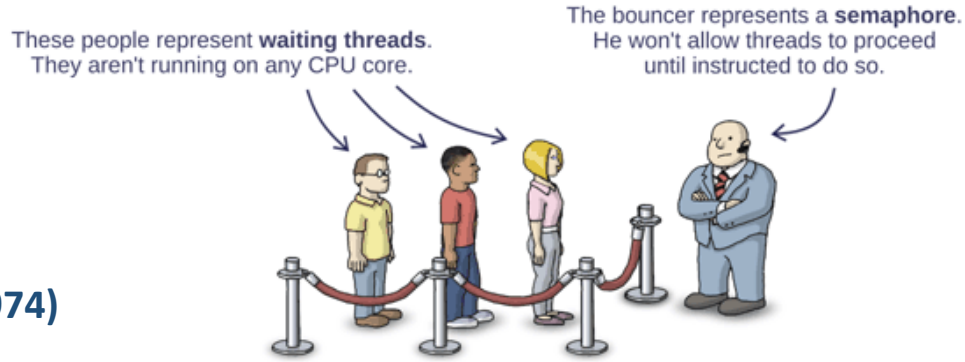
Algoritmos y Programación Paralela

Dra. Ing. Ana Cori Morón

INTRODUCCIÓN

- En 1965, Edger Dijkstra estaba interesado en el desarrollo de un sistema operativo con un conjunto de procesos cooperantes y mecanismos eficientes y fiables para dar soporte a dicha cooperación. Para ello propuso que dos o mas procesos puedan cooperar por medio de señales simples, de forma que se pueda obligar a detener un proceso hasta que reciba otra señal específica. Para esta señalización, propuso variables especiales llamadas semáforos.



se usaron por primera vez en el sistema operativo THEOS (1974)

DEFINICIÓN

Un semáforo bloquea un proceso cuando éste no puede realizar la operación deseada, en vez de desperdiciar tiempo de CPU

- Un semáforo es un tipo de dato abstracto que solo admite valores enteros no negativos y que varian su valor de una en una unidad, los semáforos se pueden clasificar en:
- Semáforo contador, que es aquel que permite que su valor pueda ser cualquier valor no negativo.
- Semáforo binario, también denominado mutex, que es aquel cuyo valor solo puede ser 0 ó 1.

SEMÁFORO S=0,1,2…

Con los procesos bloqueados se forma una cola de tipo FIFO

P1 BLQ	**P2 BLQ**	**P3 BLQ**	**…**			

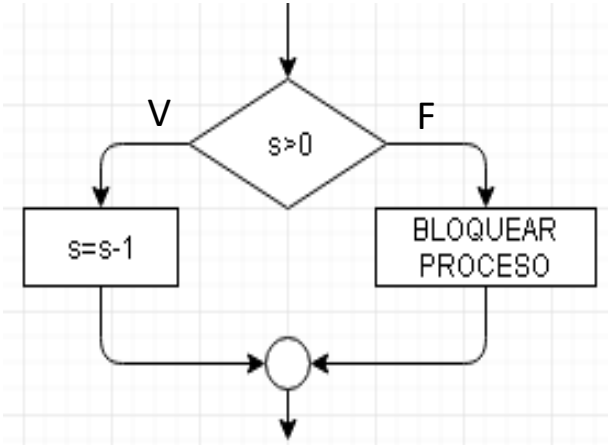
Es un TAD formado por un valor numérico y una cola de procesos bloqueados

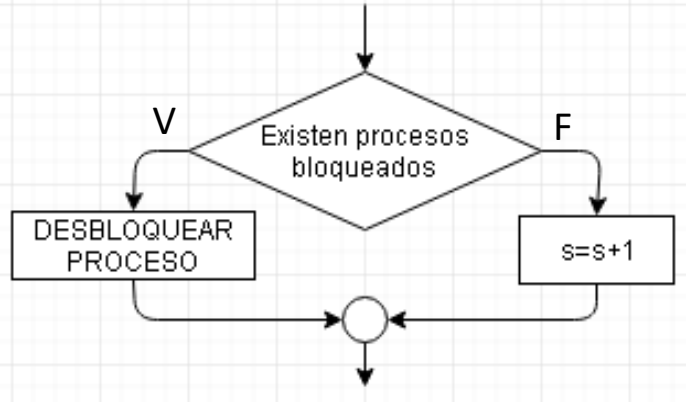
Características de los semáforos binarios

