CS3523 - Operating Systems 2 Quiz 2
Process Synchronization
Vanga Aravind Shounik-CS20BTECH11055

Question 1

Here, we need to know the number of hydrogens present in the system and number of oxygens present in the system. Let us name them numH, numO, now let us have the values of number of hydrogens already assigned to a water molecule and the number of oxygens already assigned to a water molecule as Hassign, Oassign, if number of H assigned or number of oxygens assigned to a water molecule is 0, then we try to create a water molecule.

```
Hydrogen(){
     wait(&lock);
     numH++;
     if(numH>=2 && numO>=1)
     {
          Hassign+=2;
          Oassign+=1;
     }
     signal(&lock);
}
Oxygen(){
     wait(&lock);
     numO++;
     if(numH>=2 && numO>=1)
     {
          Hassign+=2;
          Oassign+=1;
     }
     signal(&lock);
}
```

Question 2

Here, store conditional checks if the value of the ptr is changed somewhere else using the loadlinked() which gives the value of the attribute stored in the pointer. Now, we

have to always update the answer in every case, then we can do the stores conditional only once in the system at a time. Which means we have to use a lock.

Consider a binary semaphore named `lock` which tells us whether the function storeconditional is running or not.
So,

```
wait(lock);
StoreConditional(*ptr,value);
signal(lock);
```

This makes sure that the data is not misplaced while something is changing the data. And in every case, it returns 1 or true.

Question 5

In the signal and wait scheme, Since a signaling process must wait until the resumed process either leaves or waits, an additional binary semaphore, next, is introduced, initialized to 0. Here, the signaling processes may use next to suspend themselves. An integer value `next_count` is also introduced to take note of the number of processes suspended using next. So, if the process terminated itself using next, then it uses the next semaphore, so, we have to signal(next) in line 8 so that the next process starts executing.

Question 6

Here, the solution to Dining Philosopher problem without monitors is
If a philosopher is hungry, then he checks if the left spoon is not taken by anyone, and then he checks whether the right spoon is not taken by anyone and after he gets the 2 spoons, he starts eating. But in this case, the case where all the philosophers get hungry at the same time, then the first philosopher may get the left spoon but he may not get the right spoon because the right spoon is taken by the philosopher next to him and no philosopher gets the spoons to eat. So, it is a deadlock. This occurs because the philosopher takes one spoon after the other. To prevent this we use monitors and check whether both the spoons to the left and right to him are not being used by other philosophers and then he starts eating. In this case, a deadlock can't occur but a philosopher may starve. If the philosopher doesn't get the spoons, then he should wait for the spoons on both sides. Now consider a scenario where a philosopher didn't start eating because his left spoon is being used by the philosopher to the left. Now, before the left philosopher completes eating, if the right philosopher starts eating then even

after the left philosopher completes eating, the philosopher can't start eating because his right spoon is occupied. This may continue for a long time and lead to starvation.

Question 7

Here,the conditions are
Representative does other work if no students are present.
Student requests for an interview if the representative is doing other work.
Student waits in the chair if the representative is conducting an interview for others.
Student leaves if all the chairs are filled outside the room.

Let a semaphore nos(number of students) = 0, nor(number of remaining) = N, be the initial values of the semaphores respectively and a binary semaphore rep(representative)

```
Representative(){
     if(nos==0)
          //Free time
          signal(rep);
}
Student(){
     //decrements the nor value if a place is present
     wait(nor);
     //increments the number of students present
     wait(nos);
     //asks the rep for a seat when the rep is free
     wait(rep);
     /* The student waits in the queue and takes the interview
*/
     //decrements the number of students
     signal(nos);
     //increments the number of places present
     signal(nor);

}
```