N processes

var choosing: array[0,...n-1] of Boolean;

number:array[0...n-1] of integer

(a,b)<(c,d) if a<c or if a=c and b<d

max(a0,….an-1)is a number,k such that k>=ai for i=0...n-1

```
choosing[i]=true
number[i]=max(number[0],…….number[n-1])+1
choosing[i]=false
for(j=0 to n-1)
Do
    While choosing[j] do no_operation
            While number[j]!=0 and (number[j],j)<(number[i],i) do no_operation
            end
            CS
            number[i]=0;
                ....
```

semaphores

.OS provides an tools to solve the CS problem

Semaphores.

.Semaphore is a data type like int/float/char...

.Two operations are allowed in semaphore

.Wait and signal(both atomic operation)

.S is semaphore variable

wait(s): while s<=0 do no_op signal(s):s=s+1

s=s-1

Use of semaphore

wait(s)

Critical section

signal(s)

P1's CS should execute after P2's CS

P1	p2
wait(s)	CS
CS	signal(s)

The disadvantage with the definition of semaphore given above is that it requires busy waiting.. while entering the CS p1 uses the CPU cycles to get permission and waiting for it...so CPU cycles are wasted which could be given to other processes.

solution

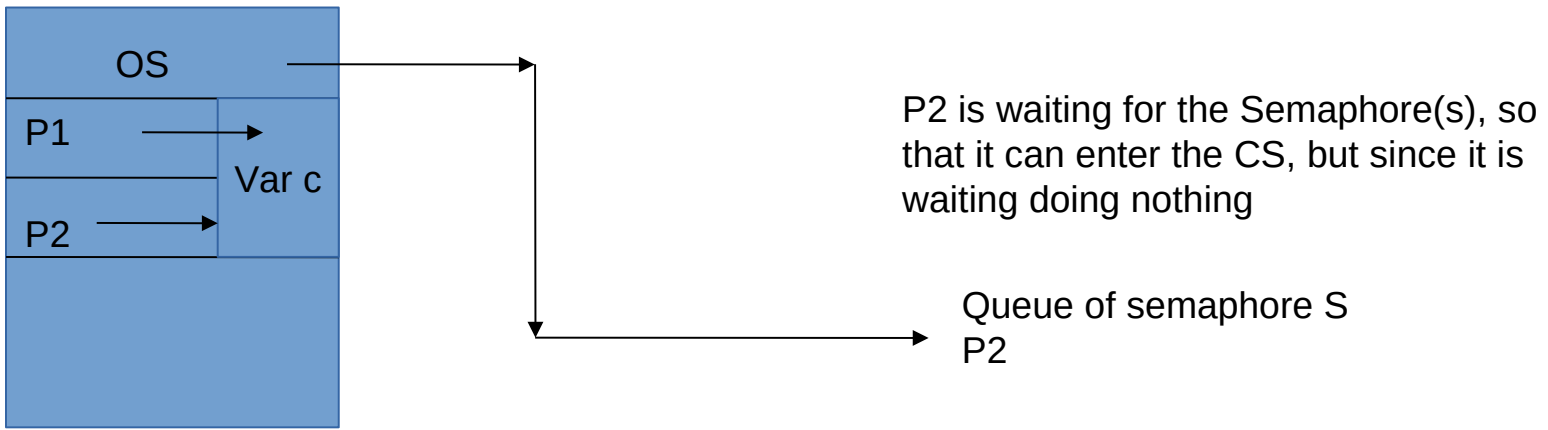
So the definition of the semaphore could be modified.

There is a waiting queue associated with each semaphore.

When a process executes the wait operation and find that the semaphore value is not positive, instead of busy waiting , it blocks itself.

