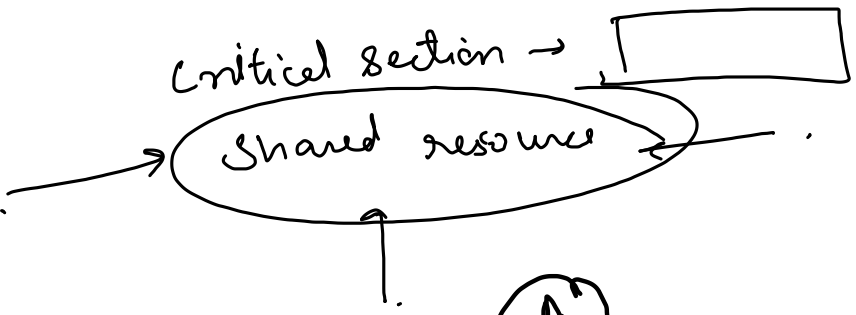
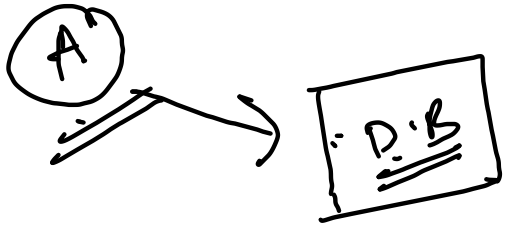
Synchronization Techniques

1) Lock variable
2) Semaphores

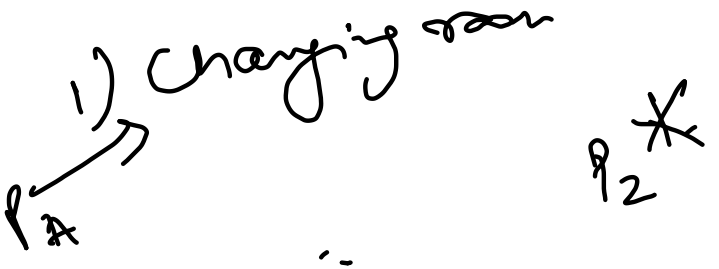
Race condition -



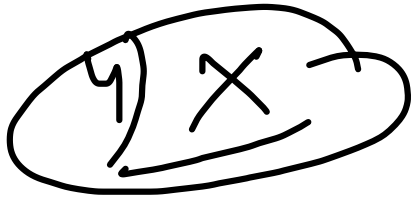


Synchronization

1) Mutual ex.
2) Progress



3) B.W → Starvation



Lock variable

Process Po	Process P1
Entry section: while (turn! =0)	Entry Section: while (turn! =1)
Critical Section	Critical Section
Exit: Turn=1	Exit: Turn=0

turn -

Flag ⇒ Turn = ~~1~~ 0 1.

P_0 | P_1

Entry
while (turn! = 0)
{ C.S. -
Exit ⇒ t = 0 }

while (turn! = 1) ;X
{ C.S. ..
Exit: t = 1 }

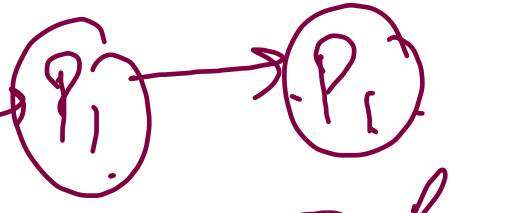
1 != 0

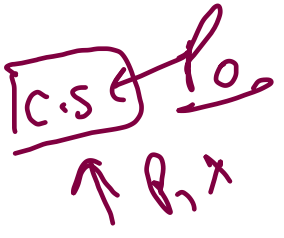
1) M.E. ✓

2) Progress X

$P_0 \rightarrow P_1 \rightarrow P_0 \rightarrow$

$P_1 \rightarrow P_0$





$P_1 \rightarrow P_2 -$

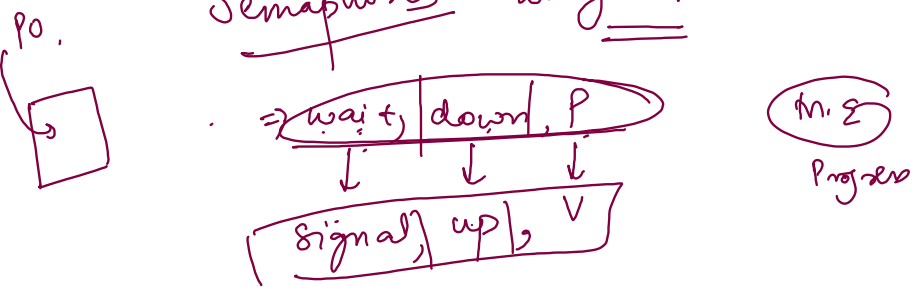
1) Turn variable →

M.E ✓ Progress ✗

2) Semaphores

$ →

Semaphores → integer variables



1. Binary - 1|0

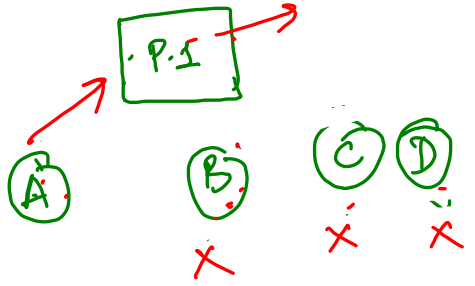
2) Counting :=> integer $(-\infty, \infty)$

