliseconds.

Priority Scheduling

The SJF algorithm is a special case of the general priority-scheduling algorithm. A priority is associated with each process, and the CPU is allocated to the process with the highest priority. Equal-priority processes are scheduled in FCFS order.

An SJF algorithm is simply a priority algorithm where the priority (p) is the inverse of the (predicted) next CPU burst. The larger the CPU burst, the lower the priority, and vice versa.

Note that we discuss scheduling in terms of high priority and low priority. Priorities are generally some fixed range of numbers, such as 0 to 7, or 0 to 4,095. However, there is no general agreement on whether 0 is the highest or lowest priority. Some systems use low numbers to represent low priority; others use low numbers for high priority. This difference can lead to confusion. In this text, we use low numbers to represent high priority. As an example, consider the following set of processes, assumed to have arrived at time 0, in the order PI, P2, ..., P5, with the length of the CPU-burst time given in milliseconds:

Process	Burst	Time Priority
P1	10	3
p2	1	1
p3	2	4
P4	1	5
P5	5	2

Using priority scheduling, we would schedule these processes according to the following Gantt chart:

0 1 6 16 18 19

The average waiting time is 8.2 milliseconds. Priorities can be defined either internally or externally. Internally defined priorities use some measurable quantity or quantities to compute the priority of a process.

(b) Give the queuing diagram representing process scheduling and show the action point for the different types of CPU schedulers.

Ans: A common representation for a discussion of process scheduling is a queuing diagram.

  - Each rectangular box represents a queue. Two types of queues are present: the ready queue and a set of device queues.
  - The circles represent the resources that serve the queues, and the arrows indicate the flow of processes in the system.
- A new process is initially put in the ready queue. It waits there until it is selected for execution, or is dispatched.
- Once the process is allocated the CPU and is executing, one of several events could occur:
  - The process could issue an I/O request and then be placed in an I/O queue.
  - The process could create a new subprocess and wait for the subprocess's termination.

- The process could be removed forcibly from the CPU, as a result of an interrupt, and be put back in the ready queue.

- A process continues this cycle until it terminates, at which time it is removed from all queues and has its PCB and resources deallocated.