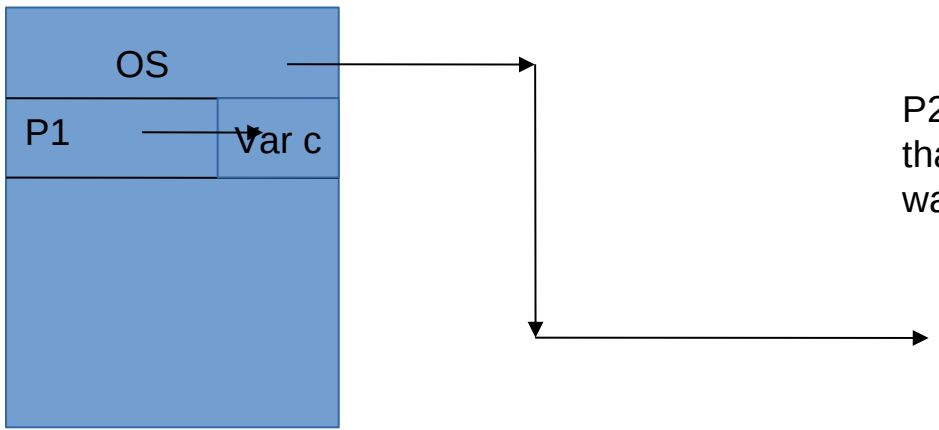
The blocking operation includes

- Placing this process in the waiting queue of the semaphore.
- Switch its state to waiting state.



The blocking operation includes

- Placing this process in the waiting queue of the semaphore.
- Switch its state to waiting state.



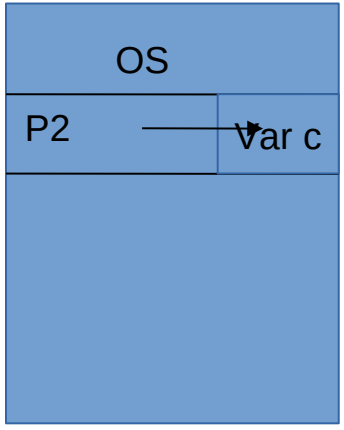
P2 is waiting for the Semaphore(s), so that it can enter the CS, but since it is waiting doing nothing

Queue of semaphore S

P2 P2 is in waiting state now

The blocking operation includes

- Placing this process in the waiting queue of the semaphore.
- Switch its state to waiting state.



P1 finish it s CS so signal operation is performed by P1, then OS wakes up P2 and P2 state becomes ready.

Dead lock and Starvation

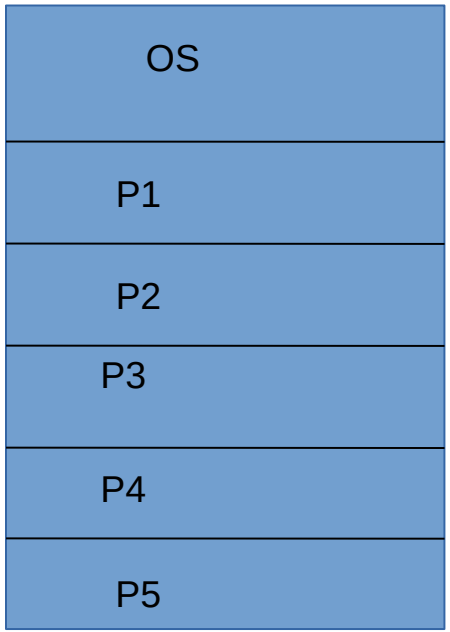
P0	P1
wait(s)	wait(q)
wait(q)	wait(s)
signal(s)	signal(q)
signal(q)	signal(s)

Both processes P0 and P1 are waiting for each other.....
This kind of situation is known as DEAD LOCK.

Indefinite blocking or starvation is a situation in which processes wait within the semaphore.

It may occur if the order in which processes are removed from the list associated with a semaphore is lIFO.

CPU Scheduling(short term scheduler)



P1
P2
P3
P4
P5
Ready processes

But CPU can execute one process at a time

CPU scheduling algo will decide which process gets the CPU next.