- Los semáforos binarios optimizados para exclusión mutua también son conocidos como Cerrojos, mutex o locks.
- Inicializados a 1 por defecto
- Sólo el proceso que hizo wait puede hacer signal.

OPERACIONES DEL SEMÁFORO

- Las operaciones que se pueden realizar sobre los semáforos son:
- Wait: Decrementa el valor de un semáforo si este es mayor a 0, si el valor es 0 se bloqueará.
- Signal: Desbloqueará algún proceso bloqueado por el semáforo. En caso que no haya ningún proceso bloqueado, incrementa el valor del semáforo.
- Initial: Inicializa el valor del semáforo a un valor mayor o igual a 0.





RESOLUCIÓN DE PROBLEMAS USANDO SEMÁFOROS

- EXCLUSIÓN MUTUA, Para implementar una solución a la exclusión mutua se debe hacer uso de un semáforo binario inicializado a 1.

```
process P1
 begin
   …
   wait (s)
   SECCIÓN CRÍTICA
   signal(s)
    ….
end;
```

```
process P2
  begin
    …
    wait (s)
    SECCIÓN CRÍTICA
    signal(s)
    ….
  end;
```

s: semaphore
initial(s,1)

RESOLUCIÓN DE PROBLEMAS USANDO SEMÁFOROS

- CONDICIÓN DE SINCRONIZACIÓN, Se hace uso de un semáforo contador cuando un proceso ha hecho que se cumpla determinada condición lo indica ejecutando un SIGNAL sobre el semáforo, cuando un proceso espera a que una condición sea cumplida, ejecuta una operación WAIT.

```
process P1
begin
        a;
        b;
end;
```

```
process P2
begin
        c;
        d;
end;
```

Problema

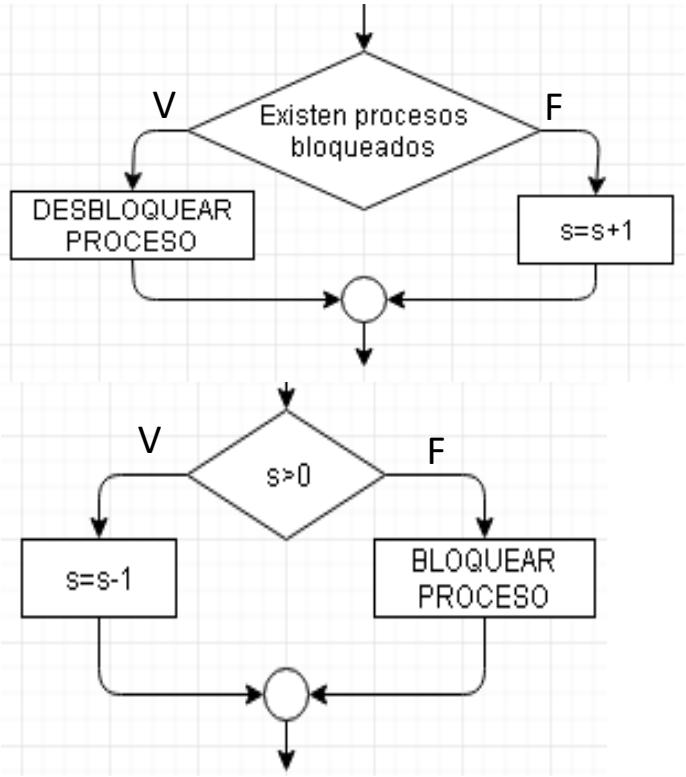
Asegúrese que P2 no pueda ejecutar “d” hasta que P1 haya ejecuta ”a”.

