# Laboratorio di Sistemi Operativi

**primavera 2009**

## System V IPC
## Semafori

## semafori

- sono differenti dalle altre forme di IPC
- sono contatori usati per controllare l'accesso a risorse condivise da processi diversi
- il protocollo per accedere alla risorsa è il seguente
  - 1) testare il semaforo
  - 2) se > 0, allora si può usare la risorsa (e viene decrementato il semaforo)
  - 3) se = 0, processo va in sleep finché il semaforo non ridiventa > 0, a quel punto wake up e goto step 1
- quando ha terminato con la risorsa, incrementa il semaforo
- semaforo a 0 significa che si sta utilizzando il 100% della risorsa



## semafori

- È importante notare che per implementare correttamente un semaforo l'operazione di verifica del valore del semaforo ed il decremento devono costituire una *operazione atomica*
- Per questa ragione i semafori sono implementati all'interno del kernel



## semafori

- forma semplice: semaforo binario

  controlla un'unica risorsa ed il suo valore è inizializzato a 1
  - Es. stampante capace di gestire 1 sola stampa alla volta
- forma più complessa:

  inizializzato ad un valore positivo indicante il numero di risorse che sono a disposizione per essere condivise
  - Es. stampante capace di gestire 10 stampe alla volta

# semafori

- forma nel System V:

  insieme di semafori...bisogna specificarne il numero

  - Es. 5 stampanti ciascuna capace di gestire 10 stampe alla volta



# Strutture associate ai semafori

```
struct semid_ds {
  struct ipc_perm  sem_perm;  /* see Section 14.6.2 */
  struct sem  *sem_base;  /* ptr to first semaphore in set */
  ushort       sem_nsems; /* # of semaphores in set */
  time_t       sem_otime; /* last-semop() time */
  time_t       sem_ctime; /* last-change time */
};
```

sem_base punta ad un array di sem_nsems elementi, dove ciascuno di tali elementi è una struttura sem

```
struct sem {
  ushort  semval;   /* semaphore value, always >= 0 */
  pid_t   sempid;   /* pid for last operation */
  ushort  semncnt;  /* # processes awaiting semval > currval */
  ushort  semzcnt;  /* # processes awaiting semval = 0 */
};
```



# Limiti di sistema

| Name | Description | Typical Value |
|---|---|---|
| SEMVMX | The maximum value of any semaphore. | 32,767 |
| SEMAEM | The maximum value of any semaphore's adjust-on-exit value. | 16,384 |
| SEMMNI | The maximum number of semaphore sets, systemwide. | 10 |
| SEMMNS | The maximum number of semaphores, systemwide. | 60 |
| SEMMSL | The maximum number of semaphores per semaphore set. | 25 |
| SEMMNU | The maximum number of undo structures, systemwide. | 30 |
| SEMUME | The maximum number of undo entries per undo structures. | 10 |
| SEMOPM | The maximum number of operations per semop call. | 10 |



# Semafori

Da qui in avanti utilizzeremo:

> semaforo per riferirci all'insieme

> semaforini per riferirci agli elementi dell'insieme

# Semafori

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semget(key_t key, int nsems, int flag );
```

Restituisce: ID del semaforo se OK,
-1 in caso di errore



# semget

- Quando il semaforo è creato viene inizializzata la struttura semid_ds
  - è inizializzata ipc_perm con i permessi dati nel *flag*
  - sem_otime è settato a 0
  - sem_ctime è settato all'istante di creazione
  - sem_nsems è settato a *nsems* (# semaforini nell'insieme)

Es: semid = semget (key, 1, IPC_CREAT|IPC_EXCL|0666);
crea un insieme con 1 solo semaforino

- Per riferirsi ad un semaforo esistente specificare *nsems*=0

Es: semid = semget (key, 0, 0666);



# Funzione semctl

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int semctl(int semid, int semnum, int cmd, union semun arg);
```

Serve ad eseguire operazioni di vario tipo con i semafori



# argomento *cmd*

- *cmd* rappresenta il comando da eseguire sull' insieme di valori del semaforo identificato da *semid*

IPC_STAT = recupera la struttura semid_ds per l'insieme *semid* (la conserva in *arg.buf)*

IPC_SET = setta alcuni dei campi della struttura ipc_perm in semid_ds (la prende da *arg.buf*)

SETVAL = setta il campo semval nella struttura sem al valore dell'argomento *arg.val* per il semaforino specificato in *semnum* ($0 \leq semnum <$ sem_nsems)

