Process Synchronization

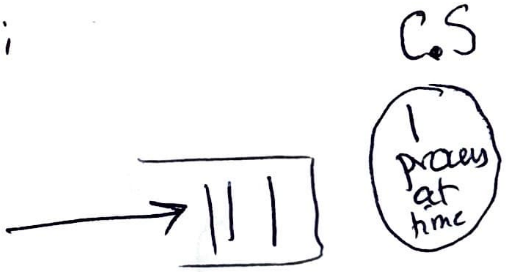
Semaphores

Integer variable than can only be acused by

wait( ) Signal( ) ⇒ atomic operations

```
Pi  while( 1 ){
        wait (S);
          C.S
        Signal (S);
    };
```

```
typedef {
    int  valve ;
    queue *  Waiting Process;
} semaphore ;
```



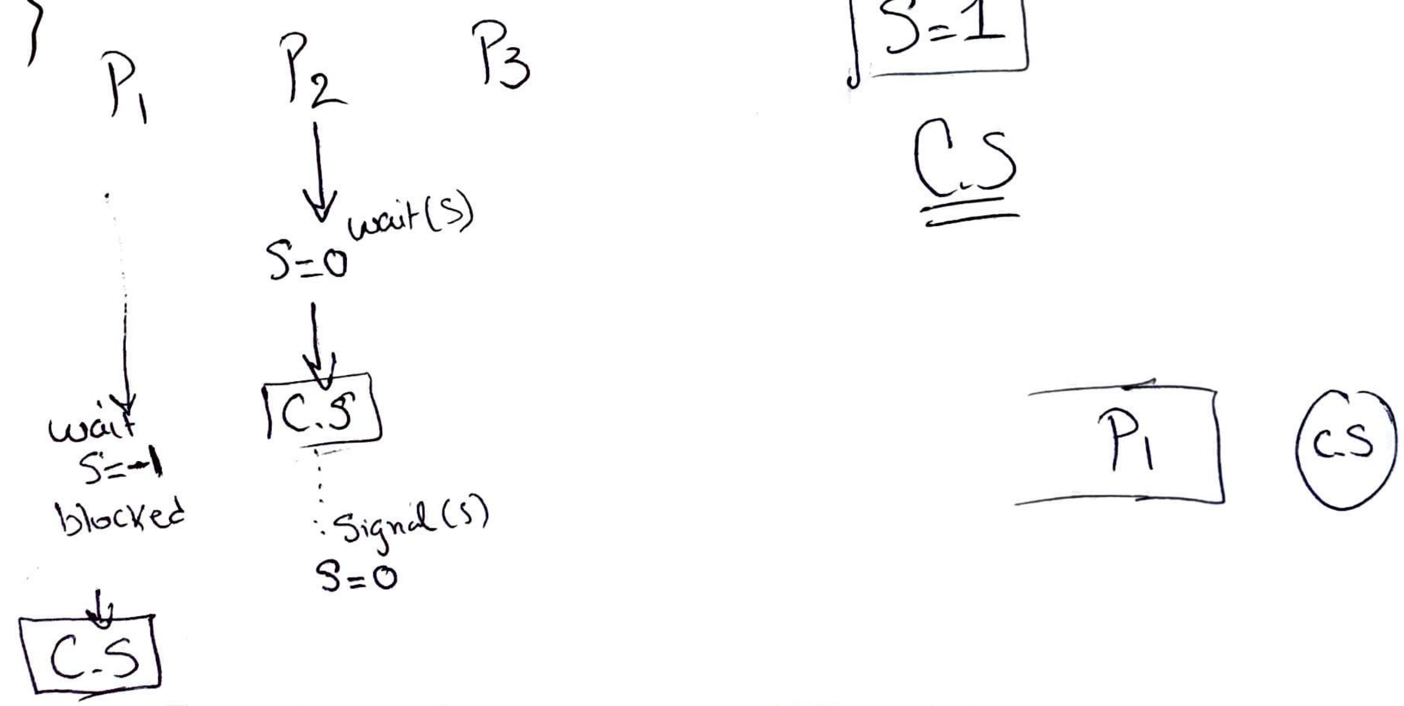
* No waste in cyles

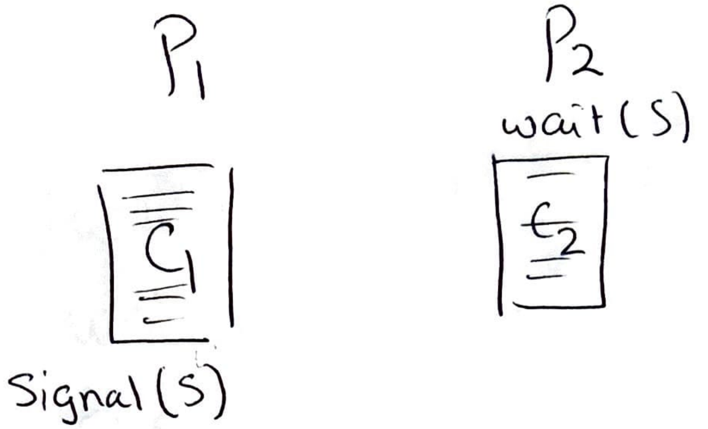
* No Starvation

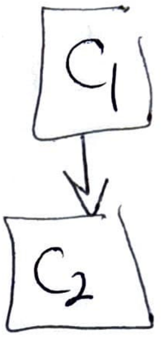
S=1
↓ S=0

```
wait(S){
        S--;
        If(S < 0){
            add process to S→WaitingProcess;
