

# Semàfors

Sistemes Operatius 2

Grau d'Enginyeria Informàtica

## Característiques

- Poden ser implementats mitjançant espera activa o passiva.
- Faciliten el disseny de protocols de sincronització entre fils (productors-consumidors, lectors-escripors)
- Habitualment són utilitzats internament per un sistema operatiu per sincronitzar l'accés a recursos compartits (quan diferents processos fan crides a sistema).

## Característiques

- Desenvolupats per Dijkstra a l'any 1965.
- El concepte de semàfor està motivat per la forma en què els trens se sincronitzen entre sí per evitar col·lisions. El semàfor és un senyal que indica si la via està lliure o ocupada. A mesura que els trens circulen per la via s'activen i es desactiven els semàfors a llarg de la via.

A l'hora de programar en C, un semàfor es declara com

```
sem_t s;
```

- El semàfor es pot interpretar com una variable sencera no negativa.
- **Només hi ha dues funcions per operar amb el semàfor:** `sem_wait` i `sem_post`. No hi ha altres funcions per manipular semàfors llevat de la inicialització i destrucció de semàfors.
- Dijkstra, originalment, va donar a aquestes funcions els noms holandesos **P** (passeren = passar) i **V** (vrigeren = deixar anar).

La “definició” de `sem_wait` i `sem_post` amb espera activa és:

`sem_wait(s):`

```
while (true) {  
    <if (s > 0)  
        s = s - 1;  
    return;>  
}
```

`sem_post(s):`

```
<s = s+1;>  
return;
```

- L'operació `sem_post` incrementa atòmicament el valor d'`s`.
- L'operació `sem_wait` decrementa el valor d'`s` només si aquest és estrictament positiu. Si `s` és zero el fil que fa la crida a `sem_wait` s'espera que `s` tingui un valor positiu. L'espera pot ser activa o passiva.
- Atenció! La implementació real dels semàfors no es realitza necessàriament amb espera activa, tal com es mostra a sobre, sinó que generalment s'adormen els processos que esperen.

# Tipus de semàfors

Depenent del valor d's, els semàfors poden ser

- **Binari** si el valor d's només pot tenir valor 0 o 1.
- **Generals** si s pot tenir qualsevol valor sencer. S'utilitza per limitar l'accés al determinats recursos (ara ho veurem).

Els exemples que es mostren a continuació són exemples que utilitzen les funcions disponibles en C. Anem a veure com implementar:

- L'exclusió mútua
- El paradigma dels productors i consumidors
- El paradigma dels lectors i escriptors

# Funcions disponibles en C

Els semàfors poden ser utilitzats per **sincronitzar processos o fils**.

- Per **inicialitzar** un semàfor
  - `sem_open(...)`: inicialització per processos
  - `sem_init(...)`: inicialització per fils
- Per **sincronitzar** processos o fils
  - `sem_wait(...)`: utilitzat normalment per notificar que es vol entrar a la secció crítica (veurem que es pot utilitzar amb altres usos).
  - `sem_post(...)`: utilitzat normalment per notificar que se surt de la secció crítica.
- Per **alliberar** els recursos associats a un semàfor
  - `sem_close(...)`: alliberament per a processos
  - `sem_destroy(...)`: alliberament per a fils

# Exclusió mútua amb semàfors (algorisme 1)

Estructura del codi per a l'exclusió mútua.

variables globals:

```
sem_t s;
```

funcio\_fil:

codi independent per a cada fil

```
sem_wait(&s);
```

secció crítica

```
sem_post(&s);
```

codi independent per a cada fil

inicialització semàfor:

```
sem_init(&s, 0, 1); // inicialitzacio s = 1
```

Proveu el codi semafor\_exlusio\_mutua.c.



# Exclusió mútua amb semàfors (algorisme 1)

Només dues funcions per manipular un semàfor: `sem_wait` i `sem_post`.