SOLUCIÓN

s: semaphore
initial(s,0)

```
process P1
begin
    repeat
        a;
        signal(s);
        b;
    forever
end;
```

```
process P2
Begin
  repeat
        c;
        wait(s);
        d;
  forever
end;
```



Suponiendo que empieza a ejecutar P1

T	0	1	2	3	4	5	6
S	0	0	1	1	0	0	0
P1		a	Sig(s)			b	
P2				c	Wai(s)		d

Suponiendo que empieza a ejecutar P2

T	0	1	2	3	4	5	
S	0	0	0 Blq P2	0	0 Dblq P2	0	
P1				a	Sig(s)		
P2		c	Wai(s)			d	

Ejemplo: suponiendo los siguientes procesos

Problema Asegúrese que P2 no pueda ejecutar “d”, si P3 haya ejecutado ”e” o P1 haya ejecutado ”a”.

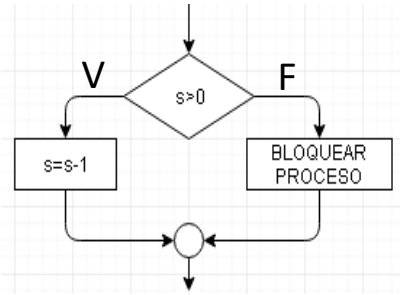
```
process P1
begin
        a;
        b;
end;
```

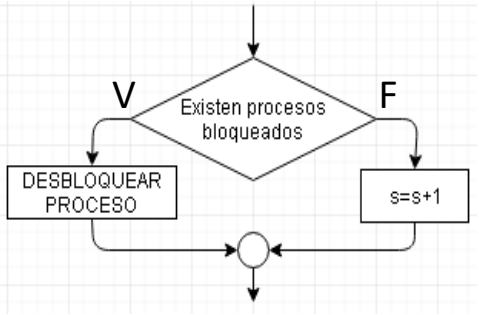
```
process P2
begin
        c;
        d;
end;
```

```
process P3
begin
        e;
        f;
end;
```

SOLUCIÓN

s: semaphore
initial(s,0)






```
process P1
begin
     repeat
          a;
          signal(s);
          b;
     forever
end;
```

```
process P2
begin
     repeat
          c;
          wait(s);
          d;
     forever
end;
```

```
process P3
begin
     repeat
          e;
          signal(s);
          f;
     forever
end;
```

Suponiendo que empieza a ejecutar P3

T	0	1	2	3	4	5	6	7	8	9
S	0		1	1	0	0	1	1	1	1
P1						a	Sig(s)			b
P2				c	Wai(s)				d	
P3		e	Sig(s)					f		

