Documentation

Solution pseudocode

```
Semaphore Customers = 0;
Semaphore Barber = 0;
Mutex Seats = 1;
int FreeSeats = N;

Barber {
      while(true) {
            /* waits for a customer (sleeps). */
            down(Customers);

            /* mutex to protect the number of available seats.*/
            down(Seats);

            /* a chair gets free.*/
            FreeSeats++;

            /* bring customer for haircut.*/
            up(Barber);

            /* release the mutex on the chair.*/
            up(Seats);
            /* barber is cutting hair.*/
      }
}

Customer {
      while(true) {
            /* protects seats so only 1 customer tries to sit
               in a chair if that's the case.*/
            down(Seats); //This line should not be here.
```

```
            if(FreeSeats > 0) {

                  /* sitting down.*/
                  FreeSeats--;

                  /* notify the barber. */
                  up(Customers);

                  /* release the lock */
                  up(Seats);

                  /* wait in the waiting room if barber is busy. */
                  down(Barber);
                  // customer is having hair cut
            } else {
                  /* release the lock */
                  up(Seats);
                  // customer leaves
            }
      }
}
```

Examples of Deadlock

Perhaps the most easily recognizable form of deadlock is traffic gridlock. Multiple lines of cars are all vying for space on the road and the chance to get through an intersection, but it has become so backed up there this no free space for potentially blocks around. This causes entire intersections or even multiple intersections to go into a complete standstill. Traffic can only flow in a single direction, meaning that there is nowhere for traffic to go once traffic has stopped. However, if the car at the very end of each line of traffic decide to back up, this frees up room for other cars to do the same and therefore the gridlock is solved. Another real-world example of deadlock is the use of a single track by multiple trains. Say multiple tracks converge onto one; there is a train on each individual track, leading to the one track. All trains are stopped, waiting for another to go, though none of them move. This is an example of deadlock because the resource, the train track, is held in a state of limbo as each train waits for another to move along so that they can continue.

How to solve deadlock

Methods of handling deadlocks: **There are three approaches to deal with deadlocks.**

1. **Deadlock prevention: The strategy of deadlock prevention is to design the system in such a way that the possibility of deadlock is excluded. Indirect method prevent the occurrence of one of three necessary condition of deadlock i.e., mutual exclusion, no pre-emption and hold and wait. Direct method prevent the occurrence of circular wait.** Prevention techniques – Mutual exclusion – **is supported by the OS.** Hold and Wait – **condition can be prevented by requiring that a process requests all its required resources at one time and blocking the process until all of its requests can be granted at a same time simultaneously. But this prevention does not yield good result because:**

- long waiting time required
- in efficient use of allocated resource
- A process may not know all the required resources in advance

No pre-emption – techniques for 'no pre-emption are'

- If a process that is holding some resource, requests another resource that can not be immediately allocated to it, the all resource currently being held are released and if necessary, request them again together with the additional resource.
- If a process requests a resource that is currently held by another process, the OS may pre-empt the second process and require it to release its resources. This works only if both the processes do not have same priority.

Circular wait One way to ensure that this condition never hold is to impose a total ordering of all resource types and to require that each process requests resource in an increasing order of enumeration, i.e., if a process has been allocated resources of type R, then it may subsequently request only those resources of types following R in ordering.

2. **Deadlock Avoidance: This approach allows the three necessary conditions of deadlock but makes judicious choices to assure that deadlock point is never reached. It allows more concurrency than avoidance detection A decision is made dynamically whether the current resource allocation request will, if granted, potentially lead to deadlock. It requires the knowledge of future process requests. Two techniques to avoid deadlock :**

- 1_Process initialtion denial
-2_Resource allocation denial

Advantages of deadlock avoidance techniques :

- Not necessary to pre-empt and rollback processes
- Less restrictive than deadlock prevention

Disadvantages:

- Future resource requirements must be known in advance
- Processes can be blocked for long periods
- Exists fixed number of resources for allocation

3. **Deadlock Detection:** Deadlock detection is used by employing an algorithm that tracks the circular waiting and killing one or more processes so that deadlock is removed. The system state is examined periodically to determine if a set of processes is deadlocked. A deadlock is resolved by aborting and restarting a process, relinquishing all the resources that the process held.

- This technique does not limit resources access or restrict process action.
- Requested resources are granted to processes whenever possible.
- It never delays the process initiation and facilitates online handling.
- The disadvantage is the inherent pre-emption losses.

