Page No.
Date

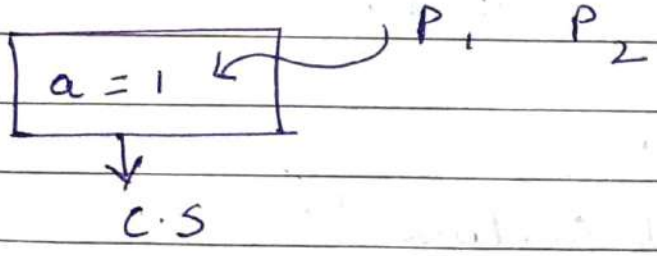
* Semaphores

Imp Semaphore is an integer variable that solves the critical section problem. After initialization, it can be accessed by two atomic operations

1) wait(s)
{
while(S<=0); // trap
S = S-1;
}

2) Signal(s)
{
S = S+1;
}

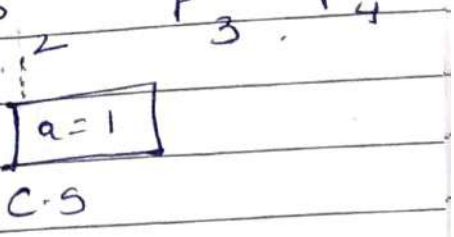
Solves critical Section problem.



P_1 is accessing critical section. Until P_1 is executing in critical section, P_2 cannot enter.

3 criterias to be satisfied.

1) Mutual Exclusion
2) Progress
3) Bounded wait.



int S; → Semaphore
S = 1;

Page No.
Date
S = 1

```
wait (S)
{
 while( S<=0); //trap
   S = S-1;
}
```

```
signal (S)
{
  S = S+1;
}
```

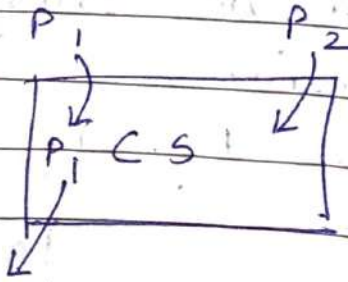
Solution of critical section problem using Semaphores

→ Let $P_1, P_2, P_3, P_4 \cdots P_n$ are the processes that wants to go to the critical section

P_1 P_2 ...

S = 1 0

```
do
{
 wait (S);
 // critical Section (P1)
 Signal (S);
 // remainder Section
}
 while (T)
```



Page No.
Date

Types of Semaphore

1) Binary Semaphore (1 & 0)
2) Counting Semaphore · ~~(-∞ to ∞)~~

Binary Semaphore - The value of a binary semaphore can range only between 0 & 1.
On some systems, binary semaphores are known as mutex locks, as they are locks that provide mutual exclusion

Counting Semaphore -
Its value can range over an unrestricted domain. It is used to control access to a resource that has multiple instances.

S = 2 ✗ 0 $P_1 \rightarrow R_1$, $P_2 \rightarrow R_2$ $P_3 \rightarrow$ wait

```
wait(S)
{
while (S<=0); //no-op
S = S-1;
}
```

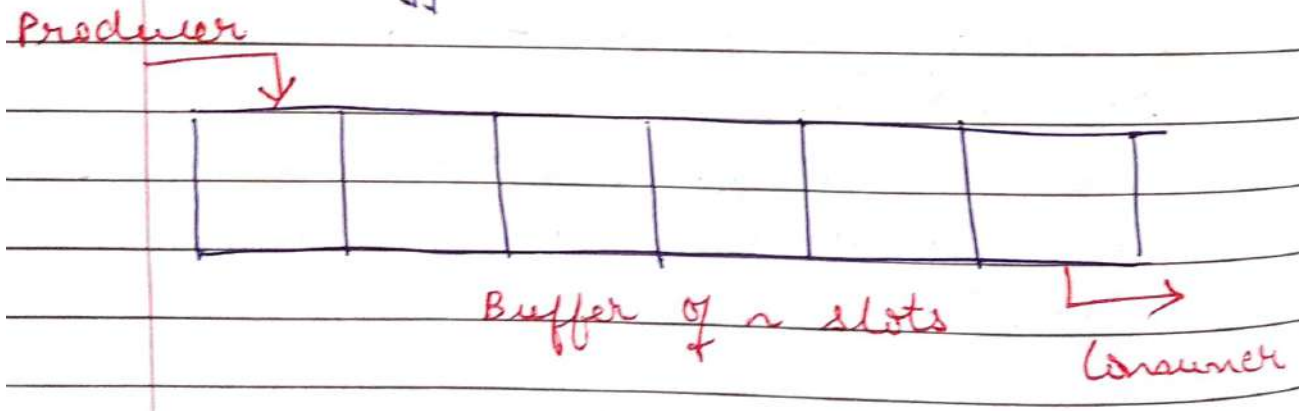
```
signal(S)
{
S = S+1;
}
```

Processes P_1, P_2, P_3
Resource (R) has 2 instances R_1, R_2
i.e. Resource can be used by 2 processes at the same time.
Set S = 2 (no. of resource instances)

Page No.
Date

* Producer Consumer Problem (Bounded Buffer Problem)

- The Producer Consumer problem is one of the classic problems of synchronization.
- There is a buffer of n slots & each slot is capable of storing one unit of data.
- There are two processes ~~run~~ running, namely, Producer & Consumer, which are operating on the buffer.



There is a buffer ~~of~~ of n slots. Producer will insert data into an empty slot in the buffer & consumer will consume / remove data from ~~buffer~~ a filled slot in the buffer.

Page No.
Date

Problems to take care of :-

1) The Producer must not insert data when the buffer is full.
2) The Consumer must not remove data when the buffer is empty.
3) The Producer & Consumer should not insert & remove data simultaneously.

Solution to the Producer Consumer problem using Semaphores:

We will make use of 3 Semaphores.

1) m(mutex), a binary semaphore which is used to acquire & release the lock.
2) empty, a counting semaphore whose initial value is the number of slots in the buffer, since, initially all slots are empty.
3) full, a counting semaphore whose initial value is 0.

P.T.O

Date

Producer

```
do {
    wait(empty); // wait until
                 empty > 0 & then
                 decrement 'empty'

    wait(mutex); // acquire lock
    /* add data to buffer */

    signal(mutex); // release lock
    signal(full); // increment
                  'full'
} while(TRUE)
```

Consumer

```
do{
    wait(full); // wait until full>0
                & then decrement
                'full'
    wait(mutex); // acquire lock
    /* remove data from buffer */

    signal(mutex); // release lock

    signal(empty); // increment 'empty'
} while(TRUE)
```