Operating System 2: Exam 2

24th February 2022

Tanmay Garg CS20BTECH11063

Q1.

Sol.

```
Oxygen()
{
While(true)
{
If num_h == 2
{
Hydrogen1.signal()
Hydrogen2.signal()
}
Else
{
Oxygen1.wait()
}
}
}
Hydrogen()
{
While(true)
{
If(num_h == 1 && num_o == 1)
{
Hydrogen2.signal()
Oxygen1.signal()
}
```

```
    Else
    {
    Hydrogen1.wait()
    }
    }
Monitor Create_Water{
Condition hydrogen, oxygen
Int num_h, num_o
Init()
{
Num_h = num_o = 0
}
}
```

Q2.

Sol.

```
int lock = 1;

acquire(){
        while(!LoadLinked(&lock))
        {
                //Wait till exit the loop is possible
        }
        StoreConditional(&lock, 0);
}

release(){
        StoreConditional(&lock, 1);
}
```

Q3.

Sol.

Mutex = 1

wait(mutex);

... body of F ...

if (next count > 0) signal(next);

else signal(mutex);

The operation x.signal() can now be implemented as

x count++;

if (next count > 0) signal(next);

else signal(mutex);

wait(x sem);

x count--;

The operation x.continue() can be implemented as

```
 if (x count > 0)
 {
next count++;
 signal(x sem);
wait(next);
next count--;
 }
```

Q4.

Sol.

```
Void fetch_xor(atomic_int *x, int a)

{

Int temp

Do

{

Temp = *x

}while(temp != compare_and_swap(x, temp, temp.xor(a)))

}
```

The function will try to apply xor between x and a using compare and swap operation till it succeeds in doing so

Q5.

Sol.

- This is the signal and wait implementation of monitor using semaphores
- Our binary semaphore mutex is used for mutual execution algorithm
- Wait(mutex) is called before entering the monitor and must execute signal(mutex) after leaving the monitor
- The signaling process must wait until the resumed process either leaves or waits
- An additional binary semaphore next is hence used and another integer variable next_count is used to keep track of number of processes suspended on next
- If there are processes suspended on next then the signal() is executed to remove one of the processes from the waiting queue and let it be run by the system
- If the signaling process wanted to suspend itself for some duration then it use next semaphore for it

Q6.

Sol.

- In the dining philosopher problem, we have 5 philosophers and 5 chopsticks
- Each chopstick is between two philosophers
- At a given point in time they can either eat or think and each philosopher requires 2 chopsticks to eat
- The monitors help us schedule the processes as to when a philosopher can eat and when he can think
- It can be possible that one of philosophers can starve to death, it may so happen that while our philosopher is ready to eat but one of the chopsticks is available but other has been occupied by the adjacent philosopher
- As soon as this philosopher puts down his chopstick, the other might start eating so the other chopstick is occupied again
- This entire cycle can repeat over and over again alternatively, and the philosopher will starve to death

Q7.

Sol.

- We can assume the chairs placed outside to be similar to a server which can accept atmost N socket connections
- If the number of connections that the server currently has is equal to N then it wont accept any more connections
- The representative can be assumed to be a program that runs the process from the connection one at a time and then breaks the connection after the process is completed
- The semaphore can be initialized to the number of allowable connections which is N
- When a connection is accepted or when a student accepts a chair then acquire() can be called
- After this the representative can take interviews of the students or the server can now run the job provided by each process based on a first come first serve basis as there is no deadline for a process to be completed
- It is assumed that the interviews will take finite amount of time to be completed and it cannot take over interview process and never end it
- The student will be like a socket connector, it will keep on trying to connect to the server till it can acquire a connection
- When the interview is completed, the connection will then be released and then another student can connect to it
- The representative, who is the server that accepts those N connections and runs jobs on a first come first basis
- When the amount of connections is N, then all calls to acquire() will be blocked until a connection is completed and then breaks and then release() can be called
- Our current design satisfies all the conditions listed
- In the case when there are no connections or when there are no students then the server can run some analyze process to analyze the interview
- When a student arrives, this process can be put to sleep and will be reawaked when the all the connections are once again empty