

Paper Cooperating Sequential Processes de Edsger Dijkstra

Capítulos 3 y 4

Semáforos

Jorge Anca



Temario

- El problema de Exclusión mutua
- Semáforos
 - El problema de Productor y Consumidor con buffer ilimitado
 - El problema del Barbero durmiente
 - El problema del Productor y Consumidor con buffer limitado
- Ejemplos de Semáforos en algunos lenguajes

El problema de exclusión mutua

Sección crítica: es un segmento de código donde se puede acceder a variables compartidas por más de un proceso. Una acción atómica es requerida en la sección crítica, sólo un proceso puede ejecutar dicha sección a la vez, los demás deben esperar.

Soluciones que utilizan Busy Waiting:

- Consumo de tiempo de procesador
- El acceso a la misma porción memoria por varios procesos obliga a que algunas deban esperar a otros

Semáforos

- Inventados por [Edsger Dijkstra](#) en 1965
- Son variables enteras no negativas
- Tipos
 - Binarios
 - Generales
- Operaciones
 - V(del holandés Verhoog, incrementar):
 - $V(S_i) \Rightarrow S_i + 1$
 - Operación atómica
 - P(del holandés Prolaag, palabra inventada por Dijkstra, decrementar):
 - $P(S_i) \Rightarrow S_i - 1$
 - Operación atómica
 - Si $S_i = 0$, $P(S_i) \Rightarrow \text{wait}$

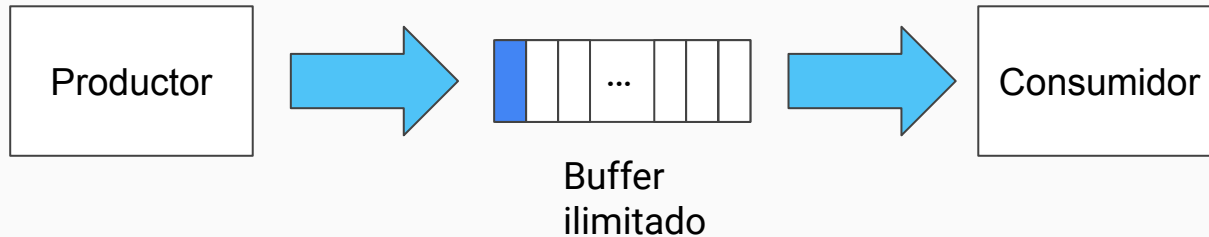


IBM 360

Semáforos: Problema del productor y consumidor con buffer ilimitado

Supuestos:

- Productor: produce información de manera cíclica
- Consumidor: consume información también de forma cíclica
- Buffer: capacidad ilimitada