            block(P);
        }
}

Signal(S){
        S++;
        If (S <= 0){
            pop P from S→waitingProcesses;
            wakeup(P);
        }
}
```







C1 executes before C2.

$\boxed{S=0}$

$\underline{S=1}$

```
Pi :  Signal (S)
        C.S
      wait (S)
```

No mutual exclusion

S = 1.

```
Pi :  wait (S)
        C.S
      wait (S)
```

No progress
Starvation

S=1   q=1

P1

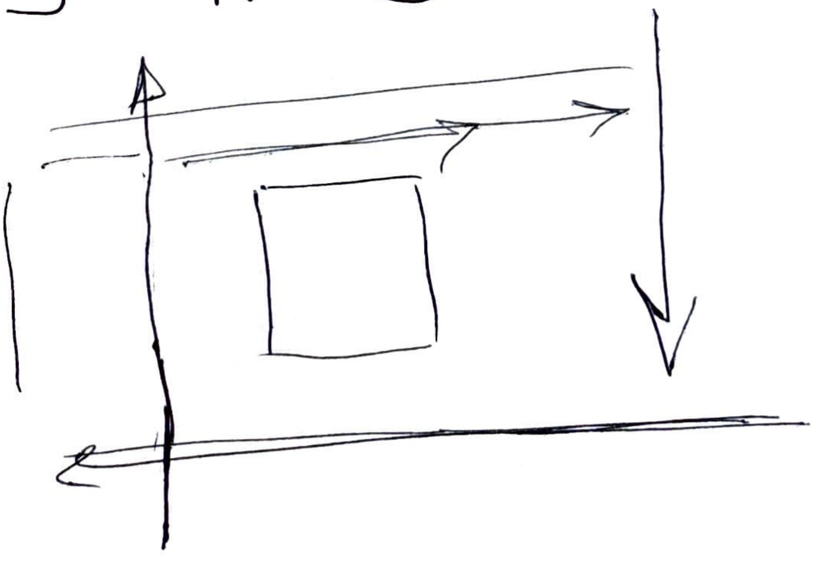
$\boxed{1}$ wait (S) $\boxed{S=0}$
wait (q)
C.S
Signal (q)
Signal (S)

P2

wait(q) $\boxed{2}$ q=0
wait(S)
C.S
Signal(S)
Signal(q)