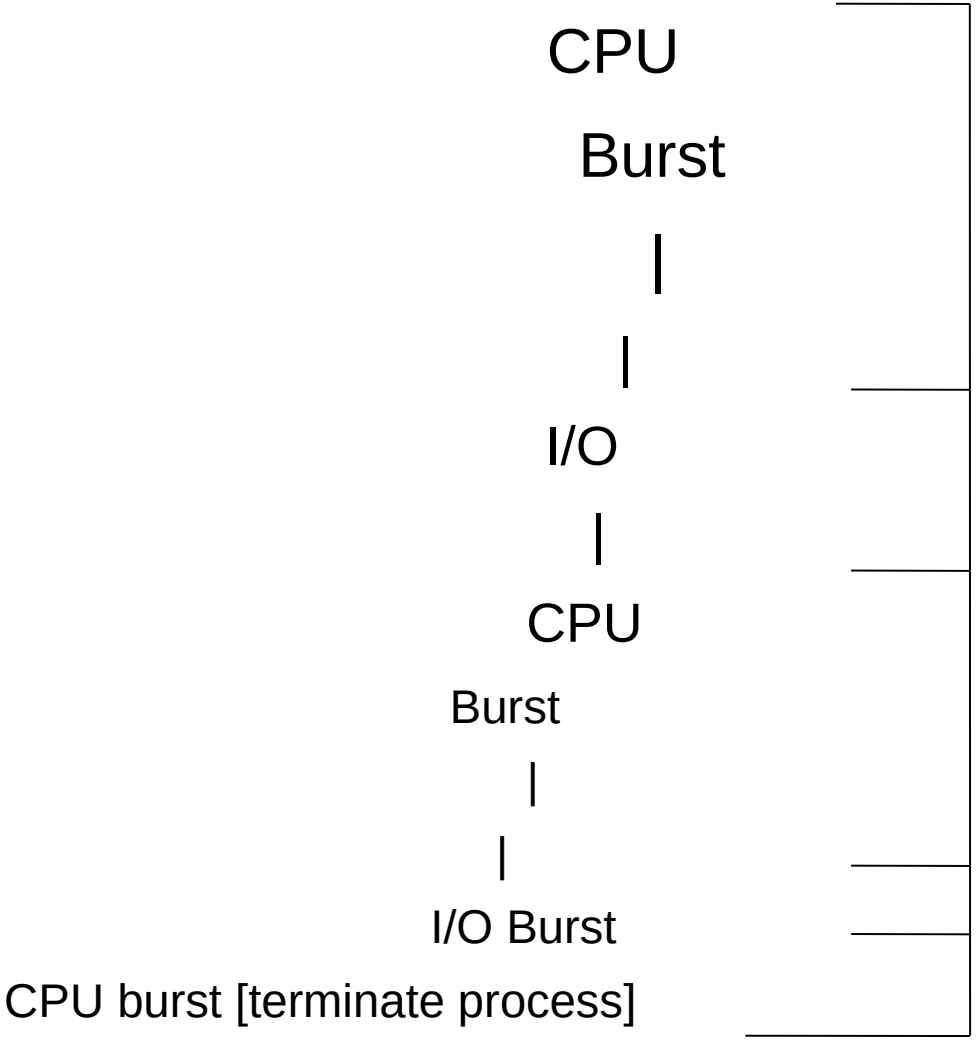
CPU-I/O Burst Cycle

Process execution consist of a cycle of CPU execution (CPU burst) and I/O wait(I/O burst) .

Processes alternate back and forth between these two states.

The last CPU burst will end with a system request to terminate execution rather than with an I/O burst.

```
#include<stdio.h>
Int main()
{
    Int x=1,y,fact=1;
    printf("\n enter the limit");
    Scanf("%d",&y);
    for(x=1;x<y;x++)
            {
            fact=fact*x;
            }
            printf("\n fact=%d",fact);
            }
```



Pre-emptive and Non Pre-emptive scheduling

Pre-emptive scheduling

CPU scheduling decision may take place under the following circumstances:

1. when a process switches from the running state to the waiting state.
2. when a process switches from the running state to the ready state.
3. when a process switches from the waiting state to the ready state.
4. when a process terminate.