THEORY ASSIGNMENT 3

ABHAY SHANKAR K: CS21BTECH11001

(1) Yes, there is. Firstly, containment results in very high contention for the outer lock (in this case, F), resulting in reduced throughput and decreased efficiency.

Furthermore, if the contained locks are independent of each other, then any thread waiting on A (say) will have already acquired F, thus preventing any thread that wishes to acquire E (say) from gaining access until F has been released, i.e. A has been acquired.

This may lead to potential starvation, as follows.

```
Thread 1:
    wait(F)
    wait(A)
    signal(F)

Thread 2:
    wait(F)
    wait(E)
    signal(F)

Thread 3:
    wait(F)
    wait(A)
    wait(E)
    signal(F)

    Do Work

    signal(E)

    Lot of work.

    signal(A)
```

If thread 3 goes first and then thread 1, then thread 2 will be denied CPU time even though the resource it is waiting for is available. Thread 1 also waits for a resouce, but that resource is actually unavailable.

(2) (a) No, deadlock cannot occur. The given policy preempts resources.

(b) Yes, indefinite blocking can occur, as preemption without ageing leads to starvation. Any process can get preempted (its resources taken away) an arbitrary number of times.

(3) Assume a deadlock occured.

Then, some subset of the three processes but have 'hold and wait'. However, each process requires at most two resources, so each deadlocked process can only be holding one instance of the resource.

Thre are four instances of the resource and only three processes, so at leaast one resource must be available - this will be acquired by one of the processes, thus breaking the deadlock.

This contradicts out assumption. Therefore, there can be no deadlocks.

(4) Rule: If each philosopher except the one requesting the chopstick has exactly one chopstick, deny the request.

Drawback:

Checking the number of chopsticks each philosopher has asynchronously is error-prone:

Suppose P1 and P2 have no chopsticks, and all others have one. P1 makes a request.

We check P2, but the process gets swapped out, and P2 makes a request. P2's request is granted (process remains in CPU), and the process checking for P1 resumes.

It will find that not all philosophers except P1 have only one chopstick and grant the request, even though that is the case.

Thus, they are now deadlocked.

Using locks, we can synchronously check the number of chopsticks each philosopher has. However, this would be forbiddingly slow.

(5) (a)

$$Need = \begin{pmatrix} 3 & 3 & 3 & 2 \\ 2 & 1 & 3 & 0 \\ 0 & 1 & 2 & 0 \\ 2 & 2 & 2 & 2 \\ 3 & 4 & 5 & 4 \end{pmatrix}$$

$$Alloc = \begin{pmatrix} 3 & 1 & 4 & 1 \\ 2 & 1 & 0 & 2 \\ 2 & 4 & 1 & 3 \\ 4 & 1 & 1 & 0 \\ 2 & 2 & 2 & 1 \end{pmatrix}$$

$$Available = \begin{pmatrix} 2 & 2 & 2 & 4 \\ 6 & 3 & 3 & 4 \\ 9 & 4 & 7 & 5 \\ 11 & 6 & 9 & 6 \\ 13 & 7 & 9 & 8 \\ 15 & 11 & 10 & 11 \end{pmatrix}$$

where each row represents # available resources after each request is satisfied.

Order: T3 T0 T4 T1 T2

(b)

$$Alloc = \begin{pmatrix} 2 & 2 & 2 & 1 \end{pmatrix}$$
$$Req = \begin{pmatrix} 2 & 2 & 2 & 4 \end{pmatrix}$$
$$Available = \begin{pmatrix} 2 & 2 & 2 & 4 \end{pmatrix}$$

Therefore,

$$Available - Req = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$$

, which is clearly an unsafe state. The request cannot be granted immediately.

(c)

$$Alloc = \begin{pmatrix} 2 & 4 & 1 & 3 \end{pmatrix}$$
$$Req = \begin{pmatrix} 0 & 1 & 1 & 0 \end{pmatrix}$$
$$Available = \begin{pmatrix} 2 & 2 & 2 & 4 \end{pmatrix}$$

Therefore,

$$Available - Req = \begin{pmatrix} 2 & 1 & 1 & 4 \end{pmatrix}$$

The need matrix is

$$\begin{pmatrix} 3 & 3 & 3 & 2 \\ 2 & 1 & 3 & 0 \\ 0 & 0 & 1 & 0 \\ 2 & 2 & 2 & 2 \\ 3 & 4 & 5 & 4 \end{pmatrix}$$

This state is safe, as the following order of grants is valid: T2 T3 T0 T1 T4

The available resources are such:

$$Available = \begin{pmatrix} 2 & 2 & 2 & 4 \\ 4 & 6 & 3 & 7 \\ 8 & 7 & 4 & 7 \\ 11 & 8 & 8 & 8 \\ 13 & 9 & 8 & 10 \\ 15 & 11 & 10 & 11 \end{pmatrix}$$

The request can be granted immediately.

(d)

$$Alloc = \begin{pmatrix} 4 & 1 & 1 & 0 \end{pmatrix}$$
$$Req = \begin{pmatrix} 2 & 2 & 1 & 2 \end{pmatrix}$$
$$Available = \begin{pmatrix} 2 & 2 & 2 & 4 \end{pmatrix}$$

Therefore,

$$Available - Req = \begin{pmatrix} 0 & 0 & 1 & 2 \end{pmatrix}$$

The need matrix is

$$\begin{pmatrix} 3 & 3 & 3 & 2 \\ 2 & 1 & 3 & 0 \\ 0 & 1 & 2 & 0 \\ 0 & 0 & 1 & 0 \\ 3 & 4 & 5 & 4 \end{pmatrix}$$

This state is safe, and the following order of grants is valid: T3 T0 T4 T1 T2 The available matrix is the same as in part a.

The request can be granted immediately.