- `sem_wait(&s)`: els fils que no poden entrar a la secció crítica s'esperen (activament o passivament) a una cua associada al semàfor `s`.
- `sem_post(&s)`: en sortir de la secció crítica despertem els processos o fils que estan esperant a la cua del semàfor `s`. Aquests competiran per entrar a la secció crítica.

Observar que

- L'algorisme funciona per qualsevol nombre de fils.
- El semàfor utilitzat és un semàfor binari. Mitjançant el semàfor binari assegurem que només un fil pot entrar a la secció crítica.

Com podem aconseguir que múltiples fils entrin a la secció crítica ?

# Exclusió mútua amb semàfors (algorisme 1)

variables globals:

```
sem_t s;
```

funcio\_fil:

codi independent per a cada fil

```
sem_wait(&s);
```

secció crítica

```
sem_post(&s);
```

codi independent per a cada fil

inicialització semàfor:

```
sem_init(&s, 0, M); // inicialitzacio s = M
```

# Exclusió mútua amb semàfors (algorisme 1)

Observar que

- S'utilitza un semàfor general inicialitzat a  $M$
- Mitjançant aquesta solució es pot limitar el nombre de fils que hi pot haver a la secció crítica. En particular, en aquest codi hi pot haver a tot estirar  $M$  fils a la secció crítica. Així s'aconsegueix limitar l'accés als recursos.

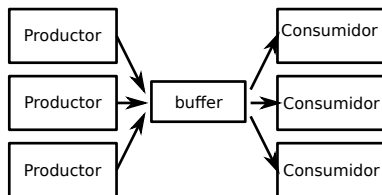
A continuació es presentarà l'algorisme de **producers i consumidors**. Per això es presentaran algorismes en què s'anirà incrementant la complexitat fins cobrir el cas més general. Veurem

- 1 Un algorisme amb múltiples productors i consumidors amb un buffer de mida 1.
- 2 Un algorisme amb un productor i un consumidor amb un buffer de mida  $M$ .
- 3 Un algorisme amb múltiples productors i consumidors amb un buffer de mida  $M$ .

Recordar que el “buffer” és el canal a través del qual el productor dóna (o entrega) les dades al consumidor.

# Múltiples productors i consumidors amb un buffer de mida 1 (algorisme 2)

Es proposa a continuació una solució per múltiples productors i consumidors amb un buffer de mida 1



- Un sol productor pot copiar una dada al buffer. Si el buffer es ple, els productors s'han d'esperar que un consumidor agafi la dada.
- Un sol consumidor pot agafar la dada del buffer. Si el buffer es buit, els consumidors s'han d'esperar que un productor hi dipositi una dada.

# Múltiples productors i consumidors amb un buffer de mida 1 (algorisme 2)

variables globals:

```
typeT buffer;  
sem_t buit (=1), ocupat (=0);
```

productors:

```
typeT data;  
while (true) {  
    produeix "data"  
    sem_wait(&buit);  
    copia "data" a "buffer"  
    sem_post(&ocupat);  
}
```

consumidors:

```
typeT data;  
while (true) {  
    sem_wait(&ocupat);  
    copia "buffer" a "data"  
    sem_post(&buit);  
    processa "data"  
}
```

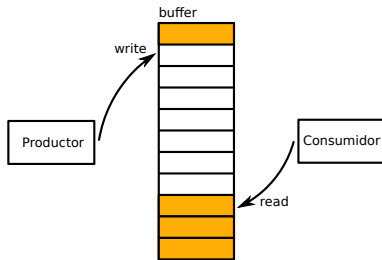
# Múltiples productors i consumidors amb un buffer de mida 1 (algorisme 2)

- El semàfor binari buit s'utilitza per indicar si el *buffer* està buit. El semàfor binari ocupat s'utilitza per indicar si el *buffer* és ple.
- Els productors
  - 1 Comproven si el *buffer* és buit.
  - 2 El productor que entra a la secció crítica copia la dada al *buffer*.
  - 3 Avisa als consumidors que el *buffer* es ple.
- Els consumidors
  - 1 Comproven si el *buffer* es ple.
  - 2 El consumidor que entra a la secció crítica copia el *buffer* a una variable local.
  - 3 Avisa als productors que el *buffer* es buit.

