## Q1

Critical Section is the part of a program, which tries to access shared resources. That resource may be any resource in a computer like a memory location, Data structure, CPU or any IO device. It cannot be executed by more than one process at the same time; operating system faces the difficulties in allowing and disallowing the processes from entering the critical section.

The critical section problem is used to design a set of protocols, which can ensure that the Race condition among the processes will never arise. In order to synchronize the cooperative processes, our main task is to solve the critical section problem. We need to provide a solution in such a way that the following conditions can be satisfied:

* Mutual Exclusion – It means that if one process is executing inside critical section then the other process must not enter in the critical section.
* Progress – It means that if one process doesn't need to execute into critical section then it should not stop other processes to get into the critical section.

## Q2

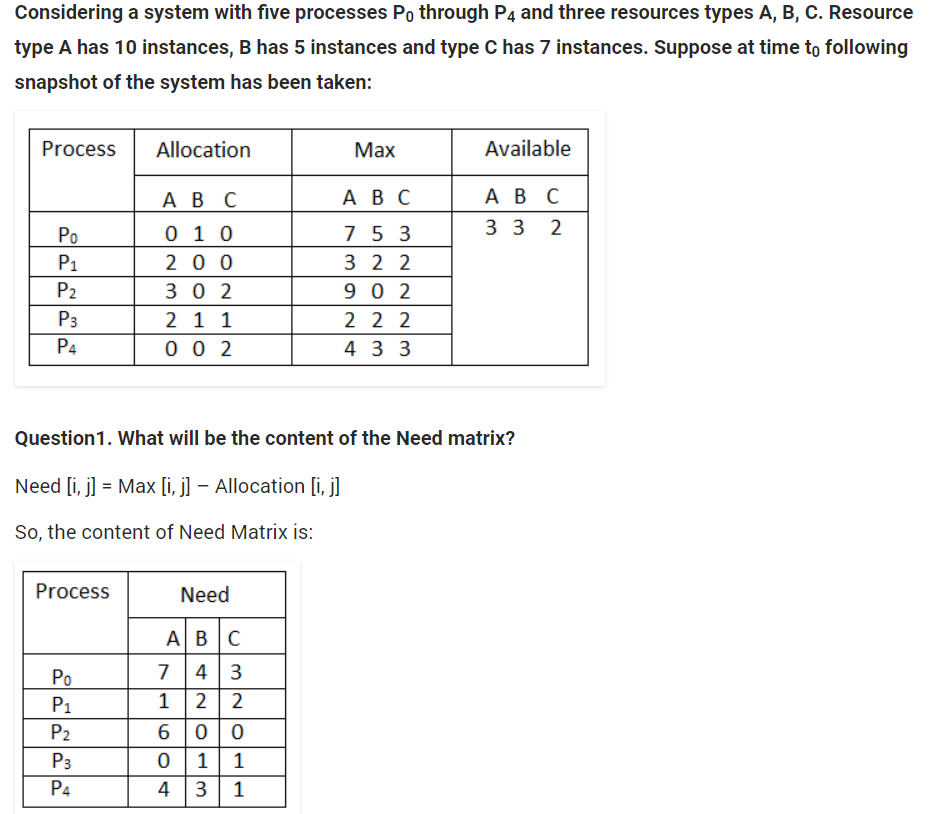
A Deadlock is a situation where each of the computer process waits for a resource which is being assigned to some another process. In this situation, none of the processes is executed since the resource it needs, is held by some other process which is also waiting for some other resource to be released. Deadlocks can be avoided by avoiding at least one of the four conditions, because all these four conditions are required to occur concurrently to cause a deadlock.

* Mutual Exclusion – Resources shared such as read-only files do not lead to deadlocks but resources, such as printers and tape drives, requires exclusive access by a single process.
* Hold and Wait – Here, processes must be prevented from holding one or more resources while simultaneously waiting for another resource.
* No Pre-emption – Pre-emption of process resource allocations can avoid the condition of deadlocks, wherever possible.
* Circular Wait – Circular wait can be avoided if we number all resources, and require that processes request resources only in strictly increasing (or decreasing) order.

## Q3

The banker’s algorithm is a resource allocation and deadlock avoidance algorithm that tests for safety by simulating the allocation for predetermined maximum possible amounts of all resources, then makes an “s-state” check to test for possible activities, before deciding whether allocation should be allowed to continue. The system is said to be in a safe state (no deadlock condition) if there exists a sequence of other valid system states that leads to successful completion of all processes.

* Processes request only 1 resource at a time.
* Request is granted only it results in a safe state.
* If request results in an unsafe state, the request is denied and the process continues to hold resources it has until its request can be met.
* All requests will be granted in a finite amount of time.
* Algorithm can be extended for multiple resource types.



## Q4

|  |  |  |
| --- | --- | --- |
| **BASIS FOR COMPARISON** | **PROCESS** | **THREAD** |
| Basic | Program in execution. | Lightweight process or part of it. |
| Memory sharing | Completely isolated and doesn’t share memory. | Shares memory with each other. |
| Resource consumption | More | Less |
| Efficiency | Less efficient as compared to process in context of communication. | Enhances efficiency in context of communication. |
| Time required for creation | More | Less |
| Context switching time | Takes more time. | Consumes less time. |
| Uncertain termination | Results in loss of process. | A thread can be reclaimed. |
| Time required for termination | More | Less |