Semáforos: Solución 1 al Problema de productor y consumidor

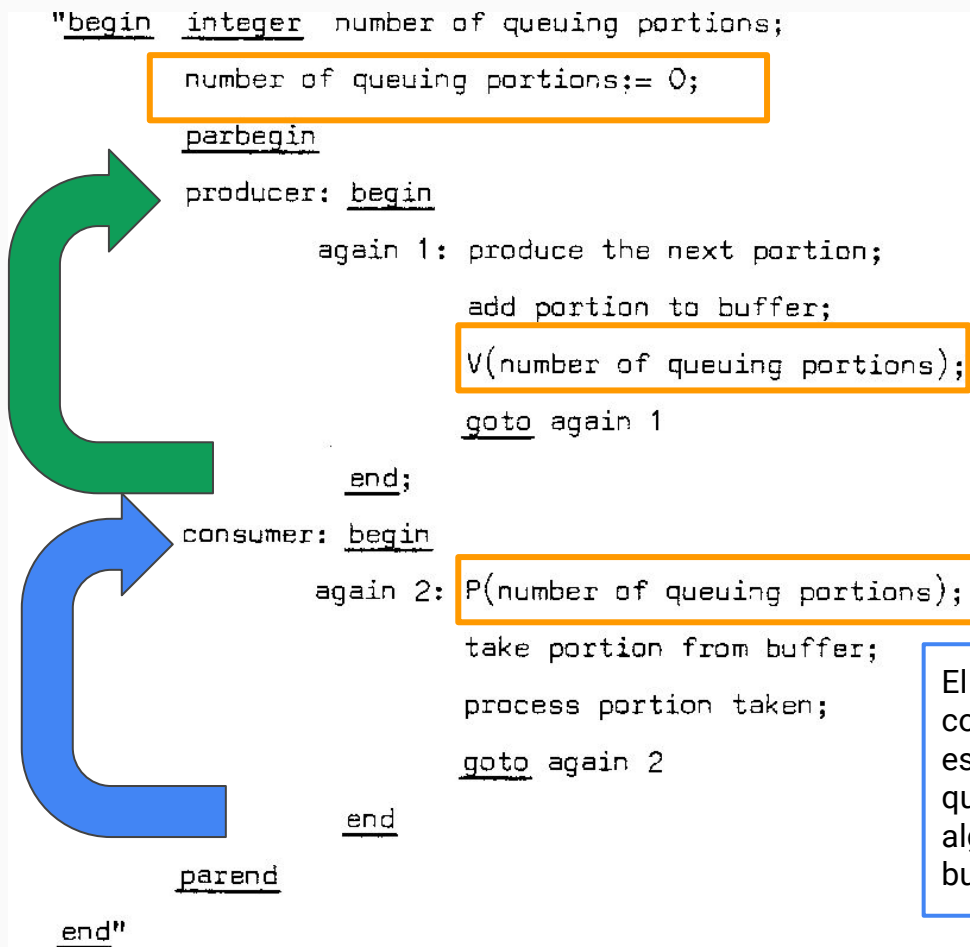
Consecuencias:

- El productor y consumidor operan de manera asincrónica

Problemas del algoritmo:

- Puede haber interferencia entre la escritura y lectura del buffer. El consumidor puede leer el buffer mientras el productor esté escribiendo el siguiente segmento de datos

```
"begin integer number of queuing portions;  
    number of queuing portions:= 0;  
    parbegin  
        producer: begin  
            again 1: produce the next portion;  
                    add portion to buffer;  
                    V(number of queuing portions);  
                    goto again 1  
        end;  
        consumer: begin  
            again 2: P(number of queuing portions);  
                    take portion from buffer;  
                    process portion taken;  
                    goto again 2  
        end  
    parend  
end"
```



El consumidor espera hasta que haya algo en el buffer

Semáforos: Solución 2 al Problema de productor y consumidor

Se agrega el semáforo buffer manipulation:

- Valores:
 - 0: se está escribiendo o leyendo en el buffer
 - 1: no se está accediendo al buffer

Consecuencias:

- Se puede extender el problema a varios productores y consumidores

```
"begin integer number of queuing portions, buffer manipulation;
```

```
number of queuing portions:= 0;
```

```
buffer manipulation:= 1;
```

```
parbegin
```

```
producer: begin
```

```
again 1: produce next portion;
```

```
P(buffer manipulation);
```

```
add portion to buffer;
```

```
V(buffer manipulation);
```

```
V(number of queuing portions);
```

```
goto again 1
```

```
end;
```

```
consumer: begin
```

```
again 2: P(number of queuing portions);
```

```
P(buffer manipulation);
```

```
take portion from buffer;
```

```
V(buffer manipulation);
```

```
process portion taken;
```

```
goto again 2
```

```
end
```

```
parend
```

```
end"
```

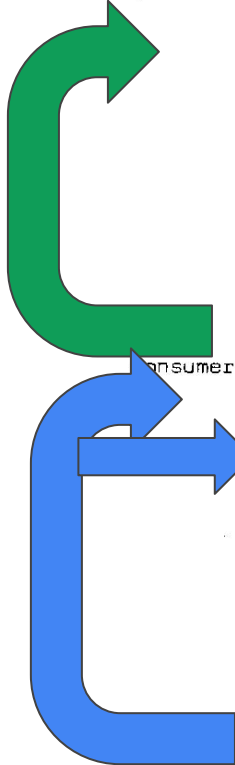
Hasta que el consumidor no haya leído el segmento, el productor no puede escribir en el buffer

