

Fonaments dels Sistemes Operatius (FSO)

Departament d'Informàtica de Sistemes i Computadores (DISCA)

Universitat Politècnica de València

Bloc Temàtic 2: Gestió de Processos

Seminari Unitat Temàtica 6

Sincronització: Semàfors Posix

fso

DISCA



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

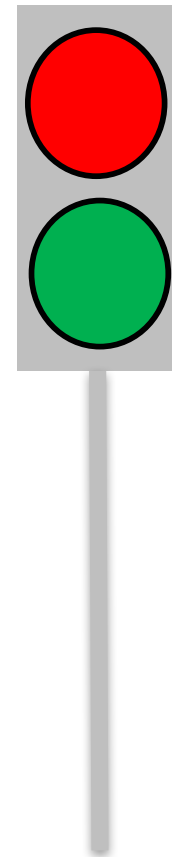
- **Objectius:**

- Familiaritzar-se amb el concepte de **secció crítica**
- Conèixer els mecanismes de **Sincronització** que ofereix el **Sistema Operatiu**
- Identificar la Sincronització d'activitats com un mecanisme bàsic dels Sistemes Operatius

- **Bibliografia**

- “Fundamentos de Sistemas operativos” Silberschatz 7ª Ed
- “Sistemas operativos: una visión aplicada” Carretero 2º Ed
- “UNIX Programación Práctica”, Kai A. Robbins, Steven Robbins. Prentice Hall. ISBN 968-880-959-4

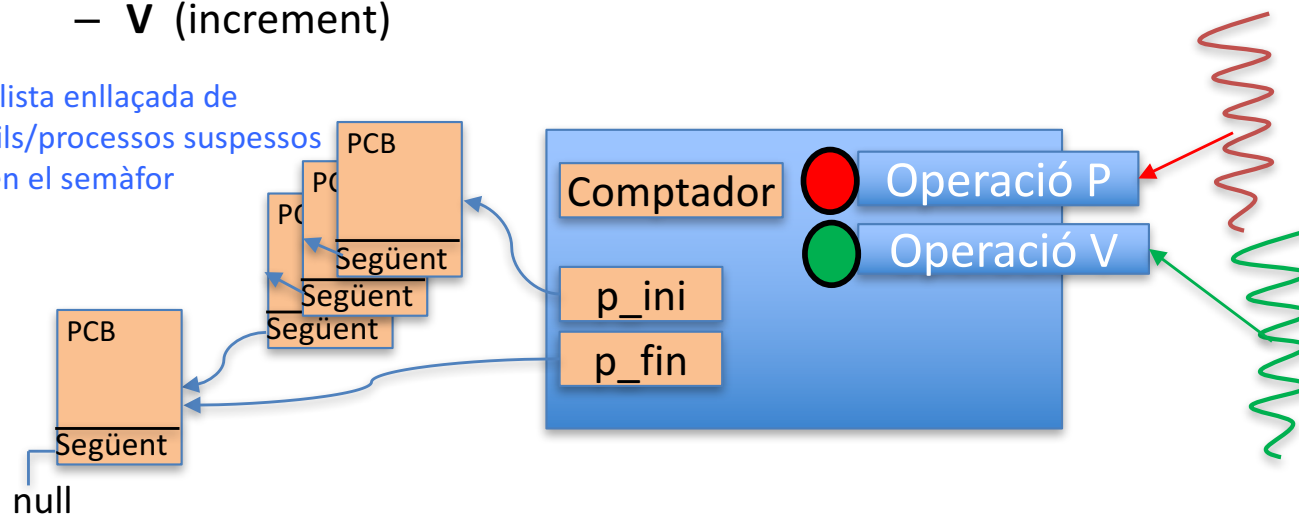
- **Contingut**
 - **Solució a nivell de Sistema Operatiu**
 - Semàfors POSIX
 - Mutex Posix
 - Exercicis