Examples and solutions on starvation:

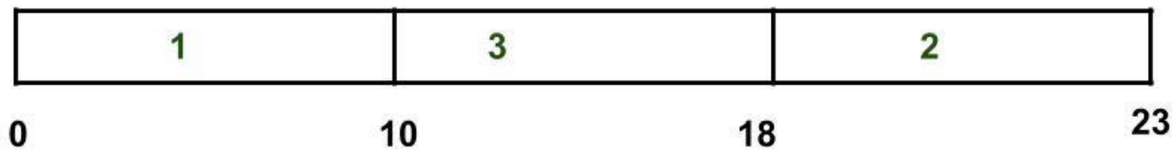
Example 1:

As we see in the below example process having higher priority than other processes getting CPU earlier. We can think of a scenario in which only one process is having very low-priority (for example 127) and we are giving other process with high-priority, this can lead indefinitely waiting for the process for CPU which is having low-priority, this leads to Starvation Further we have also discuss about the solution of starvation.

Solution to Starvation:

Aging is a technique of gradually increasing the priority of processes that wait in the system for a long time. For example, if priority range from 127(low) to 0(high), we could increase the priority of a waiting process by 1 Every 15 minutes. Eventually, even a process with an initial priority of 127 would take no more than 32 hours for the priority 127 process to age to a priority-0 process

Process	Burst time	Priority
1	10	2
2	5	0
3	8	1



Example 2:

In the below example, the process P2 is having the highest priority and the process P1 is having the lowest priority. In general, we have a number of processes that are in the ready state for its execution. So, as time passes, if only that processes are coming in the CPU that are having a higher priority than the process P1, then the process P1 will keep on waiting for its turn for CPU allocation and it will never get CPU because all the other processes are having higher priority than P1. This is called Starvation.

Process	Arrival Time	Burst Time	Priority
P1	0 ms	4 ms	100 ms
P2	0 ms	7 ms	1
P3	0 ms	10 ms	2
...	...	...	High Priority

Gantt Chart

Solutions of starvation:

Some solutions that can be implemented in a system to handle Starvation are as follows:

- An independent manager can be used for the allocation of resources. This resource manager distributes resources fairly and tries to avoid Starvation.
- Random selection of processes for resource allocation or processor allocation should be avoided as they encourage Starvation.
- The priority scheme of resource allocation should include concepts such as Aging, where the priority of a process is increased the longer it waits, which avoids Starvation.

1. Aging

It is a technique to avoid starvation in a scheduling system. It works by adding an aging factor to the priority of each request. The aging factor must increase the request priority as time passes and must ensure that a request will eventually be the highest priority request (after it has waited long enough)

OR

A condition which is used to reduce starvation of low priority tasks. It is a process which gradually increases the priority of task depending on waiting time. It ensure that jobs in the lower level queues will eventually complete their execution.

2.Priority Scheduling

Priority scheduling, it is the priority is not always set as the inverse of the CPU burst time, rather it can be internally or externally set, but yes the scheduling is done on the basis of priority of the process where the process which is most urgent is processed first, followed by the ones with lesser priority in order.

Processes with same priority are executed in FCFS manner.

The priority of process, when internally defined, can be decided based on memory requirements, time limits ,number of open files, ratio of I/O burst to CPU burst etc.

Whereas, external priorities are set based on criteria outside the operating system, like the importance of the process, funds paid for the computer resource use, market factor etc.

Types of Priority Scheduling Algorithm

Priority scheduling can be of two types:

1. Preemptive Priority Scheduling: If the new process arrived at the ready queue has a higher priority than the currently running process, the CPU is preempted, which means the processing of the current process is stopped and the incoming new process with higher priority gets the CPU for its execution.
2. Non-Preemptive Priority Scheduling: In case of non-preemptive priority scheduling algorithm if a new process arrives with a higher priority than the current running process, the incoming process is put at the head of the ready queue, which means after the execution of the current process it will be processed.

EXAMPLE OF REAL WORLD EXAMPLE ON STARVATION:

STARVATION

A starvation generally occurs as a result of a deadlock, live lock, or caused by a greedy process.

A process which requires to complete a task that is unable to gain regular access to the shared resources.

A real world example is the "**Airport with a single check-in counter**" problem.

Imagine an airport with a single check-in counter and two queues, one for business class and one for economy class. As you may know the business class has the priority over the economy class.

REAL WORLD EXAMPLE ON DEADLOCK:

1- Producer-consumer – Audio-Video player: network and display threads; shared buffer – Web servers: master thread and slave thread

2- Reader-writer – Banking system: read account balances versus update

3- Dining Philosophers – Cooperating processes that need to share limited resources

• Set of processes that need to lock multiple resources – Disk and tape (backup),

• Travel reservation: hotel, airline, car rental databases

Explanation for real world application and how did apply the problem

the problem of real application consists of students and teachers. The teachers remain in a state of sleep all the time , and apply this problem through when one of the students comes, the teacher with whom he entered wakes up, when the second student enters, the second teacher wakes up , when the third student enters, the third teacher wakes up ,and when the fourth student enters, the fourth teacher wakes up , the rest of the students are waiting . If it was a full queue, no students would enter again, and if it was an empty queue, the teachers would sleep and begin to admit the students .