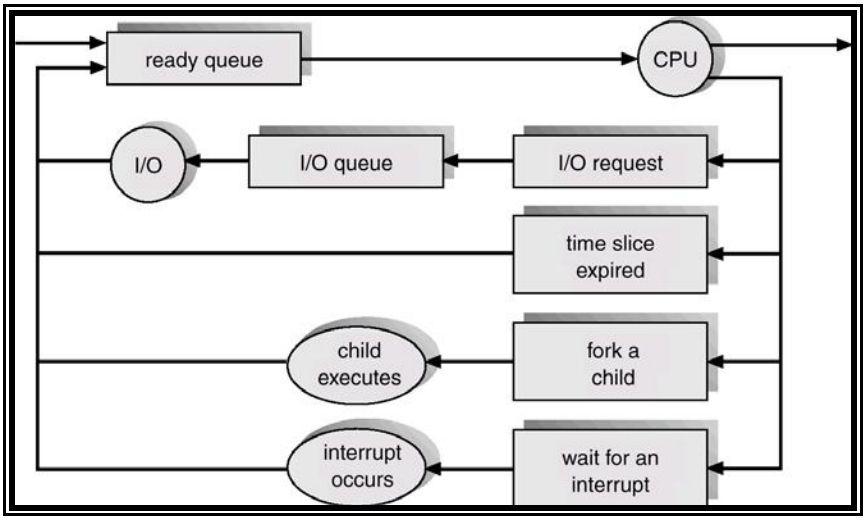


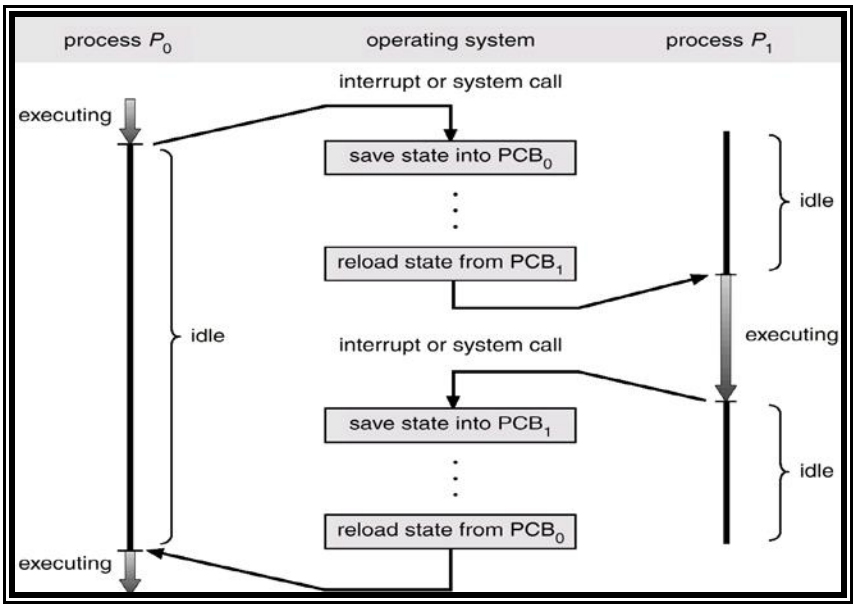
Fig.-Queuing diagram representation of process scheduling

3)(a)Explain Context switch along with a suitable diagram.Name and briefly define the scheduling criteria. List out the circumstances under which a CPU scheduling decision may have to be taken by the CPU scheduler.

Ans: Switching the CPU to another process requires saving the state of the old process and loading the saved state for the new process. This task is known as a context switch. The context of a process is represented in the PCB of a process; it includes the value of the CPU registers, the process state and memory-management information. When a context switch occurs, the kernel saves the context of the old process in its PCB and loads the saved context of the new process scheduled to run.

Context-switch time is pure overhead, because the system does no useful work while switching. Its speed varies from machine to machine, depending on the memory speed, the number of registers that must be copied, and the existence of special instructions (such as a single instruction to load or store all registers). Typical speeds range from 1 to 1000 microseconds.

Context -switch times are highly dependent on hardware support. For instance, some processors (such as the Sun UltraSPARC) provide multiple sets of registers. A context switch simply includes changing the pointer to the current register set. Of course, if active processes exceed register sets, the system resorts to copying register data to and from memory, as before. Also, the more complex the operating system, the more work must be done during a context switch.



Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – Number of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

(b) Assume, we have the workload as shown below. All 5 processes arrive at time 0, in the order given below. The length of the CPU burst time is given in milliseconds

Process : P1 P2 P3 P4 P5

Burst time : 10 29 3 7 12

Considering the FCFS, SJF and RR (q=10 ms) scheduling algorithms, which algorithm would give the minimum average turnaround time.

Ans: The Gantt-chart for FCFS scheduling is

P1	P2	P3	P4	P5

0 10 39 42 49 61

Turnaround time = Finished Time – Arrival Time

Turnaround time for process P1 = 10 – 0 = 10

Turnaround time for process P2 = 39 – 0 = 39

Turnaround time for process P3 = 42 – 0 = 42

Turnaround time for process P4 = 49 – 0 = 49

Turnaround time for process P5 = 61 – 0 = 61

Average Turnaround time = (10+39+42+49+61)/5 = 40.2

The Gantt-chart for SJF scheduling is

P3	P4	P1	P5	P2

0 3 10 20 32 61

Turnaround time for process P1 = 3 – 0 = 3

Turnaround time for process P2 = 10 – 0 = 10

Turnaround time for process P3 = 20 – 0 = 20

Turnaround time for process P4 = 32 – 0 = 32

Turnaround time for process P5 = 61 – 0 = 61

Average Turnaround time = (3+10+20+32+61)/5 = 25.2

The Gantt-chart for RR scheduling is

P1	P2	P3	P4	P5	P2	P5	P2

0 10 20 23 30 40 50 52 61

Turnaround time for process P1 = 10 – 0 = 10

Turnaround time for process P2 = 61 – 0 = 61

Turnaround time for process P3 = 23 – 0 = 23

Turnaround time for process P4 = 30 – 0 = 30

Turnaround time for process P5 = 52 – 0 = 52

Average Turnaround time = (10+61+23+30+52)/5 = 44.2

So SJF gives minimum turnaround time.