Deadlock

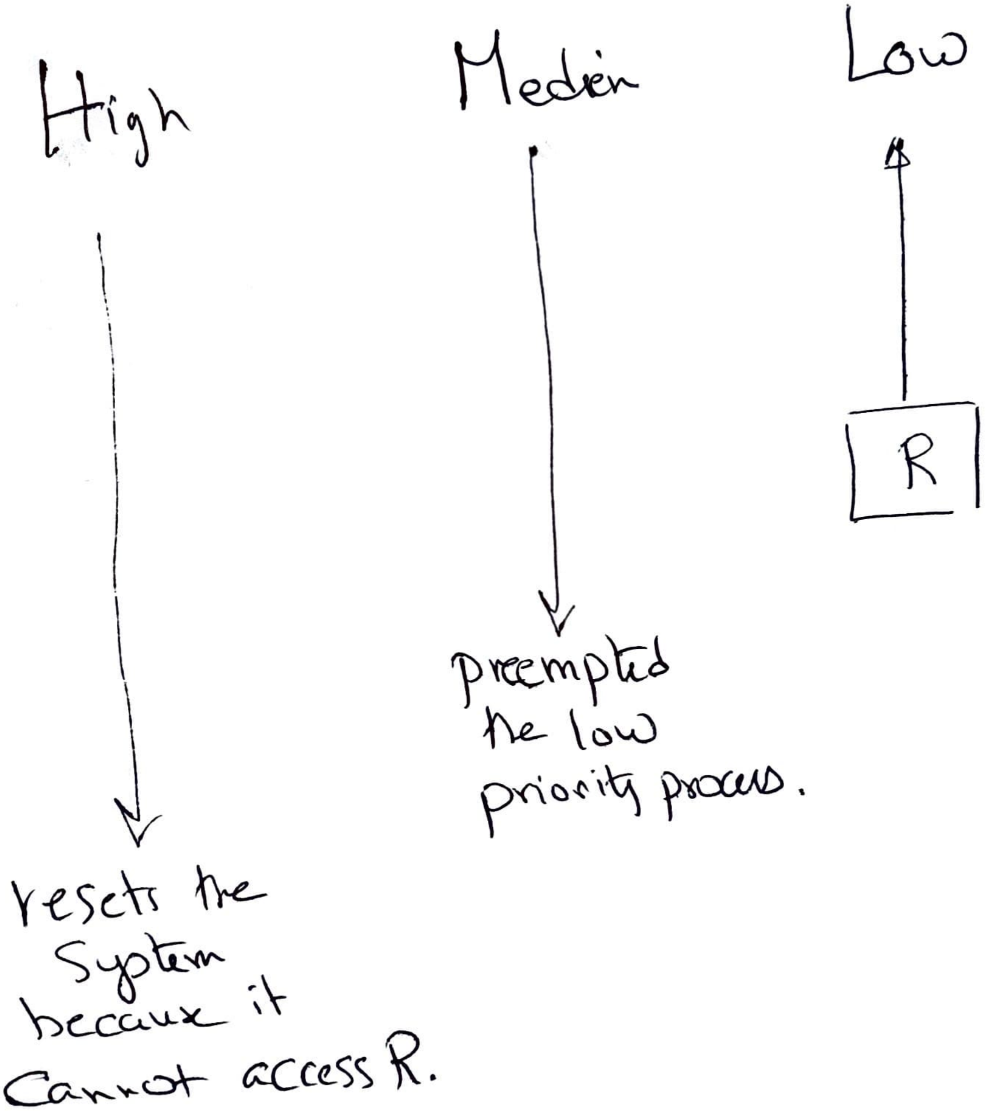
A Set of processes are deadlocked if each is waiting for an event that Can only happen by a waiting process.



Priority Inversion

When a high priority process waits for a resource held by a lower priority process.

Mars landing Pathfinder 1997.


High
Medien
Low
R
preempted the low priority process.
resets the system because it Cannot access R.

Priority Inheritance

Give the low priority process the priority of the process waiting until it is done. then it gets back its priority (low).

Classical Problems of Synchronization

1 Bounded buffer problem

semaphores

- mutex : buffer access (Prevent the producer and Consumer from accessing the same location at same time));
- empty : # of empty of buffers
- Full : # of Full buffers

mutex=1
empty=n
Full=0

Producer

```
do{
    item= produce item;
    wait(empty);
    wait (mutex);
    put item inbuffer();
    Signal (mutex);
    Signal ( Full);
}while(1)
```

Consumer

```
do{
wait (Full);
wait (mutex);
    item =getitem()!
Signal (mutex);
Signal (empty);
consume-item();
}while (1);
```

n= 2

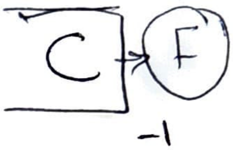
m=1
E=2
F=0

Init

C

⋮

wait(F) →



⋮

↓

allowed to run.

P1

E= 1
M=0

item

m=1
F=0

//

E=0
m=1

item

m=0
F= 1

E=-1