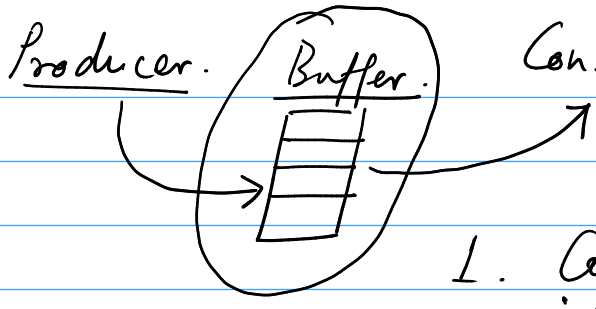


1. Consumer cannot consume if the buffer is empty
2. Producer cannot produce if the buffer is full.
3. Buffer should be accessed mutually exclusively.

Data Structures

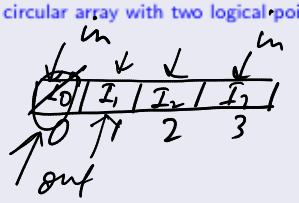
/*The shared buffer is implemented as a circular array with two logical pointer in and out*/

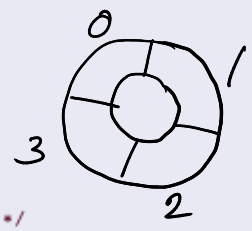
```
#define BUFFER_SIZE 10 4
typedef struct{
...
}item;

item buffer[BUFFER_SIZE];

int in=0;       /*in points to the next free position in the buffer*/
int out=0;      /*out points to the first full position in the buffer*/

int counter=0;  /*incremented every time an item is produced and decremented every time
                  an item is consumed*/
```





Producer

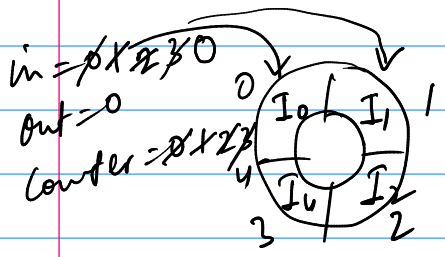
/*The producer produces a new item in the local variable nextProduced*/

```
while(true){
   while(counter == BUFFER_SIZE);
   buffer[in]=nextProduced;
   in=(in+1)%BUFFER_SIZE;
   counter++;
}
```

Consumer

/*The consumer has a local variable nextConsumed in which the item to be consumed is stored*/

```
while(true){
   while(counter == 0);
   nextConsumed=buffer[out];
   out=(out+1)%BUFFER_SIZE;
   counter--;
}
```

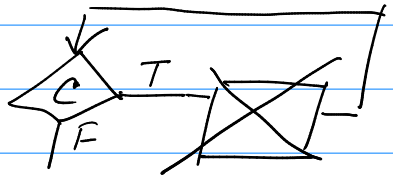


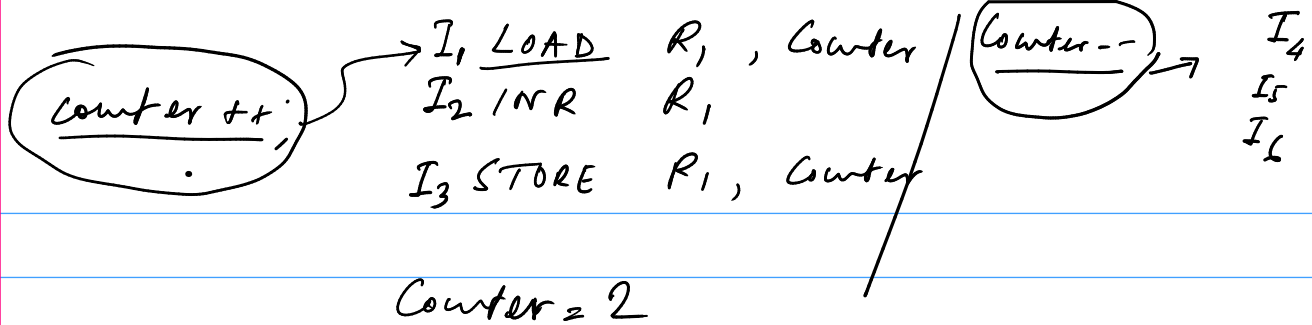
Pr.