El orden importa sino ocurre un deadlock

Semáforos: Solución 3 al Problema de productor y consumidor

- Solución que "demuestra" que se puede omitir el uso de semáforos no binarios

```
"begin integer numqueupor, buffer manipulation, consumer delay;
numqueupor:= 0; buffer manipulation:= 1; consumer delay:= 0;
parbegin
producer: begin
again 1: produce next portion;
P(buffer manipulation);
add portion to buffer;
numqueupor:= numqueupor + 1;
if numqueupor = 1 then V(consumer delay);
V(buffer manipulation);
goto again 1
end;
consumer: begin integer oldnumqueupor;
wait: P(consumer delay);
go on: P(buffer manipulation);
take portion from buffer;
numqueupor:= numqueupor - 1;
oldnumqueupor:= numqueupor;
V(buffer manipulation);
process portion taken;
if oldnumqueupor = 0 then goto wait else goto go on
end
parend
end"
```



Semáforos binarios

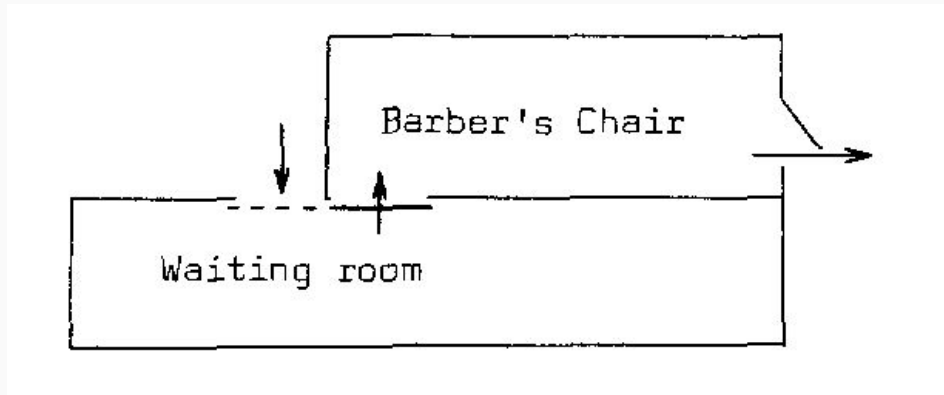
El consumidor no lee si no hay elementos en el buffer

Mientras el buffer no este vacio el consumidor sigue tomando segmentos del buffer a su velocidad

Semáforos: Problema del barbero durmiente

Supuestos:

- Los clientes entran de uno a la sala de espera
- El barbero cuando termina con cliente entra en la sala de espera y si hay clientes llama al siguiente sino se echa a dormir
- Si el cliente encuentra al barbero durmiendo en sala de espera lo despierta



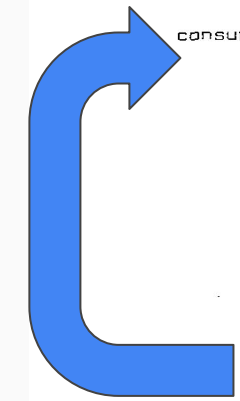
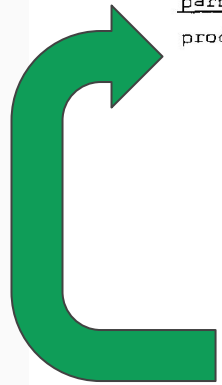
Semáforos: Solución al Problema del barbero durmiente

- No importa si el buffer está vacío sino si hay consumidores que quieran datos del buffer

Conclusiones:

- Los árboles generales pueden ser reemplazados por binarios
- Sin embargo, puede ser más fácil de implementar

```
"begin integer numqueupor, buffer manipulation, consumer delay;
numqueupor:= 0; buffer manipulation:= 1; consumer delay:= 0;
parbegin
producer: begin
again 1: produce next portion;
P(buffer manipulation);
add portion to buffer;
numqueupor:= numqueupor + 1;
if numqueupor = 0 then
begin V(buffer manipulation);
V( consumer delay) end
else
V(buffer manipulation);
goto again 1
end;
consumer: begin
again 2: P(buffer manipulation);
numqueupor:= numqueupor - 1;
if numqueupor = - 1 then
begin V(buffer manipulation);
P(consumer delay);
P(buffer manipulation) end;
take portion from buffer;
V(buffer manipulation);
process portion taken;
goto again 2
end
end"
parend
end"
```



El consumidor trata de leer en el buffer sin esperar sino no hay nada, espera

Semáforos: Problema del productor y consumidor con Buffer limitado

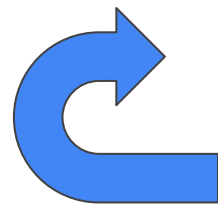
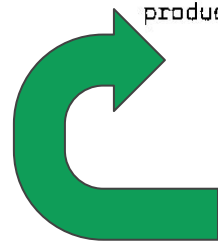
Supuestos:

- Productor: produce información de manera cíclica
- Consumidor: consume información también cíclica
- Buffer: capacidad limitada igual a N



Semáforos: Solución al Problema de productor y consumidor con buffer limitado

- Consecuencias
 - El consumidor y el productor están sincronizados por el buffer



```
"begin integer number of queuing portions, number of empty positions,  
    buffer manipulation;  
    number of queuing portions:= 0;  
    number of empty positions:= N;  
    buffer manipulation:= 1;  
    parbegin  
        producer: begin  
            again 1: produce next portion;  
                P(number of empty positions);  
                P(buffer manipulation);  
                add portion to buffer;  
                V(buffer manipulation);  
                V(number of queuing portions); goto again 1 end;  
        consumer: begin  
            again 2: P(number of queuing portions);  
                P(buffer manipulation);  
                take portion from buffer;  
                V(buffer manipulation);  
                V(number of empty positions);  
                process portion taken; goto again 2 end  
    parend  
end" .
```

Si no hay lugar en el buffer, el productor espera a que se haga un lugar

Semáforos en distintos lenguajes

Semáforos en C

```
#include <stdio.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>

sem_t mutex;

void* thread(void* arg)
{
    //wait
    sem_wait(&mutex);
    printf("\nEntered..\n");

    //critical section
    sleep(4);

    //signal
    printf("\nJust Exiting...\n");
    sem_post(&mutex);
}

int main()
{
    sem_init(&mutex, 0, 1);
    pthread_t t1,t2;
    pthread_create(&t1,NULL,thread,NULL);
    sleep(2);
    pthread_create(&t2,NULL,thread,NULL);
    pthread_join(t1,NULL);
    pthread_join(t2,NULL);
    sem_destroy(&mutex);
    return 0;
}
```

Semáforos en Rust

```
use std_semaphore::Semaphore;

// Create a semaphore that represents 5
resources
let sem = Semaphore::new(5);

// Acquire one of the resources
sem.acquire();

// Acquire one of the resources for a
limited period of time
{
    let _guard = sem.access();
    // ...
} // resources is released here

// Release our initially acquired resource
sem.release();
```

Semáforos en Java

Hay varios tipos de semáforos:

- Semaphore
- BinarySemaphore
- Counting Semaphores
- Bounded Semaphores
- Timed Semaphores

```
Semaphore sem= new Semaphore(1);
```

```
...
```

```
sem.acquire();
```

```
try
```

```
{
```

```
//critical section
```

```
}
```

```
finally
```

```
{
```

```
sem.release();
```

```
}
```


¿PREGUNTAS?