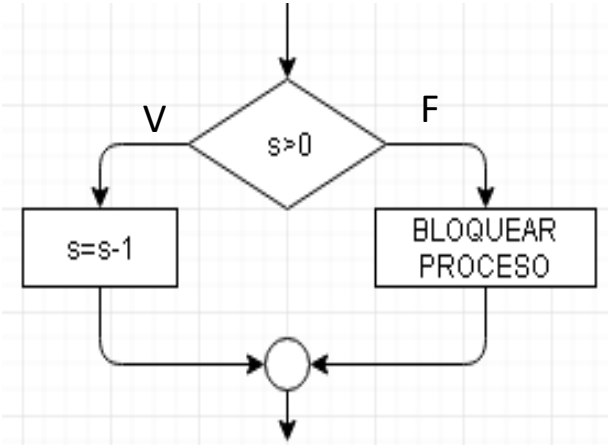
SOLUCIÓN

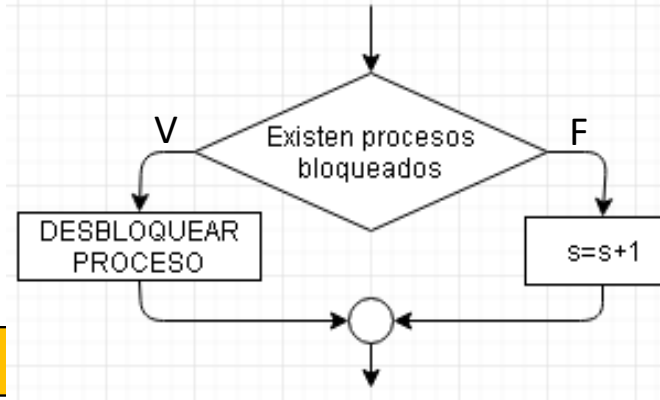
s: semaphore
initial(s,0)

```
process P1
begin
    repeat
        a;
        signal(s);
        b;
    forever
end;
```

```
process P2
begin
    repeat
        c;
        wait(s);
        d;
    forever
end;
```

```
process P3
begin
    repeat
        e;
        signal(s);
        f;
    forever
end;
```





Suponiendo que empieza a ejecutar P2

T	0	1	2	3	4	5	6	7	8
S	0	0	0 Blq P2	0	0 Dbl P2	0	0	1	
P1							a	Sig(s)	
P2		c	Wai(s)			d			
P3				e	Sig(s)				f

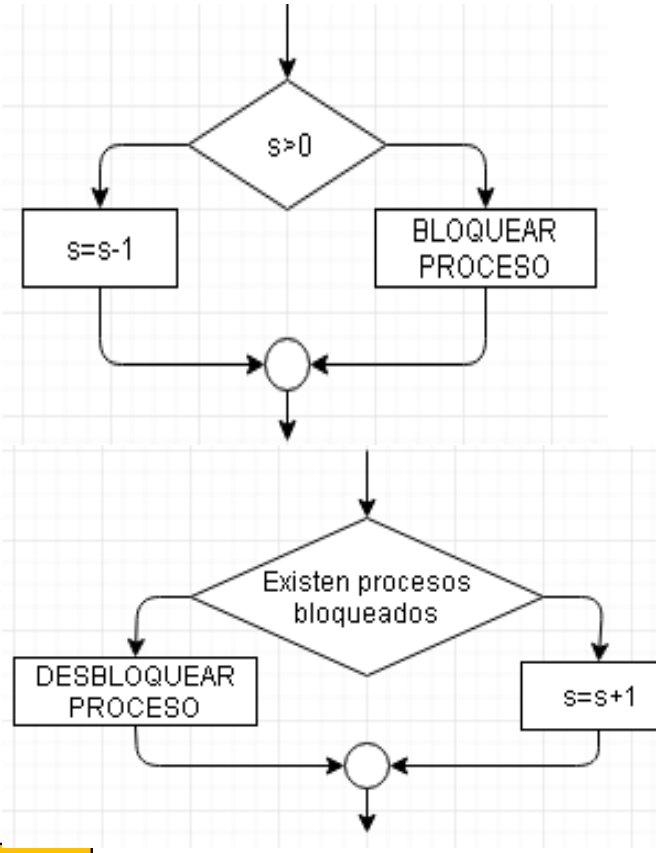
SOLUCIÓN

s: semaphore
initial(s,0)

```
process P1
begin
    repeat
        a;
        signal(s);
        b;
    forever
end;
```

```
process P2
begin
    repeat
        c;
        wait(s);
        d;
    forever
end;
```

```
process P3
begin
    repeat
        e;
        signal(s);
        f;
    forever
end;
```



Suponiendo que empieza a ejecutar P1

T	0	1	2	3	4	5	6	7	8
S	0		1	1	0	0	1	1	
P1		a	Sig(s)						b
P2				c	Wai(s)			d	
P3						e	Sig(s)		

En caso del enunciado anterior, si la condición lógica fuese una *"y"*

```
process P1
begin
        a;
        b;
end;
```

```
process P2
begin
        c;
        d;
end;
```

```
process P3
begin
        e;
        f;
end;
```

Deseamos que P2 ejecute "***d***" si P3 ejecutó "***e***" **y** P1 ejecutó "***a***".