# Un productor i un consumidor amb un buffer de mida M (algorisme 3)

Anem a ampliar l'algorisme anterior per un *buffer* de mida M amb **un productor i un consumidor**.

- El productor només pot escriure dades al *buffer* si aquest no és ple.
- El consumidor ha de comprovar que hi ha una dada al *buffer* abans d'agafar-la.
- Per implementar aquest algorisme s'utilitzen semàfors generals.





# Un productor i un consumidor amb un buffer de mida M (algorisme 3)

variables globals:

```
typeT buffer[M];  int w = 0, r = 0;
sem_t buit (=M), ocupat (=0);
```

productor:

```
typeT data;
while (true) {
    produeix "data"
    sem_wait(&buit);
    copia "data" a "buffer[w]"
    w = (w + 1) % M;
    sem_post(&ocupat);
}
```

consumidor:

```
typeT data;
while (true) {
    sem_wait(&ocupat);
    copia "buffer[r]" a "data"
    r = (r + 1) % M;
    sem_post(&buit);
    processa "data"
}
```

Proveu el codi `semafor_1prod1cons.c` i "juguem" amb els paràmetres i el codi...

# Un productor i un consumidor amb un buffer de mida M (algorisme 3)

Observar que

- S'utilitzen semàfors generals que indiquen la mida del *buffer* utilitzat.

El productor

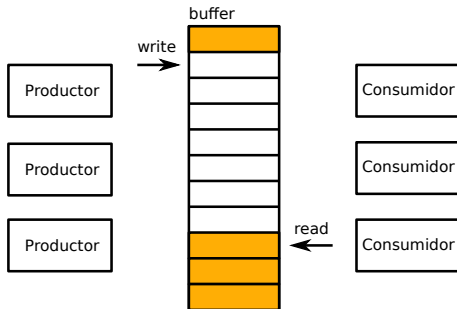
- No pot entrar a la secció crítica si el *buffer* es ple.
- En copiar les dades al *buffer* notifica al consumidor la presència d'una nova dada.

El consumidor

- No pot entrar a la secció crítica si no hi ha dades al *buffer*.
- En copiar del *buffer* notifica al productor que ha agafat una dada.

# Múltiples consumidors i productors amb un buffer de mida M (algorisme 4)

Ampliem l'algorisme anterior per múltiples consumidors i productors



- Cal assegurar que només un productor pugui copiar la dada al  $\text{buffer}[w]$ .
- Cal assegurar que només un consumidor pugui agafar la dada de  $\text{buffer}[r]$ .

# Múltiples consumidors i productors amb un buffer de mida M (algorisme 4)

variables globals:

```
typeT buffer[M]; int w = 0, r = 0;  
sem_t buit (=M), ocupat (=0);  
sem_t clauProd (=1), clauCons (=1);
```

productor:

```
typeT data;  
while (true) {  
    produeix "data"  
    sem_wait(&buit);  
    sem_wait(&clauProd);  
    copia "data" a "buffer[w]"  
    w = (w + 1) % M;  
    sem_post(&clauProd);  
    sem_post(&ocupat);  
}
```

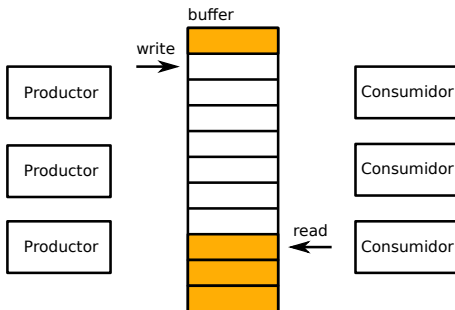
consumidor:

```
typeT data;  
while (true) {  
    sem_wait(&ocupat);  
    sem_wait(&clauCons);  
    copia "buffer[r]" a "data"  
    r = (r + 1) % M;  
    sem_post(&clauCons);  
    sem_post(&buit);  
    processa "data"  
}
```

# Múltiples consumidors i productors amb un buffer de mida M (algorisme 4)

Observar que

- S'utilitzen dos nous semàfors binaris, `clauProd` i `clauCons`, utilitzats per assegurar que un sol productor o un sol consumidor accedeixen a la secció crítica. Els semàfors `clauProd` i `clauCons` asseguren l'exclusió mútua.