GETALL = recupera i valori del semaforo presenti nell'insieme (li conserva in *arg.array*)

SETALL = setta i valori del semaforo (quelli dell'insieme) a quelli contenuti in *arg.array*

IPC_RMID = rimuove il semaforo (l'insieme)

## union semun

```
union semun {
    int              val;    /* for SETVAL  */
    struct semid_ds  *buf    /* for IPC_STAT e IPC_SET */
    ushort           *array; /* per GETALL e SETALL */
}
```

- il campo *val* si usa con SETVAL per settare il valore del semaforino *semnum* specificato nell'insieme con identificatore *semid*; in realtà i suoi valori possono essere
  - *val* ≥ 1 ad indicare che la risorsa è disponibile nella quantità specificata (semaforo lockable)
  - *val* = 0 ad indicare è da ritenersi già utilizzata al 100% (semaforo unlockable)



## esempio d'uso

- semctl(semid, 0, IPC_RMID, 0);
  - rimuove il semaforo
- semctl (semid, 0, SETVAL, 1) ;
  - setta il valore del (primo) semaforino ad 1
- i =semctl (semid, 2, GETVAL, ignorato)
  - i contiene il valore del terzo semaforino
  - il quarto argomento è ignorato
  - serve a controllare il valore del semaforino



## Funzione semop

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
```

int semop(int *semid*, struct sembuf *semoparray[ ]*, size_t *nops* );

Descrizione: esegue automaticamente un'array di operazioni, date in *semoparray,* sul semaforo identificato da *semid*

Restituisce: 0 se OK,
-1 in caso di errore



## Struttura *sembuf*

```
struct sembuf {
   ushort  sem_num;  /* member # in set (0, 1, ..., nsems-1) */
   short   sem_op;   /* operation (negative, 0, or positive) */
   short   sem_flg;  /* IPC_NOWAIT, SEM_UNDO */
};
```

- sem_num : numero del semaforino
- sem_op :
  - se < 0 : richiesta. Il processo dorme (se IPC_NOWAIT non è specificato) finché non è disponibile la risorsa (in quantità pari a |*sem*_op| ) e poi viene sottratto il valore di |*sem*_op| da *semval* nella struttura sem
  - se > 0 : rilascio. Il valore viene sommato a *semval*
  - se = 0 : si vuole aspettare (se IPC_NOWAIT non è specificato) fino a che il valore del semaforo *semval* =0

# esempio d'uso (1)

```
struct sembuf sem_lock={0, -1, 0};
struct sembuf sem_unlock={0, 1, 0};

semid=semget(IPC_PRIVATE,1,IPC_CREAT | 0666);
semctl(semid,0,SETVAL, 1);   // inizializza i valori di semid

semop(semid,&sem_lock,1);
printf("sem loccato\n");

   // utilizzo della risorsa gestita dall'unico semaforino nell'insieme

semop(semid,&sem_unlock,1);
printf("sem unloccato \n");
```



# esempio d'uso (1 più)

```
struct sembuf sem_lock;
sem_lock.sem_num=0;
sem_lock.sem_op=-1;
sem_lock.sem_flg=0;

struct sembuf sem_unlock;
sem_unlock.sem_num=0;
sem_unlock.sem_op=1;
sem_unlock.sem_flg=0;

semid=semget(IPC_PRIVATE,1,IPC_CREAT | 0666);
semctl(semid,0,SETVAL, 1);   // inizializza i valori di semid

semop(semid,&sem_lock,1);
printf("sem loccato\n");

// utilizzo della risorsa gestita dall'unico semaforino nell'insieme

semop(semid,&sem_unlock,1);
printf("sem unloccato \n");
```



# esempio d'uso (2)

```
struct sembuf sem_lock_0={0, -1, 0};
struct sembuf sem_unlock_0={0, 1, 0};
struct sembuf sem_lock_1={1, -1, 0};
struct sembuf sem_unlock_1={1, 1, 0};


semid=semget(IPC_PRIVATE,2,IPC_CREAT | 0666);
semctl(semid,0,SETVAL, 2);   // inizializza i valori di semid
semctl(semid,1,SETVAL, 1);

semop(semid,&sem_lock_0,1);
printf("sem 0 loccato\n");

   // utilizzo della risorsa gestita dal semaforino  numerato 0 nell'insieme

semop(semid,&sem_unlock_0,1);
printf("sem 0  unloccato \n");
```