SOLUCIÓN

```
process P1
begin
  repeat
        a;
        signal(s);
        b;
    forever
end;
```

```
process P2
begin
  repeat
        c;
        wait(s);
        wait(t);
        d;
  forever
end;
```

```
process P3
begin
   repeat
        e;
        signal(t);
        f;
   forever
end;
```

s,t: semaphore
initial(s,0)
Initial(t,0)

Suponiendo que empieza a ejecutar P3

Tmp	0	1	2	3	4
S	0				
T	0		1		
P1					
P2				c	Wai(s)
P3		e	Sig		

Suponiendo que empieza a ejecutar P2

Tmp	0	1	2	3	4
S	0		Blq		
T	0				
P1					a
P2		c	Wai		
P3				e	

Suponiendo que empieza a ejecutar P1

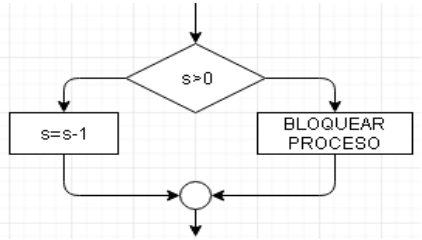
Tmp	0	1	2	3	4
S	0		1		0
T	0				
P1		a	Sig		
P2				c	Wai
P3					

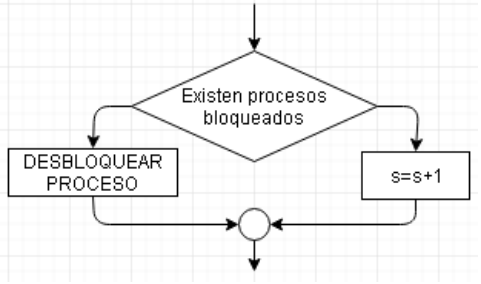
SOLUCIÓN

```
process P1
begin
  repeat
          a;
          signal(s);
          b;
     forever
end;
```

```
process P2
begin
  repeat
          c;
          wait(s);
          wait(t);
          d;
  forever
end;
```

```
process P3
begin
   repeat
          e;
          signal(t);
          f;
   forever
end;
```





Suponiendo que empieza a ejecutar P3

s,t: semaphore
initial(s,0)
Initial(t,0)

Tmp	0	1	2	3	4	5	6	7	8
S	0		0	0	0 Blq P2	0	0 DBlq P2	0	
T	0		1	1	1	1	1	0	
P1						a	Sig(s)		
P2				c	Wai(s)			Wai(t)	d
P3		e	Sig(t)						

SOLUCIÓN

```
process P1
begin
  repeat
        a;
        signal(s);
        b;
    forever
end;
```

```
process P2
begin
  repeat
        c;
        wait(s);
        wait(t);
        d;
  forever
end;
```

```
process P3
begin
   repeat
        e;
        signal(t);
        f;
   forever
end;
```

s,t: semaphore
initial(s,0)
Initial(t,0)

Suponiendo que empieza a ejecutar P2

Tmp	0	1	2	3	4	5	6	7	8
S	0	0	0 Blq P2	0	0	0	0 Dblq P2	0	0
T	0	0	0	0	1	1	1	0	0
P1						a	Sig		
P2		c	Wai					Wai	d
P3				e	Sig				

SOLUCIÓN

```
process P1
begin
  repeat
        a;
        signal(s);
        b;
    forever
end;
```

```
process P2
begin
  repeat
        c;
        wait(s);
        wait(t);
        d;
  forever
end;
```

```
process P3
begin
   repeat
        e;
        signal(t);
        f;
   forever
end;
```

s,t: semaphore
initial(s,0)
Initial(t,0)

Suponiendo que empieza a ejecutar P1

Tmp	0	1	2	3	4	5	6	7	8	9
S	0		1	1	0	0	0	0	0	
T	0	0	0	0	0	0	1	1	0	
P1		a	Sig(s)					b		
P2				c	Wai(s)				Wai(t)	d
P3						e	Sig(t)			

LINKS DE VIDEOS

- https://www.youtube.com/watch?v=e2ujg5K310s
- https://www.youtube.com/watch?v=Xqtiry7RKIw