# Múltiples lectors i escriptors: primera proposta (algorisme 5)

Es proposa a continuació una solució per múltiples lectors i escriptors.

Recordem les regles per evitar interferències entre fils

- Un escriptor necessita accés exclusiu a la base de dades (o el registre) en escriure-hi.
- Si no hi ha cap escriptor a la base de dades (o el registre), múltiples lectors poden accedir a la vegada per realitzar transaccions.

# Múltiples lectors i escriptors: primera proposta (algorisme 5)

variables globals:

```
sem_t rw (=1);
```

lectors:

```
while (true) {  
    sem_wait(&rw);  
    llegeix dades  
    sem_post(&rw);  
    processa dades  
}
```

escriptors:

```
while (true) {  
    processa dades  
    sem_wait(&rw);  
    escriu dades  
    sem_post(&rw);  
}
```

# Múltiples lectors i escriptors: primera proposta (algorisme 5)

Observar que

- Només hi ha un escriptor modificant la dada.
- Però aquest algorisme no aconsegueix que hi pugui haver múltiples lectors accedint a una dada.

Es proposa una modificació de l'anterior algorisme per permetre que múltiples lectors puguin accedir a la dada

- Només el primer lector agafarà el semàfor. La resta de lectors no cal que l'agafin.



# Múltiples lectors i escriptors: segona proposta (algorisme 6)

variables globals:

```
int nr = 0;  
sem_t rw (=1), clauLec (=1);
```

lectors:

```
while (true) {  
    sem_wait(&clauLec);  
    nr = nr + 1;  
    if (nr == 1) sem_wait(&rw);  
    sem_post(&clauLec);  
    llegeix les dades  
    sem_wait(&clauLec);  
    nr = nr - 1;  
    if (nr == 0) sem_post(&rw);  
    sem_post(&clauLec);  
    processa dades  
}
```

escriptors:

```
while (true) {  
    processa dades  
    sem_wait(&rw);  
    escriu dades  
    sem_post(&rw);  
}
```

## Múltiples lectors i escriptors: segona proposta (algorisme 6)

Observar que

- El codi dels escriptors no canvia respecte la primera solució.
- Els lectors utilitzen un protocol d'entrada i un de sortida per llegir dades. Aquest protocol només el pot executar un lector cada cop. El primer lector que entra al protocol executa `sem_wait(&rw)`. El darrer lector que surt del protocol executa `sem_post(&rw)`.

Quin defecte té aquesta solució ?

- L'algorisme proposat dóna preferència als lectors sobre els escriptors. Un cop hi ha un lector llegint dades, poden entrar tants lectors com vulguin mentre els escriptors s'esperen.

## Múltiples lectors i escriptors: segona proposta (algorisme 6)

- No es proposarà una solució justa pels lectors-escriptors fent servir semàfors: és molt complexa!
- Es farà servir un altre mecanisme que permet implementar-ho de forma molt més senzilla: **els monitors**. Ara els veurem!

# Conclusions

Hi ha només dues funcions per manipular un semàfor: `sem_wait` i `sem_post`.

- `sem_wait(s)`: els fils que no poden entrar a la secció crítica s'esperen (activament o passivament) a una cua associada al semàfor `s`.
- `sem_post(s)`: en sortir de la secció crítica despertem als processos o fils que estan esperant a la cua del semàfor `s`.

No hi ha una separació clara entre

- El protocol per entrar o sortir d'una secció crítica
- El procediment per adormir-se o despertar-se

Als **monitors** se separen aquestes dues tasques en funcions diferents permetent doncs **més graus de llibertat**.