- **Semàfor**

- Un semàfor es pot entendre conceptualment com un valor enter que admet operacions d'increment i decrement
 - El decrement **pot** suspendre el procés que l'invoca
 - L'increment **pot** despertar a un altre procés prèviament suspès
- Es un tipus de dada que el Sistema Operatiu posa a disposició dels processos d'usuari
 - Es declara com una variable de tipus “semàfor”, indicant el seu valor enter inicial
 - Poseeix dues operacions d'accés, implementades com a crides al sistema
 - **P** (decrement)
 - **V** (increment)

Llista enllaçada de
fils/processos suspesos
en el semàfor



- **Semàfor: Operacions d'accés**

- Declaració i inicialització

S : semafor(N);

- Es declara el semàfor “S” amb un valor inicial “N”
- “N” ha de ser major o igual que zero (no pot ser negatiu)

- Decrement

P(S);

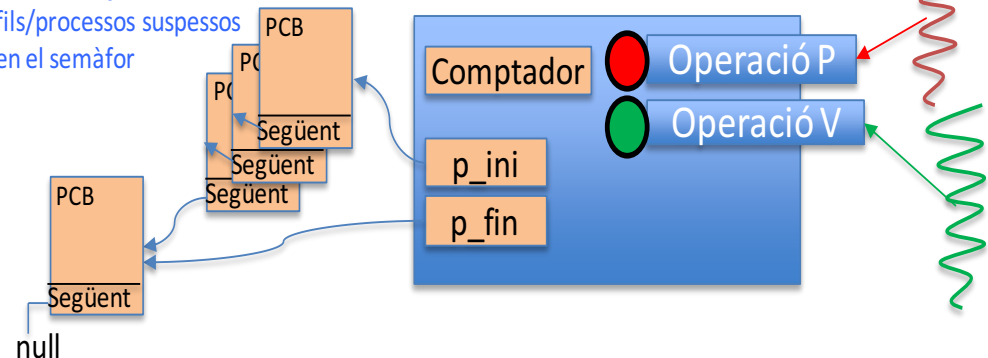
S = S - 1;
si S < 0 aleshores suspendre(S);

- Increment

V(S);

S = S + 1;
si S <= 0 aleshores despertar(S);

Llista enllaçada de
fils/processos suspesos
en el semàfor



Notes:

- El S.O. Asegura que les operacions **P** i **V** són **atòmiques**.
- **suspendre(S)**: suspén al procés invocant en una cua associada a “S”
- **despertar(S)**: tria un procés de la cua de “S” i el desperta (passa a preparats)

- **Semàfors: Solució al problema de la secció crítica**
 - Definim un semàfor compartit, denominat “mutex”, inicialitzat a 1
Semaphore mutex(1);
 - Codi dels N processos/fils:

```
void *fil_i(void *p) {  
  
    while(1) {  
        P(mutex);  
  
        /* Secció crítica */  
  
        V(mutex);  
  
        /* Secció restant */  
  
    }  
}
```

Aquesta solució
acompleix:

- Exclusió mútua
- Progrés
- Espera limitada si la cua del semàfor és FIFO

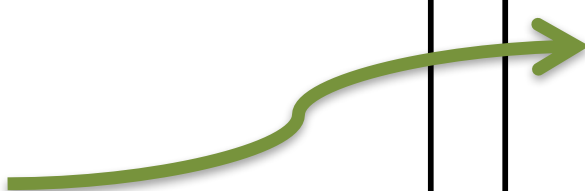
- **Semàfors:** com ferramenta per a **sincronitzar** processos en general
- Per exemple:
 - **Establir** un cert **ordre o precedència** en l'execució de zones de codi de diferents processos/fils
 - “fil1” ha d'executar la funció “F1” abans que “fil2” execute “F2”
 - Es defineix un semàfor compartit denominat “sinc” i inicialitzat a zero

Semaphore sinc(0);

```
void *fil1(void *p)
{
    ...

    F1;
    V(sinc);
    ...
}
```

```
void *fil2(void *p)
{
    ...
    P(sinc);
    F2;
    ...
}
```



