Operating System - Mid-term II

Question1 [Pts:20 (10+10)]

a. Explain precisely how the situation of a complete traffic jam in a four-way crossing/chowke technically merits being a deadlock. Explain this by showing how four conditions of the deadlock are held in this situation.
Answer:

1. Mutual exclusion condition applies, since only one vehicle can be on a section of the street at a time.
2. Hold-and-wait condition applies, since each vehicle is occupying a section of the street, and waiting to move on to the next section of the street.
3. No-preemptive condition applies, since a section of the street that is a section of the street that is occupied by a vehicle cannot be taken away from it.
4. Circular wait condition applies, since each vehicle is waiting on the next vehicle to move. That is, each vehicle in the traffic is waiting for a section of street held by the next vehicle in the traffic.

b. Explain the difference between binary semaphore and mutex.

The mutex is similar to the principles of the binary semaphore with one significant difference: the principle of ownership. Ownership is the simple concept that when a task locks (acquires) a mutex only it can unlock (release) it. If a task tries to unlock a mutex it hasn’t locked (thus doesn’t own) then an error condition is encountered and, most importantly, the mutex is not unlocked. If the mutual exclusion object doesn't have ownership then, irrelevant of what it is called, it is not a mutex.

Question 2[Pts:15]

Give a solution for the problem of metro bus service in Lahore. There is a one-way narrow bridge on ravi where metro buses coming from opposite directions cannot cross. This means that if a bus enters from south i.e. Lahore (let’s call it southbus) then till the time it crosses the bridge and the bridge is clear, no bus from north i.e. Shahadara (let’s call it northbus) may enter it, and vice versa. Also, another southbus must be allowed to enter if another southbus is already using the bridge and vice versa. The traffic controller has contacted you for your repute in OS concepts to give an amicable solution. Using your synchronization knowledge, you are required to implement a starvation free solution. Use the following template skeleton to complete give the

```
Southbus(){

//approaching bridge entrance

//your synchronization logic goes here

    Pass the bridge code.

// exiting the bridge

// and your synchronization logic goes here

}
```

```
Northbus(){

//approaching bridge entrance

//your synchronization logic goes here

    Pass the bridge code.

// exiting the bridge

// and your synchronization logic goes here

}
```

Question 3[Pts: 15]

You are required to implement an algorithm of Coke machine which is a shared buffer

Two types of users

- Producer: Restocks the coke machine
- Consumer: Removes coke from the machine

Requirements

- Only a single person can access the machine at any time.
- If the machine is out of coke, wait until coke is restocked.
- If machine is full, wait for consumers to drink coke prior to restocking.
- If the machine is not full, producer can produce as many cokes as he likes (NOT more than machine's capacity)
- If the machine is not empty, consumer can consume as many cokes as they likes (until the machine is empty)

Solution:

```
shared binary semaphore mutex = 1;
shared counting semaphore empty = MAX;
shared counting semaphore full = 0;
shared anytype buffer[MAX];
shared int in, out, count;
```

PRODUCER :

```
anytype item;

repeat {

        /* produce something */
        item = produce();

        /* wait for an empty space */
        wait(empty);

        /* store the item */
        wait(mutex);
        buffer[in] = item;
        in = in + 1 mod MAX;
        count = count + 1;
        signal(mutex);

        /* report the new full slot */
        signal(full);

} until done;
```

CONSUMER:

```
anytype item;

repeat {

        /* wait for a stored item */
        wait(full);

        /* remove the item */
        wait(mutex);
        item = buffer[out];
        out = out + 1 mod MAX;
        count = count - 1;
        signal(mutex);

        /* report the new empty slot */
        signal(empty);

        /* consume it */
        consume(item);

} until done;
```