CO21BTECH11002
Aayush Kumar
OS2 Theory Assignment 2

1)
This solution has a major drawback that it increases the time for running the processes significantly. ‘F’ becomes a bottleneck and it restricts the ability of processes to run in parallel even though it prevents deadlocks.

2)

a)
A deadlock cannot occur since preemption exists. Any process can request a resource anytime and if the resource is acquired by a blocked process then it is transferred to the new process and the new process does not need to wait for blocked processes to execute.

b)
Yes, indefinite blocking can occur. It is possible that a process may never get all the resources that it needs if the resources are continuously preempted by new processes.

3)
Let’s say that there exists a cycle between processes 1, 2, and 3, i.e., each process has acquired a resource and every process is waiting for other processes to release the resources. Now, since there are 4 resources, one resource will be free. Whichever process acquires this free resource first will execute and complete its job and release both the resources. Hence, some processes will always complete execution and break the cycle. Therefore the system is deadlock free.

4)

a)
A solution to this situation can be rejecting a request for a single chopstick if only one chopstick is present and no other philosopher has two chopsticks and the request is made by a philosopher that has zero chopsticks. This way if only one chopstick is present then it will be given to a philosopher that already has one chopstick and this philosopher will be able to finish eating. Hence, the system will not go into a deadlock.

b)
Difficulty in implementing this solution is that we need to store extra information about how many chopsticks are present and how many chopsticks each philosopher has. This requires us to use two extra mutexes for updating the two values.

5)

a)
Available: 2, 2, 2, 4

	Allocation				Max				Need			
	A	B	C	D	A	B	C	D	A	B	C	D
T0	3	1	4	1	6	4	7	3	3	3	3	2
T1	2	1	0	2	4	2	3	2	2	1	3	0
T2	2	4	1	3	2	5	3	3	0	1	2	0
T3	4	1	1	0	6	3	3	2	2	2	2	2
T4	2	2	2	1	5	6	7	5	3	4	5	4

- For T0: need>available
- For T1: need>available
- For T2: need<available:

T2 will finish execution.
After T2 finishes, available: 4,6,3,7

- For T0: need<available:
  T0 will finish execution.
  After T0 finishes, available: 7,7,7,8
- For T1: need<available:
  T1 will finish execution.
  After T1 finishes, available: 9,8,7,10
- For T3: need<available:
  T3 will finish execution.
  After T3 finishes, available: 13,9,8,10
- For T4: need<available:
  T4 will finish execution.

Hence, the order of execution is T2, T0, T1, T3, T4. Since the processes can complete execution, the system is in safe state.

b)
Since the need of T4 is more than what is available, we can not grant this request immediately, else the system will go into an unsafe state.

c)
If we grant this request, the available matrix becomes 2,1,1,4 and the need for T2 becomes 0,0,1,0. T2 can run first and all the processes would finish execution in order T2, T0, T1, T3, T4. The system is in safe state. Hence, this request can be granted.

d)
If we grant this request, the available matrix becomes 0,0,1,2 and the need for T3 becomes 0,0,1,0. T3 can run first and all the processes would finish execution in order T3, T1, T0, T2, T4. The system is in safe state. Hence, this request can be granted.