Previous year Question and Answer
Subject- Operating System

Q.1)(a) What do you mean by graceful degradation in multiprocessor systems?

Ans: The ability to continue providing service proportional to the level of surviving hardware is called **graceful degradation.**

(b) Define system call.

Ans: System calls provide an interface between the process and the Operating System. System calls allow user-level processes to request some services from the operating system which process itself is not allowed to do.

(c) List the three requirements that must be satisfied by critical section problem.

Ans: A solution to the critical-section problem must satisfy the following three requirements:

1. Mutual Exclusion: If process Pi is executing in its critical section, then no other processes can be executing in their critical sections.

2. Progress: If no process is executing in its critical section and some processes wish to enter their critical sections, then only those processes that are not executing in their remainder section can participate in the decision on which will enter its critical section next, and this selection cannot be postponed indefinitely.

3. Bounded Waiting: There exists a bound on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.

(i) What is kernel of an Operating System?

Ans: Kernel is an active part of an OS i.e., it is the part of OS running at all times. It is a programs which can interact with the hardware. Ex: Device driver, dll files, system files etc.

3) Consider the following set of processes, with the length of CPU-burst time given in milliseconds:

Process	**Burst Time (ms)**	**Priority**
P1	**1**	**1**

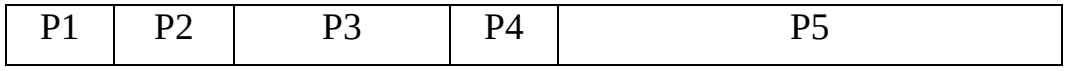
P2	**1**	**1**
P3	**2**	**3**
P4	**1**	**4**
P5	**5**	**2**

The processes are assumed to have arrived in order P1, P2, P3, P4, P5 all at time 0.

(a) Draw Gantt charts illustrating the execution of these processes using FCFS, SJF, a non-preemptive priority (a smaller priority implies a higher priority) and RR (quantum=1) scheduling.

(b) What is the turnaround time of each process for each of the scheduling algorithms in part (a).

Ans: The Gantt-chart for FCFS scheduling is



Turnaround time = Finished Time – Arrival Time

Turnaround time for process P1 = 1 – 0 = 1

Turnaround time for process P2 = 2 – 0 = 2

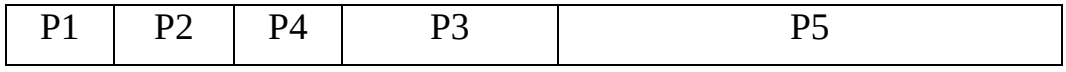
Turnaround time for process P3 = 4 – 0 = 4

Turnaround time for process P4 = 5 – 0 = 5

Turnaround time for process P5 = 10 – 0 = 10

Average Turnaround time = (1+2+4+5+10)/5 = 4.4

The Gantt-chart for SJF scheduling is



Turnaround time for process P1 = 1 – 0 = 1

Turnaround time for process P2 = 2 – 0 = 2

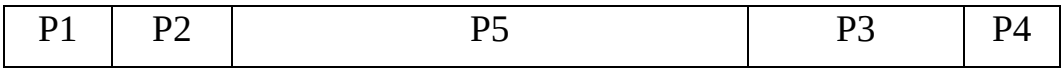
Turnaround time for process P3 = 5 – 0 = 5

