1. What is a critical region? How do they relate to controlling access to shared resources?

Ans: A critical region is a section of code in which a shared resources is accessed.

To control access to the shared resource, access is controlled to the critical region of code. By controlling the code that accessed the resource we can control access to the resource.

2. Explain why Windows, Linux, and Solaris implement multiple locking mechanisms. Describe the circumstances under which they use spinlocks, mutex locks, semaphores, adaptive mutex locks, and condition variables

ANS: These operating systems provide different locking mechanisms depending on the application developers' needs. Spinlocks are useful for multiprocessor systems where a thread can run in a busy-loop (for a short period of time) rather than incurring the overhead of being put in a sleep queue. Mutexes are useful for locking resources. Solaris 2 uses adaptive mutexes, meaning that the mutex is implemented with a spin lock on multiprocessor machines. Semaphores and condition variables are more appropriate tools for synchronization when a resource must be held for a long period of time, since spinning is inefficient for a long duration.

3. Explain the Dining-Philosophers problem with a simple solution. How deadlocks are

handled in that solution?

The Dining Philosopher Problem – The Dining Philosopher Problem states that K philosophers seated around a circular table with one chopstick between each pair of philosophers. There is one chopstick between each philosopher. A philosopher may eat if he can pick up the two chopsticks adjacent to him. One chopstick may be picked up by any one of its adjacent followers but not both.

Semaphore Solution to Dining Philosopher –

Each philosopher is represented by the following pseudocode:

```
process P[i]
 while true do
  {  THINK;
     PICKUP(CHOPSTICK[i], CHOPSTICK[i+1 mod 5]);
     EAT;
     PUTDOWN(CHOPSTICK[i], CHOPSTICK[i+1 mod 5])
  }
```

The philosopher can only use the chopstick on his or her immediate left or right.

The philosophers never speak to each other, which creates a dangerous possibility of

deadlock. Deadlock could occur if every philosopher holds a left chopstick and waits

perpetually for a right chopstick (or vice versa).

4. Explain the difference between preemptive and non-preemptive scheduling.

Distinguish between PCS and SCS scheduling.

1.

BASIS FOR COMPARISON	**PREEMPTIVE SCHEDULING**	**NON PREEMPTIVE SCHEDULING**
Basic	The resources are allocated to a process for a limited time.	Once resources are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Interrupt	Process can be interrupted in between.	Process can not be interrupted till it terminates or switches to waiting state.
Starvation	If a high priority process frequently arrives in the ready queue, low priority process may starve.	If a process with long burst time is running CPU, then another process with less CPU burst time may starve.
Overhead	Preemptive scheduling has overheads of scheduling the processes.	Non-preemptive scheduling does not have overheads.
Flexibility	Preemptive scheduling is flexible.	Non-preemptive scheduling is rigid.
Cost	Preemptive scheduling is cost associated.	Non-preemptive scheduling is not cost associative.
Example	FCFS	Round Robin

Explain the difference between preemptive and nonpreemptive scheduling.

Answer:

Preemptive scheduling allows a process to be interrupted in the midst of its execution, taking the CPU away and allocating it to another process. Nonpreemptive scheduling ensures that a process relinquishes control of the CPU only when it finishes with its current CPU burst.

Distinguish between PCS and SCS scheduling.

Answer:

PCS scheduling is done local to the process. It is how the thread library schedules threads onto available LWPs. SCS scheduling is the situation where the operating system schedules kernel threads. On systems using either many-to-one or many-to-many, the two scheduling models are fundamentally different. On systems using one-to-one, PCS and SCS are the same.

5. What is processor affinity? Why it might improve performance? Which of the following scheduling algorithms could result in starvation?a. First-come, first-served b. Shortest job first c. Round robin d. Priority

Processor affinity is also known as **"cache affinity"**. It can bind and unbind a process or a thread to a CPU or a group of CPUs and ensure that the process or thread executes only on the designated CPU and not on any other CPU. It is a modification of the symmetric multiprocessing OS's native central queue scheduling algorithm. Every item in the queue does have a tag that specifies its kin processor. When it comes to resource allocation, each task is assigned to its kin processor before others.

Processor Affinity is useful if you have a heavy program like video rendering. When you dedicated a core for the video editing program, it ensures that the core of the processor is always dedicated to the task. It improves performance because it reduces reduce cache problems as there is no delay with a dedicated core.

Shortest job first and priority-based scheduling algorithms could result in starvation.