- **Semàfor:** com ferramenta per a controlar el nombre de processos, o de vegades, que se pot accedir a un codi
 - Els semàfors són útils per a establir que un **màxim nombre** de processos/fils poden passar per un cert punt del codi
 - Siga un codi que executen concurrentment molts fils
 - Hi ha suficients recursos perquè 5 fils puguin accedir a un determinat lloc del codi, on s'invoca a la funció "F"
 - Definim un semàfor compartit "max_5", inicialitzat a **cinc**

```
Semaphore max_5(5);
```

```
.....
```

```
pthread_t th1, th2, th3, th4, th5;
```

```
pthread_attr_t attr;
```

```
pthread_attr_init(&attr);
```

```
pthread_create(&th1, &attr, fil_i NULL);
```

```
pthread_create(&th2, &attr, fil_i NULL);
```

```
pthread_create(&th3, &attr, fil_i NULL);
```

```
pthread_create(&th4, &attr, fil_i NULL);
```

```
.....
```

```
void *fil_i(void *p) {  
  
    ...  
    P(max_5);  
    F;  
    V(max_5);  
    ...  
}
```


- **Semàfor : útil com “comptador de recursos”**

- Quan un fil/procés **necessita un recurs**, fa l'**operació $P(S)$** , és a dir, decrementa el comptador de S i si no hi ha suficients recursos disponibles, el fil es suspén
- Quan un fil/procés **termina d'utilitzar un recurs** fa l'**operació $V(S)$** és a dir, incrementa el comptador de S . Si hi haguera algun fil suspés a l'espera de l'alliberament del recurs, l'operació **V** el desperta.
- Si el comptador del semàfor és positiu indica la quantitat de recursos disponibles

si $S > 0$ aleshores $S = \text{cantidad de recursos disponibles}$

- Si el comptador del semàfor és negatiu, el seu valor absolut indica la quantitat de processos suspesos en la cua del semàfor.

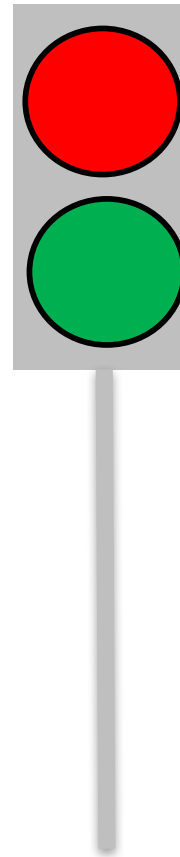
si $S < 0$ aleshores $|S| = \text{nombre de processos suspesos}$

- Quan el comptador del semàfor és zero, ambdues afirmaciones anteriores són aplicables

**si $S = 0$ aleshores \rightarrow no hi ha processos suspesos
no hi ha recursos disponibles**

- El semàfor és un mecanisme essencial del S.O. que permet:
 - implementar cues de processos suspesos a l'espera de poder usar recursos
 - Sincronització d'activitats

- Soluciones a nivell de Sistema Operatiu
- **Semàfors POSIX**
- Mutex Posix
- Exercicis



- POSIX.1b va introduir el tipus de variable semàfor `sem_t`


```
#include <semaphore.h>
sem_t sem;
```

- Els **semàfors es poden compartir** entre processos i poden ser accedits per part de tots els fils del procés.
- Els **semàfors s'hereten de pares a fills** igual que els descriptors de fitxer.