Turnaround time for process P4 = 3 – 0 = 3

Turnaround time for process P5 = 10 – 0 = 10

Average Turnaround time = (1+2+5+3+10)/5 = 4.2

The Gantt-chart for Priority scheduling is



Turnaround time for process P1 = 1 – 0 = 1

Turnaround time for process P2 = 2 – 0 = 2

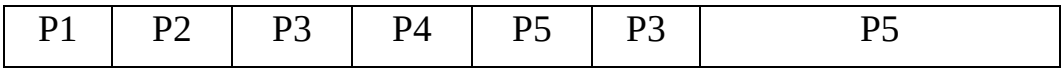
Turnaround time for process P3 = 9 – 0 = 9

Turnaround time for process P4 = 10 – 0 = 10

Turnaround time for process P5 = 7 – 0 = 7

Average Turnaround time = (1+2+9+10+7)/5 = 5.8

The Gantt-chart for RR scheduling is



Turnaround time for process P1 = 1 – 0 = 1

Turnaround time for process P2 = 2 – 0 = 2

Turnaround time for process P3 = 6 – 0 = 6

Turnaround time for process P4 = 4 – 0 = 4

Turnaround time for process P5 = 10 – 0 = 10

Average Turnaround time = (1+2+6+4+10)/5 = 4.8

6) (b) What is a thread? Why is it that threads are faster to create than processes? What advantages do kernel threads provide over user threads?

Ans: A thread is a single sequence stream within in a process. It is also called lightweight processes. In a process, threads allow multiple executions of streams. The CPU switches rapidly back and forth among the threads giving illusion that the threads are running in parallel. A thread can be in any of several states (Running, Blocked, Ready or Terminated).

An operating system that has thread facility, the basic unit of CPU utilization is a thread. A thread has or consists of a program counter (PC), a register set, and a stack space. Threads are not independent of one other like processes as a result

threads shares with other threads their code section, data section, OS resources such as open files and signals.

Because a thread is smaller than a process, thread creation typically uses fewer resources than process creation. Creating a process requires allocating a process control block (PCB), a rather large data structure. The PCB includes a memory map, list of open files, and environment variables. Allocating and managing the memory map is typically the most time-consuming activity. Creating either a user or kernel thread involves allocating a small data structure to hold a register set, stack, and priority.

Advantage:

- Because kernel has full knowledge of all threads, Scheduler may decide to give more time to a process having large number of threads than process having small number of threads.
- Kernel-level threads are especially good for applications that frequently block.

8) (a) Explain Readers-Writers problem using semaphores.

Ans: A data object (such as a file or record) is to be shared among several concurrent processes. Some of these processes may want only to read the content of the shared object, whereas others may want to update (that is, to read and write) the shared object. We distinguish between these two types of processes by referring to those processes that are interested in only reading as readers, and to the rest as writers. Obviously, if two readers access the shared data object simultaneously, no adverse effects will result. However, if a writer and some other process (either a reader or a writer) access the shared object simultaneously, chaos may ensue.

To ensure that these difficulties do not arise, we require that the writers have exclusive access to the shared object. This synchronization problem is referred to as the readers-writers problem. Since it was originally stated, it has been used to test nearly every new synchronization primitive. The readers-writers problem has several variations, all involving priorities. The simplest one, referred to as the first readers-writers problem, requires that no reader will be kept waiting unless a writer has already obtained permission to use the shared object. In other words, no reader should wait for other readers to finish simply because a writer is waiting. The second readers-writers problem requires that, once a writer is ready, that writer performs its write as soon as possible. In other words, if a writer is waiting to access the object, no new readers may start reading.

A solution to either problem may result in starvation. In the first case, writers may starve; in the second case, readers may starve. For this reason, other variants of the problem have been proposed. In this section, we present a solution to the first readers-writers problem.