$S = 4\ 3\ 2\ 1\ 0$

P1 P2 P3 P4

A B C D E

$S = 1 \cdot 0 \cdot 1$

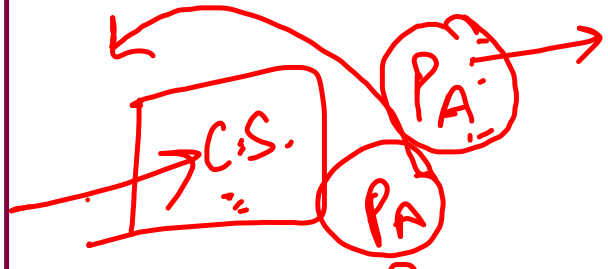


Binary semaphore

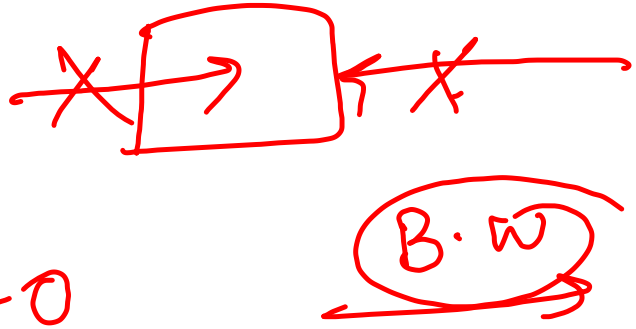
Process →

```
Wait(S)
{
    while(S<=0);
    S = S-1;
}
Critical Section
Signal(S)
{
    S = S+1;
}
```

S = 0 / 1

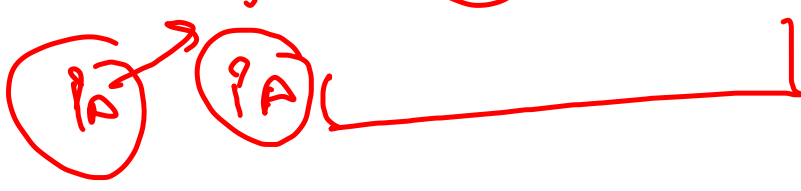


1) PB
2) PC
3) PD
PA



LIFO

FIFO



while (T)

{

}

→ while ( )

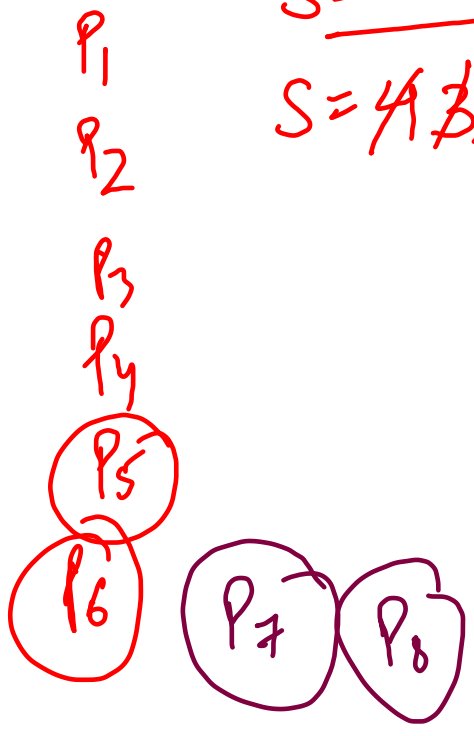
while(T);

while(F);

→

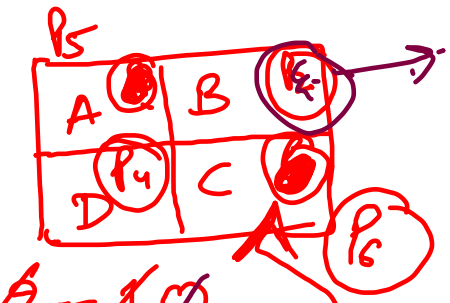
Counting Semaphore

```
wait(S)
{ S = S-1;
   if (S<0)
   { block;
   }
Critical
 section
Signal(S)
{ S = S+1;
   if (S<=0)
     wake up;
   }
```

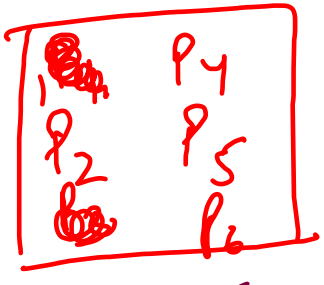


S = no. of resources

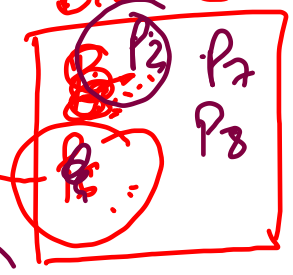
S = 4 3 2 1 0 -1 -2 -3 -4



C.S.