1. **SJF** is a non-preemptive or preemptive scheduling algorithm.The processor should know in advance how much time the process will take. This is a best approach to minimize waiting time.
2. The **starvation** in SJF exist only if the process with Lower Burst Time appears in queue before the process with Higher Burst time is executed, So then the algorithm will always choose the process with lowest Burst Time, the process with higher Burst Time will never be able to get the share of CPU. Sothat it is very difficult to find starvation problem in SJF.
3. SJF is the most optimal scheduling algorithm.The real difficulty with SJF is **knowing the length of the next CPU request**
4. SJF algorithm is simply a priority algorithm where the priority is the predicted next CPU burst.
5. **Priority scheduling** is a non-preemptive algorithm.

6. Each process is assigned a priority. Process with highest priority is to be executed first and so on. Processes with same priority are executed on first come first served basis. Priority can be decided based on memory requirements, time requirements.
7. In **priority-based scheduling** algorithms, a major problem is **starvation**. A process that is ready to run but waiting for the CPU can be considered blocked. A **priority scheduling** algorithm can leave some low-priority processes waiting indefinitely.

6. Suppose that the following processes arrive for execution at the times indicated. Each

process will run for the amount of time listed. In answering the questions, use

nonpreemptive scheduling, and base all decisions on the information you have at the

time the decision must be made.

Process	Arrival Time	Burst Time
P1	0.0	8
P2	0.4	4
P3	1.0	1

a. What is the average turnaround time for these processes with the FCFS scheduling

algorithm?

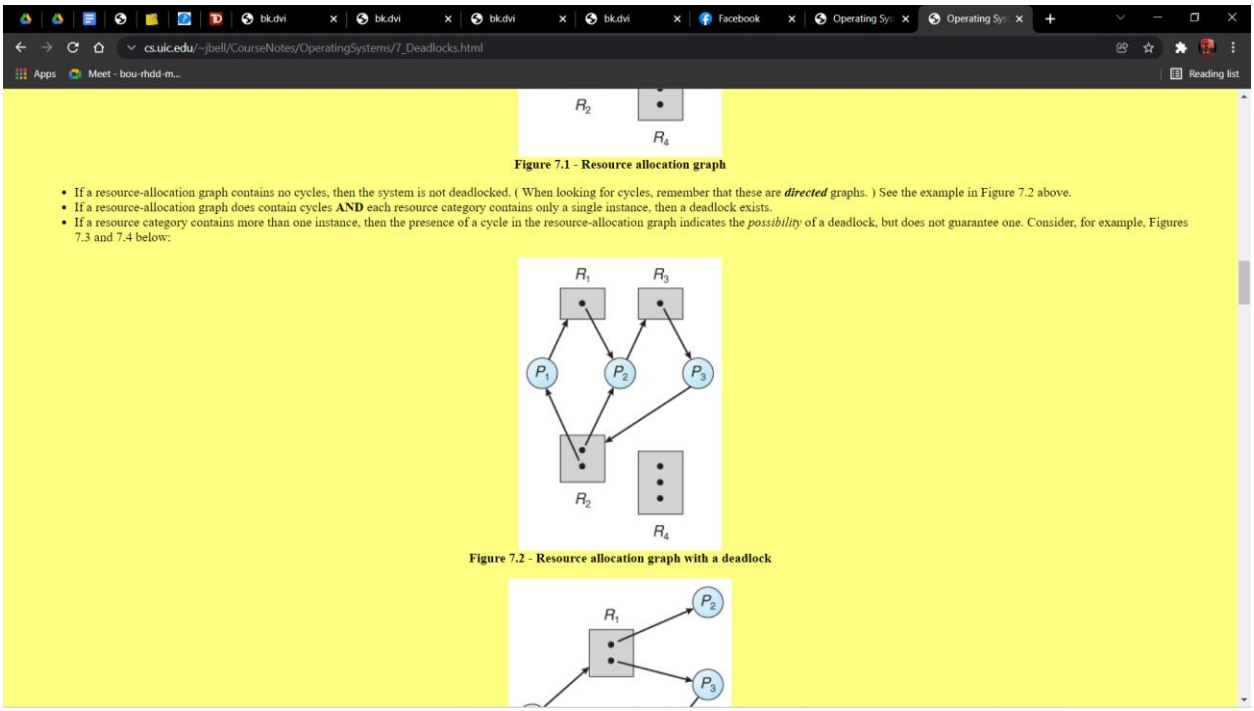
b. What is the average turnaround time for these processes with the SJF scheduling

algorithm?

7. Describe the general strategies for dealing with deadlocks. For single unit resources,

we can model resource allocation and requests as a directed graph connecting processes

and resources. Given such a graph, what is involved in deadlock detection?



8. What must the banker's algorithm know a priori in order to prevent deadlock?

Answer the following questions using the banker's algorithm:

a. What is the content of the matrix Need?

b. Is the system in a safe state?

c. If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately?