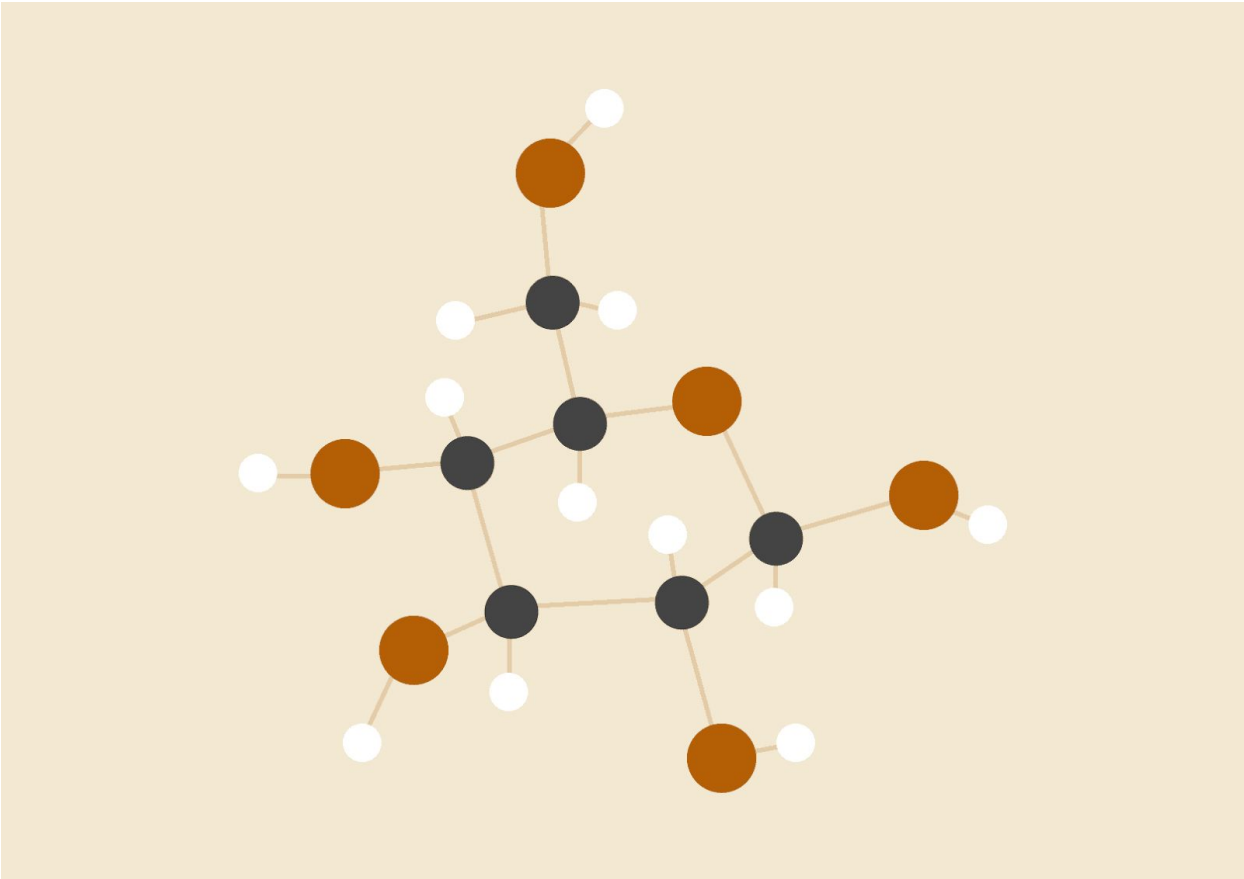
OPERATING SYSTEM 2

Theory assignment



Happy Kumar

03 - 03 - 2019

CS17BTECH11018

Dr. Sathya Peri

Ans 1 :

a) Producer can't produce less than 0, so minimum value of empty can be 0.

The minimum value of full can be 0, as it can't have consumed process in negative numbers.

b) Producer can't produce more than the buffer size, so maximum value of buffer will be n.

The maximum value of full can be n, as consumer can't consume more the the buffer size.

c) 0 <= full <= n(buffer size)

0 <= empty <= n(buffer size)

n - 1 <= full + empty <= n(buffer size)

Ans 2 :

The solution to this problem uses

```
semaphore rw mutex = 1;
semaphore mutex = 1;
int read count = 0;
```

The code for writer process is as:

```
while (true) {
        wait(waiting_queue);
        wait(rw mutex);
        signal(waiting_queue);
        . . .
        /* writing is performed */
```

```
        . . .
        signal(rw mutex);
}
```

The code for reader is shown below :

```
while (true) {
        wait(waiting_queue);
        wait(mutex);
        read count++;
        if (read count == 1)
                wait(rw mutex);
        signal(waiting_queue);
        signal(mutex);
        . . .
        /* reading is performed */
        . . .
        wait(mutex);
        read count--;
        if (read count == 0)
                signal(rw mutex);
        signal(mutex);
}
```

This solution can only satisfy the condition that "no thread shall be allowed to starve" if and only if semaphores preserve first-in first-out ordering when blocking and releasing threads. Otherwise, a blocked writer, for example, may remain blocked indefinitely with a cycle of other writers decrementing the semaphore before it can.

Ans 3 :

```
void lock spinlock(int *lock) {
{
        while (true) {
        if (*lock == 0) {
                /* lock appears to be available */
                if (!compare and swap(lock, 0, 1))
                        break;
        }
}
```

This function works correctly as it only lets CAS if lock is available otherwise keeps waiting before checking.

Ans 4 :

```
if (getValue(&sem) > 0)
        wait(&sem);
```

This doesn't work correctly as two threads can come and check for check for getValue(&sem) and pass and only one thread will be passed by invoking wait(&sem) and the other thread will be there blocked doing nothing, thereby blocked while waiting for semaphore and hence it doesn't work.