Block State



LIFO

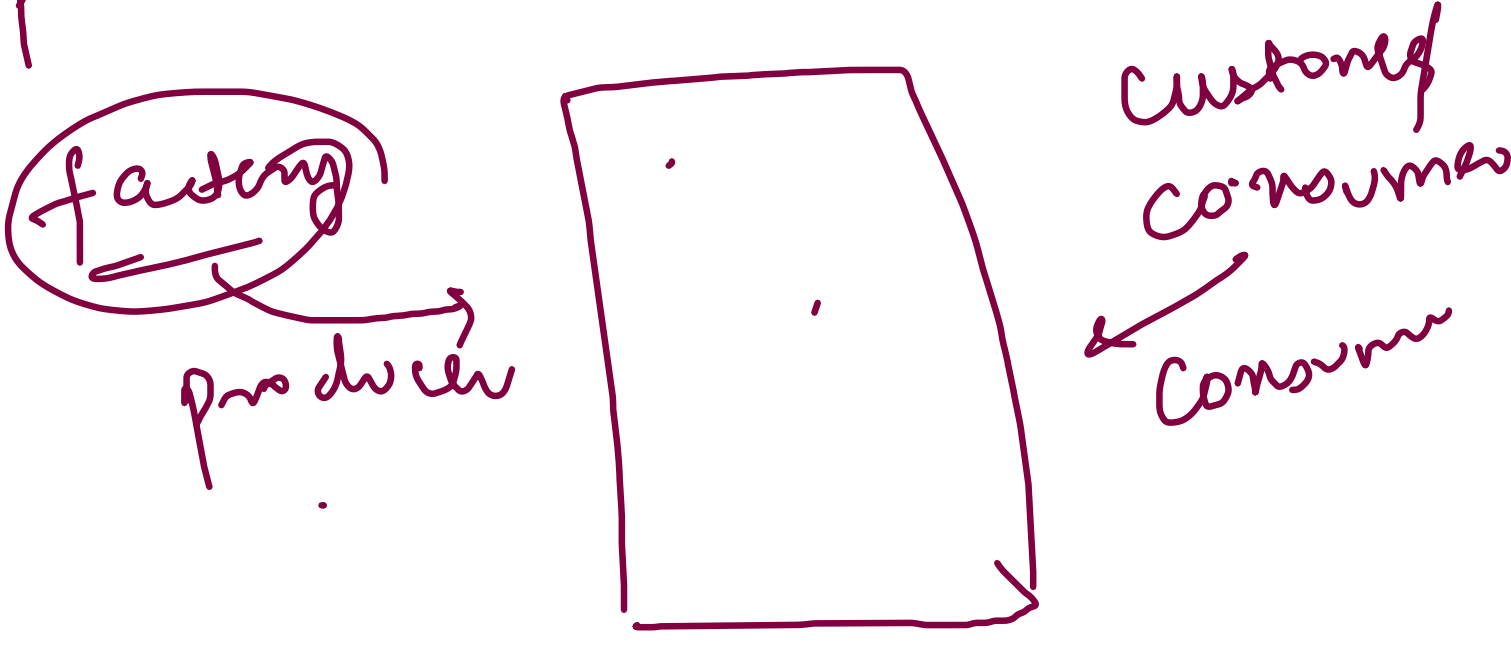
FIFO

Binary counting

1) Producer - consumer problem.

2) Dining philosphers

→ Bounded buffer problem.

