**Unit – III Process Synchronization**

### Classic Problems on Synchronization:

**Bounded Buffer Problem / Producer Consumer Problem:**

Bounded buffer problem or producer-consumer problem is a classical synchronization problem where we have a buffer with n cells or n slots and there are 2 process producers and consumers can produce and consume one article at a time.

**Mutex** - Semaphore for mutual exclusion to access buffer pool, initialized to 1.

**Empty** - Semaphore to count empty buffer N.

**Full** - Semaphore to count fill buffer 0.

|  |  |
| --- | --- |
| **Producer** | **Consumer** |
| Wait (empty)  Wait (mutex)  Critical section  Signal (mutex)  Signal (full) | Wait (empty)  Wait (mutex)  Critical section  Signal (mutex)  Signal (empty) |

**Readers / Writers Problem:**

There is a shared piece of text and 2 types of process in accessing this text reader and writer. There is no clash between reader and reader therefore when a reader is inside critical section then other readers may get an only entry but when a write is inside critical section then neither the reader nor the writer gets an entry. Hence in the solution, we have used 3 resources a semaphore mutex for synchronization between writer and reader-writer. While read count (RC) is a simple integer variable which is given security by reading semaphore which works for synchronization between reader- reader.

**Writer**

while(1)

{

wait(mutex)

write

signal(mutex)

}

**Reader**

while(1)

{

wait(Read)

Rc = Rc + 1;

if(Rc = = 1)

{

wait (mutex)

}

wait(Read)

Rc = Rc-1

if(Rc ==0)

{

signal(mutex)

}

signal(Read)

}

**Dining Philosopher Problem:**

In this problem, there is a circular table and number of philosopher a sitting on the table. There is a chop-stick placed between every philosopher. Philosopher prior only 2 processes either they think or eat (using 2 chop-stick).

Solution suffers from dead-lock and the following modification can be done:

1. Allow **n-1** philosopher to sit on the table.
2. Allow **n+1** chop-stick to be put on the table.
3. **n-1** philosopher picks the left chop-stick first(the right and then the last philosopher pick the right first and left or vice-versa.
4. Division can be done between add an even number philosopher.
5. Take one more semaphore and consider 2 wait operation as a critical section.

Semaphore chopstick[] = new Semaphore[5];

process philosopher\_i

{

while (true)

{

chopstick[i].acquire();

chopstick[(i+1) % 5].acquire();

chopstick[i].release();

chopstick[(i+1) % 5].release();

}

}