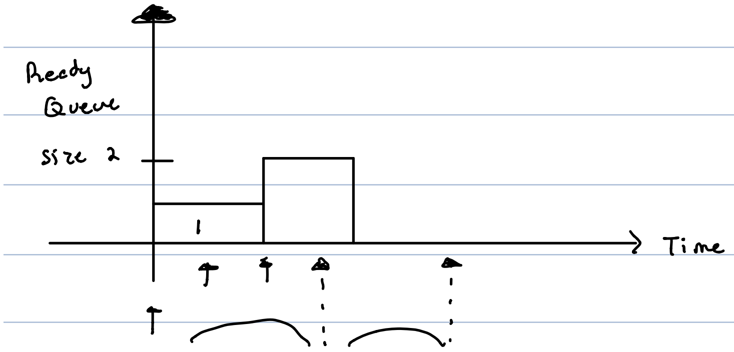


Process Sync.

1. Disable / Enable Interrupts

2. Simple Hardware Instructions [atomic

→ Test and set

↳ First process that finds lock = FALSE, set to True and enters critical section

→ swap

①

n process
boolean waiting [n]
boolean lock;
} Init to FALSE

②

```
Pi : do {
        waiting [i] = true;
        key = true;
      while (waiting[i] && key)

        key = TestAndSet(&lock)
    waiting[i] = False;
  }
```

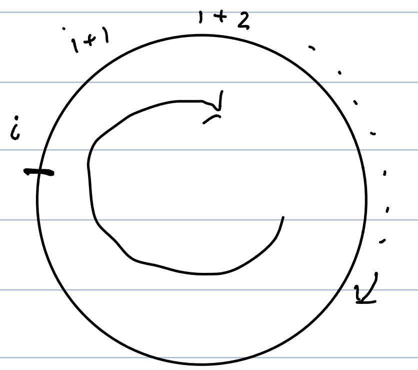
C.S ③

```
j = (i+1) % n;
while ((j != && waiting[j]
          j = (j+1) % n;
if   j == i
       lock = False
else
     waiting [j] = False
```

exit

R.S (Remainder Section) ④

```
) while (1);
```



Always comes back

P$_2$ ✓

P$_5$ ↻

↻

P$_3$

$\boxed{3}$ Semaphores

An integer value that can only be accessed by two automic operations.

- Wait()
- Signal ()

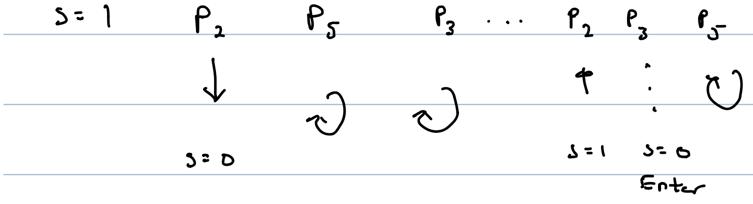
```
Signal (s) {
      s++;
}
```

binary 0 1

couting

```
wait(s) {
    while (s ≤ 0) ;// do nothing
      s--;
}
```

```
Pi   do {
       wait(s)
        C.S
       signal (s)
        R.S
     } while (1)
```



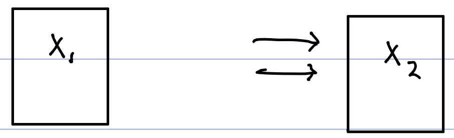
Init s = 0;

P_0 P_1

wait(s)

$\boxed{x_1}$

$\boxed{x_2}$



Signal (s);

Counting Semaphores allows up to n processes inside the C.S.

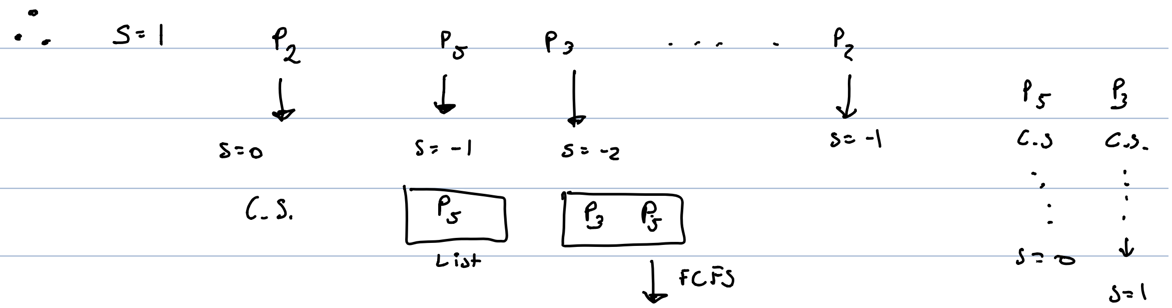
→

Idea: block a process not a spin loop

```
typedef struct {
        int value;
        struct process * list;
} semaphore;
```

```
signal (semaphore *s)
   s→value ++;
   if (s→value <= 0){
      remove P from s→list;
      wake up (P)
   }
}
```

```
wait (semaphore * S) {
     S → value --;
     if (s→value < 0)
            // Add this process to s→list block
}
```


∴ S=1
P_2
P_5
P_3
. . . .
P_2
S=0
S=-1
S=-2
S=-1
C.S.
P_5
List
P_3 P_5
FCFS
P_5
P_3
C.S
C.S.
S=0
S=1