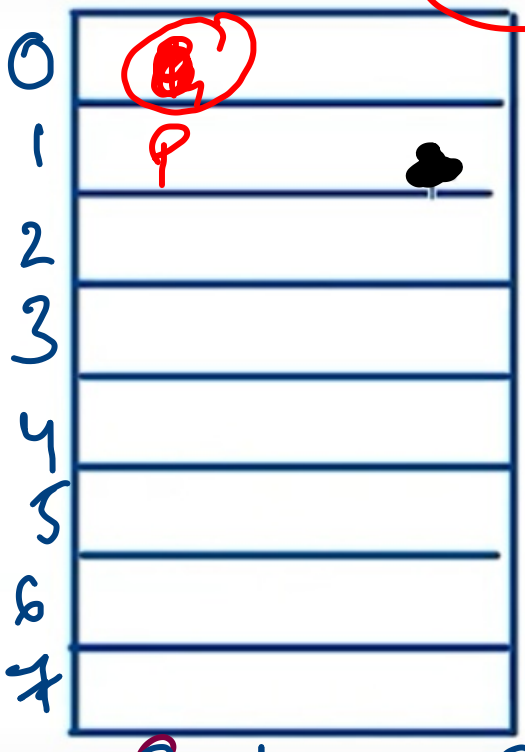


In = 0 1 2

Out = 0 1

P

```
Void producer()
{
  Int product_p;
  while(True)
  {
    product_p=produce_it
    ..
    while(count==N);
    Buffer [in] = product_p
    in= (in+1) mod N;

    count = count+1;
  }
}
```



C

```
int consumer()
{
  Int product_c;
  while(True)
  {
    while(count==0);
    product_c= Buffer [out];
    out= (out+1) mod N;

    count = count-1;
  }

  return(product_c)
}
```

Buffer = 8

count = 0 1 2 1

Kitne spaces occupied hain ←

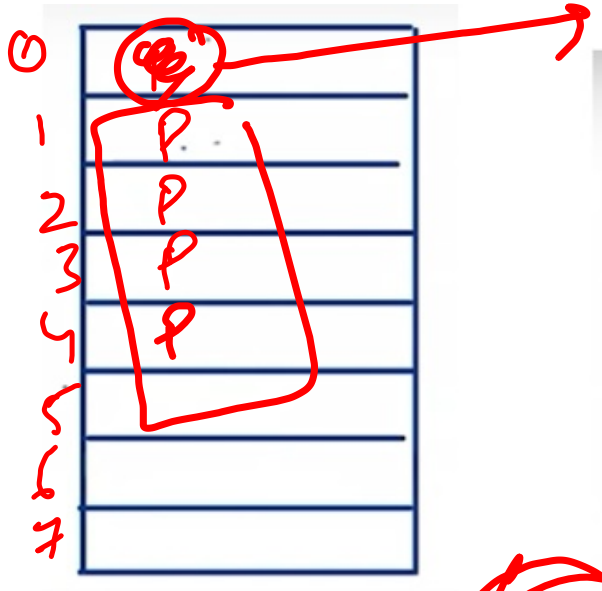
Best case → Produce & consume are not coming together

```
Void producer()
{
  Int product_p;
  while(True)
  {
    product_p=produce_it

    while(count==N) ;
    Buffer [in] = product_p
    in= (in+1) mod N;

    count = count+1;
  }
}
```

```
Load Rp, m[count];
Incr Rp;
Store m[count] , Rp;
```

Preempt

M.E.X



```
int consumer()
{
  Int product_c;
  while(True)
  {
    while(count==0) ;
    product_c= Buffer [out];
    out= (out+1) mod N;

    count = count-1;
  }

  return(product_c);
}
```

```
Load Rp, m[count];
Dec Rp;
Store m[count] , Rp;
```

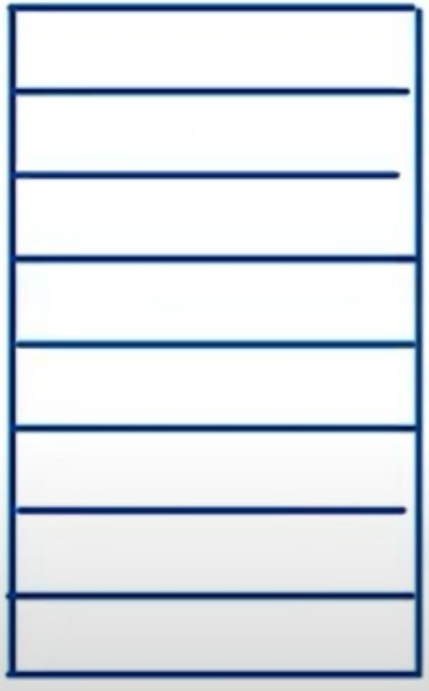
Preempt

Race Condition

P

```
Void producer()
{
 Int product_p;
 while(True)
 {
  product_p=produce_item();

  Down (Empty);
  Down(Mutex);


  Buffer [in] = product_p;
  in= (in+1) mod N;

  Up(Mutex);
  Up(full);
 }
```

```
Binary Semaphore Mutex =1;
       Semaphore full=0;
       Semaphore Empty = N;
```



C

```
int Consumer()
{
  Int product_c;
  while(True)
  {
    Down (Full);
    Down(Mutex);

   product_c= Buffer [out]
    out= (out+1) mod N;

    Up(Mutex);
    Up(Empty);
  }

  return(product_c);
}
```