

Question 1)

Livelock occurs when two or more processes continually repeat the same interaction in response to changes in the other processes without doing any useful work. These processes are not in the waiting state, and they are running concurrently.

Deadlock a state in which each member of a group of actions, is waiting for some other member to release a lock.

A livelock the other hand is almost similar to a deadlock, except that the states of the processes involved in a livelock constantly keep on changing with regard to one another, none progressing. Thus Livelock is a special case of resource starvation.

Starvation is a problem which is closely related to both, Livelock and Deadlock. In a dynamic system, requests for resources keep on happening. Thereby, some policy is needed to make a decision about who gets the resource when. This process, being reasonable, may lead to a some processes never getting serviced even though they are not deadlocked.

Starvation happens when “greedy” threads make shared resources unavailable for long periods.

Starvation itself can occur for one process without another process being cyclically blocked; in this case no livelock or deadlock exists, just a single unfortunate process that gets no resources allocated by the scheduler.

Question 2)

True.

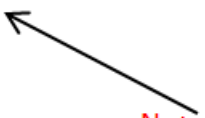
As we know we are using acquire and release with a mutex lock . Acquiring mutex, means no other threads are going to be able to access mutex as long as it is not released. So only after releasing mutex it can continue.

Question 3)

```
P(Semaphore s){  
    while(S == 0); /* wait until s=0 */  
    s=s-1;  
}
```

```
V(Semaphore s){  
    s=s+1;  
}
```

Note that there is
Semicolon after while.
The code gets stuck
Here while s is 0.



according to above definition of P and V signals we have:

- a) Yes it waits till empty becomes 0.
- b) Yes it waits till full is not empty(with the help of P signal) and then
- c) No because there is no lock declared to serve this purpose
- d) Yes as we have two different semaphores for each of operations

Question 4)

a) Unfortunately as there is no mutex, lock, barrier or semaphore , we may encounter a situation in which a thread over write on another thread and the latent thread will have the final effect.

b)

```
void insertList(...) {  
    lock.aquire();  
    int listNum = key;  
    listInsert(...)  
    lock.release();  
}
```

```
}
```

```
void insertHash(...){  
    lock.aquire();  
    nn → next = head;  
    head = nn;  
    lock.release();  
}
```

Question 5)

a) CBBA

YES

P3 → print(C);

c = 2-1

B = 1+1

P2 → print(B);

B = 2 -1 -1

A = 0 + 1

P1 → print(A);

b) CCBC

NO

P3 → print(C);

c = 2-1

B = 1+1

P3 → print(C);

c = 1-1

B = 2+1

P2 → print(B);

B = 3-1

It's not possible to print C as the c semaphore is 0 which makes it wait for ever.

c) BCBA

NO

P2 → print(B);

B = 1-1

A = 0+1

P3 → print(C);

C = 2-1

B = 0+1 // now the second wait is going to go out of the while wait

/* continue p2 */ P2 → print(B);

P1 → print(A);

A = 1-1

C = 1+1

d) BCAC

NO

P2 → print(B);

B = 1-1

A = 0+1

P3 → print(C);

C = 2-1

B = 0+1 // now the second wait is going to go out of the while wait

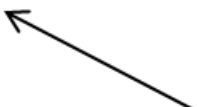
SO it's not what we want

Question 6)

```
P(Semaphore s){  
    while(s == 0); /* wait until s=0 */  
    s=s-1;  
}
```

```
V(Semaphore s){  
    s=s+1;  
}
```

Note that there is
Semicolon after while.
The code gets stuck
Here while s is 0.



- a) No
- b) Yes
- c) Yes
- d) No
- e) No
- f) Yes