- Les operacions que suporta són:

```
-int sem_init(sem_t *sem, int pshared, unsigned int value);
-int sem_destroy(sem_t *sem);
-int sem_wait(sem_t *sem);
-int sem_trywait(sem_t *sem);
-int sem_post(sem_t *sem);
-int sem_getvalue(sem_t *sem, int *sval);
```

Operació
P(sem)



Operació
V(sem)



- **Productor Consumidor: Versió 1.0**

- Definim un semàfor “mutex” inicialitzat a u → **sem_init(&mutex,0,1);**

```
void *func_prod(void *p) {
    int item;

    while(1) {
        item = produir();

        sem_wait(& mutex);

        while (comptador == N)
            /*bucle buit*/ ;
        buffer[entrada] = item;
        entrada = (entrada + 1) % N;
        comptador = comptador + 1;

        sem_post(&mutex);
    }
}
```

```
void *func_cons(void *p) {
    int item;

    while(1) {
        sem_wait(&mutex);

        while (comptador == 0)
            /*bucle buit*/ ;
        item = buffer[sortida];
        sortida = (sortida + 1) % N;
        comptador = comptador - 1;

        sem_post(&mutex);

        consumir(item);
    }
}
```

- Aquesta solució **no funciona**, perquè un productor o consumidor que haja tancat “mutex” i es quede en el bucle “while”, deixa tota la resta de fils bloquejats!!

- Productor Consumidor: Versió 2.0

```
#include <semaphore.h>
sem_t mutex, items, buits;
```

```
void *func_prod(void *p) {
    int item;
    while(1) {
        item = producir();
        sem_wait(&buits);
        sem_wait(&mutex);

        buffer[entrada] = item;
        entrada = (entrada + 1) % N;
        comptador = comptador + 1;

        sem_post(&mutex);
        sem_post(&items);
    }
}
```

```
void *func_cons(void *p) {
    int item;
    while(1) {
        sem_wait(&items);
        sem_wait(&mutex);

        item = buffer[sortida];
        sortida = (sortida + 1) % N;
        comptador = comptador - 1;

        sem_post(&mutex);
        sem_post(&buits);
        consumir(item);
    }
}
```

```
sem_init(&mutex,0,1);
sem_init(&buits,0,N); //indica el nombre de buits
sem_init(&items,0,0); //indica el nombre de ítems
...
```

- Soluciones del Sistema Operatiu. Semàfors.
- Semàfors POSIX
- **Mutex Posix**
- Exercicis



- mecanisme de **sincronització entre fils (threads)**
 - POSIX.1c defineix els objectes “mutex” per a la sincronització de fils
 - Són com semàfors que només poden pendre **valor inicial 1**
 - S'utilitzen només per a garantir **l'exclusió mútua**.
 - Funcionen com un pany. Dues operacions bàsiques: **tancament i apertura**
 - Cada mutex té en cada instant:
 - **estat** : Dos possibles estats interns, obert i tancat
 - **propietari**: Un fil és el propietari del mutex si ha executat sobre ell una operació de tancament amb èxit

- **Funcionament del mutex**

- Un **mutex es crea** inicialment **obert i sense propietari**
- Quan un fil invoca l'operació de tancament sobre un mutex
 - Si el mutex estava obert (sense propietari), el fil el tanca i passa a ser-ne el propietari
 - Si el mutex ja estava tancat, el fil invocant se suspén
- Quan el fil propietari del mutex invoca la operació d'apertura
 - S'obri el mutex
 - Si hi havia fils suspesos, s'en selecciona un i es desperta. El fil despertat pot tancar de nou el mutex (i passa a ser el seu nou propietari)



• Funcionament del mutex

Un mutex es crea inicialment obert i sense propietari



Quan un fil invoca l'operació de tancament sobre un mutex i....



mutex estava obert?

NO



SI



el fil el **tanca** i passa a ser-ne el **propietari**



el fil se **suspés** en la **cua** de mutex



Quan el fil propietari del mutex invoca la operació d'apertura i...



hi havia fils suspesos?

NO

SI



s'en selecciona un i es desperta. El fil despertat tanca el mutex i passa a ser **propietari**



S'obri el mutex



- **Crides POSIX per a gestió de mutex:**

- **Creació i destrucció de mutex:**

- pthread_mutex_init
- pthread_mutex_destroy

- **Inicialització dels atributs del mutex**

- pthread_mutexattr_init
- pthread_mutexattr_destroy
- Modificació/Consulta de valors a l'atribut: compartició, protocol, etc.