Co:

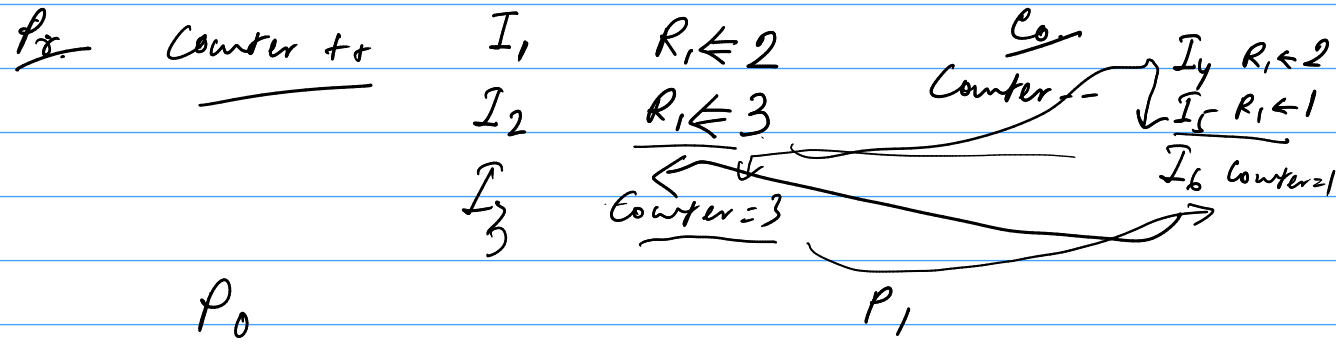
while(True){
while(counter == 0);

}





Counter = 2



P0

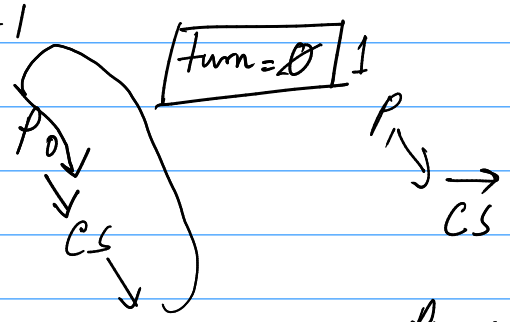
```
do{
    while(turn != 0);
    < critical section >
    turn = 1;
    < remainder section >
}while(true);
```

P1

```
do{
    while(turn != 1);
    < critical section >
    turn = 0;
    < remainder section >
}while(true);
```

i = 0 J = 1

i = 1 J = 0



turn = 0

P1

Progress not satisfied.

Strict Alternation

P_0

```
do{
    flag[0] = true;
    while (flag[1]);
    < critical section >
    flag[0] = false;
    < remainder section >
}while(true);
```

P_1

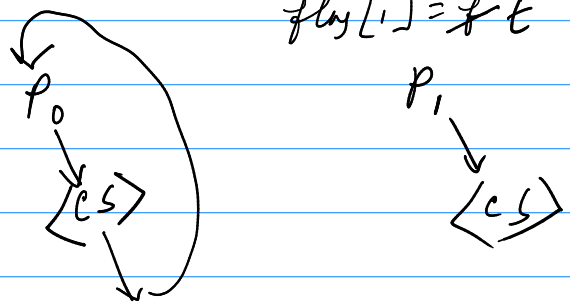
```
do{
    flag[1] = true;
    while (flag[0]);
    < critical section >
    flag[1] = false;
    < remainder section >
}while(true);
```

$i=0, j=1$

$i=1, j=0$

~~flag[0] = f t f t~~

~~flag[1] = f t~~



P_0

```
do{
    flag[0] = true;
    turn = 1;
    while (flag[1] && turn == 1);
    < critical section >
    flag[0] = false;
    < remainder section >
}while(true);
```

P_1

```
do{
    flag[1] = true;
    turn = 0;
    while (flag[0] && turn == 0);
    < critical section >
    flag[1] = false;
    < remainder section >
}while(true);
```

$i=0$
$J=1$

$flag[0] = \neq t$
$flag[1] = \neq t$
$turn = 0 \neq 0$

$i=1$
$J=0$

P_0
↓

choosing = [f f f] choosing[n]

number = [0 0 0] number[n]

(3)

P_0 P_1 P_2

(1)

$J=0$ $J=1$ $J=2$

$(2,0) < (3,2)$

$(1,1) < (1,1)$

$(a,b) < (c,d)$

if $a < c \Rightarrow t$

$(2,2) < (3,3)$

$(2,2) < (2,3)$

if $(a == c)$

if $b < d \Rightarrow t$

```
do{
    {choosing[i]=true;
    number[i]=max(number[0], number[1], ..., number[n-1]) + 1;
    choosing[i]=false;
    for(j=0; j<n; j++){
        while(choosing[j]);
        while((number[j]!=0)&&((number[j], j)<(number[i], i)));
    }
    < critical section >
    number[i]=0;
    < remainder section >
}while(true);
```

P_0

$(1,0) < (2,0)$ $J=0$, $J=1$, $J=2$

$(2,0) < (2,0)$

S = +0+0+1

while(S<=0);

S--;

P_0 P_1 P_2 P_3 . . . .

wait(S) wait(S)

<CS> <CS>

signal(S) signal(S)

wait(S){ while(S<=0); S--; }

signal(S){ S++; }

P_0 P_1 P_2

<CS> wait(S) wait(S)

signal(S) <CS>