In the solution to the first readers-writers problem, the reader processes

share the following data structures:

semaphore mutex, wrt;

int readcount;

The semaphores mutex and wrt are initialized to 1; readcount is initialized to 0. The semaphore w r t is common to both the reader and writer processes. The mutex semaphore is used to ensure mutual exclusion when the variable readcount is updated. The readcount variable keeps track of how many processes are currently reading the object. The semaphore wrt functions as a mutual-exclusion semaphore for the writers. It is also used by the first or last reader that enters or exits the critical section. It is not used by readers who enter or exit while other readers are in their critical sections.

The code for a writer process is

```
do{

wait (wrt) ;

. . .

writing is performed

...

signal(wrt);

}while(1);
```

The code for a reader process is

```
do{
```

```
wait (mutex) ;

readcount++;

if (readcount == 1)

wait (wrt) ;

signal (mutex) ;

. . .

reading is performed

...

wait (mutex) ;

readcount--;

if (readcount == 0)

signal(wrt1;

signal (mutex) ;

}while(1);
```

Note that, if a writer is in the critical section and n readers are waiting, then one reader is queued on wrt, and n - 1 readers are queued on mutex. Also observe that, when a writer executes signal (wrt), we may resume the execution of either the waiting readers or a single waiting writer.

9)(c) Write a note on Semaphore.

Ans: A semaphore S is an integer variable that, apart from initialization, is accessed only through two standard atomic operations: wait and signal. These operations were originally termed P (for wait; from the Dutch proberen, to test) and V (for signal; from verhogen, to increment). The classical definition of wait in pseudocode is

```
wait(S) {
```

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

The classical definitions of signal in pseudocode is

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

(d) Write a short note on Peterson's solution.

Ans: Peterson's solution is a software based solution to the critical section problem.

Consider two processes P0 and P1. For convenience, when presenting Pi, we use Pi to denote the other process; that is, j == 1 - i.

The processes share two variables:

```
boolean flag [2] ;
int turn;
```

Initially flag [0] = flag [1] = false, and the value of turn is immaterial (but is either 0 or 1). The structure of process Pi is shown below.

```
do{
    flag[i]=true
    turn=j
    while(flag[j] && turn==j);
        critical section
    flag[i]=false
```

Remainder section

}while(1);

To enter the critical section, process Pi first sets flag [il to be true and then sets turn to the value j, thereby asserting that if the other process wishes to enter the critical section it can do so. If both processes try to enter at the same time, turn will be set to both i and j at roughly the same time. Only one of these assignments will last; the other will occur, but will be overwritten immediately. The eventual value of turn decides which of the two processes is allowed to enter its critical section first.

We now prove that this solution is correct. We need to show that:

1. Mutual exclusion is preserved,

2. The progress requirement is satisfied,

3. The bounded-waiting requirement is met.

To prove property 1, we note that each Pi enters its critical section only if either flag [jl == false or turn == i. Also note that, if both processes can be executing in their critical sections at the same time, then flag [i] ==flag [jl == true. These two observations imply that P0 and P1 could not have successfully executed their while statements at about the same time, since the value of turn can be either 0 or 1, but cannot be both. Hence, one of the processes say Pj-must have successfully executed the while statement, whereas Pi had to execute at least one additional statement ("turn == j"). However, since, at that time, flag [j] == true, and turn == j, and this condition will persist as long as Pi is in its critical section, the result follows:

To prove properties 2 and 3, we note that a process Pi can be prevented from entering the critical section only if it is stuck in the while loop with the condition flag [j] == true and turn == j; this loop is the only one. If Pi is not ready to enter the critical section, then flag [j] == false and Pi can enter its critical section. If Pi has set flag[j] to true and is also executing in its while statement, then either turn == i or turn == j. If turn == i, then Pi will enter the critical section. If turn == j, then Pi will enter the critical section. However, once Pi exits its critical section, it will

reset flag [jl to false, allowing Pi to enter its critical section. If Pi resets flag [j 1 to true, it must also set turn to i.

Thus, since Pi does not change the value of the variable turn while executing the while statement, Pi will enter the critical section (progress) after at most one entry by Pi (bounded waiting).

----THANK YOU-----