- **Tancament i apertura de mutex**

- pthread_mutex_lock
- pthread_mutex_trylock
- pthread_mutex_unlock

Operació
P (sem)

Operació
V (sem)

- **Exemple:** accés concurrent a una variable per part de dos fils
 - Codi del programa principal:

```
pthread_mutex_t m = PTHREAD_MUTEX_INITIALIZER;

int V = 100;

// Codi dels fils (transparència següent)

int main ( ) {
    pthread_t      fil1, fil2;
    pthread_attr_t atributs;
    pthread_attr_init(&atributs);
    pthread_create(&fil1, &atributs, f_fil1, NULL);
    pthread_create(&fil2, &atributs, f_fil2, NULL);

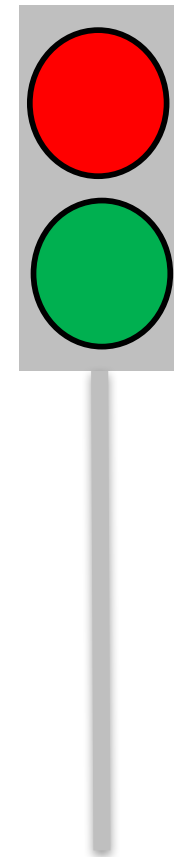
    pthread_join(fil1, NULL);
    pthread_join(fil2, NULL);
}
```

- Exemple
 - Codi dels fils:

```
void *f_fil1(void *p) {  
    int c;  
  
    for(c=0; c<1000; c++) {  
        pthread_mutex_lock(&m);  
  
        V = V + 1;  
  
        pthread_mutex_unlock(&m);  
    }  
    pthread_exit(0);  
}
```

```
void *f_fil2(void *p) {  
    int c;  
  
    for(c=0; c<1000; c++) {  
        pthread_mutex_lock(&m);  
  
        V = V - 1;  
  
        pthread_mutex_unlock(&m);  
    }  
    pthread_exit(0);  
}
```

- Soluciones del Sistema Operatiu. Semàfors.
- Semàfors POSIX
- Mutex Posix
- **Exercicis**



- **Exercici S06.1:**

¿Quins són els possibles valors que pendrà x com a resultat de l'execució concurrent dels següents fils?

```
#include <semaphore.h>
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
sem_t s1,s2,s3;
int x;
```



```
void *func_hilo1(void *a)
{
    sem_wait(&s1);
    sem_wait(&s2);
    x=x+1;
    sem_post(&s3);
    sem_post(&s1);
    sem_post(&s2);
}
void *func_hilo2(void *b)
{
    sem_wait(&s2);
    sem_wait(&s1);
    sem_wait(&s3);
    x=10*x;
    sem_post(&s2);
    sem_post(&s1);
}
```

```
int main()
{
    pthread_t h1,h2 ;
    x = 1;
    sem_init(&s1,0,1); /*Inicializa a 1*/
    sem_init(&s2,0,1); /*Inicializa a 1*/
    sem_init(&s3,0,0); /*Inicializa a 0*/

    pthread_create(&h1,NULL,func_hilo1,NULL);
    pthread_create(&h2,NULL,func_hilo2,NULL);
    pthread_join(h1,NULL);
    pthread_join(h2,NULL);
}
```

Exercici S06.2:

Determineu quins valors possibles tindrien les variables x i y en acabar l'execució dels següents tres processos concurrents. Els valors inicials són: $x=1$, $i=4$, $S1=1$, $S2=0$ i $S3=1$.

Procés A

```
P(S2);  
P(S3);  
x = y * 2;  
y = y + 1;  
V(S3);
```

Procés B

```
P(S1);  
P(S3);  
x = x + 1;  
y = 8 + x;  
V(S2);  
V(S3);
```

Procés C

```
P(S1);  
P(S3);  
x = y + 2;  
y = x * 4;  
V(S3);  
V(S1);
```

