

# Semafori System V

Laboratorio Sistemi Operativi

Antonino Staiano  
Email: antonino.staiano@uniparthenope.it

# Semafori System V

- I semafori System V aggiungono un ulteriore livello di dettaglio ai semafori contatori Posix, definendo:
  - Un **insieme** di semafori contatori: uno o più semafori ognuno dei quali è un semaforo contatore
    - Esiste un limite sul numero di semafori contatori in un insieme (dipende dalla implementazione, es. Solaris è 25, Linux 250)
  - N.B.:
    - quando parliamo di “Semafori System V” ci riferiamo ad un insieme di semafori contatori
    - quando parliamo di “Semafori Posix” ci riferiamo ad un singolo semaforo contatore

# Semafori System V

- Il kernel associa a ciascun insieme di semafori la seguente struttura (**<sys/sem.h>**):

```
struct semid_ds {
    struct ipc_perm sem_perm; /* info su proprietari, permessi, ... */
    struct sem *sem_base;    /* ptr a primo semaforo */
    ushort sem_nsems;        /* numero semafori */
    time_t sem_otime;        /* tempo ultima semop() */
    time_t sem_ctime;        /* tempo ultima modifica */
};
```

- Il campo **sem\_base** è il puntatore alle strutture **sem**

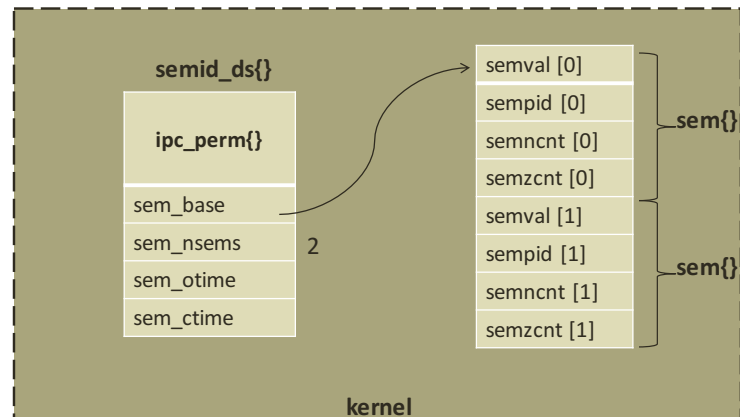
```
struct sem {
    ushort semval;          /* valore del semaforo (>= 0) */
    pid_t sempid;          /* pid processo ultima operazione */
    ushort semncnt;        /* #processi in attesa di semval > current_value */
    ushort semzcnt;        /* #processi in attesa semval=0 */
};
```

# Semafori System V

- La struttura **ipc\_perm** contiene informazioni per ogni oggetto di IPC che il kernel memorizza (**<sys/ipc.h>**)

```
struct ipc_perm {
    uid_t    uid;    /* user id proprietario */
    gid_t    gid;    /* group id proprietario */
    uid_t    cuid;   /* user id creatore */
    gid_t    cgid;   /* group id creatore */
    mode_t   mode;   /* permessi lettura-scrittura */
    ulong_t  seq;    /* numero sequenza */
    key_t    key;    /* chiave IPC */
};
```

## Strutture del kernel per un insieme di due semafori



## Semafori System V

- I semafori possono essere utilizzati sia singolarmente, sia come insiemi
- Un insieme di semafori:
  - viene creato con **semget ()**
  - le operazioni vengono eseguite con **semop()**
  - le operazioni di controllo sono eseguite con **semctl()**
- Quando si crea un semaforo, deve essere specificata una chiave **key** che deve essere nota a tutti i processi che vogliono utilizzare la struttura

## Identificatori e chiavi

- Ogni struttura di IPC (semafori e memoria condivisa) è individuata da un identificatore intero non negativo
- L'identificatore è un **nome interno** associato ad un oggetto di IPC
  - Un oggetto di IPC è associato con una chiave (**key**) che agisce come **nome esterno**
- Quando è creata una struttura di IPC bisogna specificare una chiave
  - Il tipo di dato di questa chiave è il tipo di dato di sistema primitivo **key\_t** (definito spesso come intero long in **<sys/types.h>**)
  - Questa chiave è convertita in un identificatore dal kernel

## ftok(): chiavi per IPC

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok(const char *pathname, int proj_id);
```

- le chiavi possono essere ottenute con la funzione **ftok()** e risultano univocamente associate ai due parametri:
  - un file esistente di percorso assoluto **pathname**
  - un identificativo di progetto **proj\_id**
  - restituisce - 1 in caso di errore

# Semafori: creazione e accesso

- La funzione **semget()** crea un insieme di semafori o accede ad un insieme di semafori esistente

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget ( key_t key, int nsems, int semflg )
```
- semget()** ritorna un intero, l' *identificatore del semaforo*, utilizzato dalle funzioni **semop()** e **semctl()**
- L'argomento **key** è un valore di accesso associato con il semaforo oppure **IPC\_PRIVATE** (il quale assicura la creazione di un nuovo oggetto di IPC unico)
- nsems** specifica il numero di semafori nell'insieme; se si sta accedendo ad un insieme esistente si passa 0

# Semafori: creazione e accesso

- I valori possibili per **semflg** sono una combinazione dei **permessi** con:
  - IPC\_CREAT** per creare un nuovo semaforo. Se il flag non è usato, **semget()** cerca il semaforo associato alla chiave e controlla i permessi dell'utente
  - IPC\_EXCL** viene utilizzato con **IPC\_CREAT** affinché **semget()** ritorni un errore se il semaforo esiste
- N.B.: la struttura **sem** associata ad ogni semaforo dell'insieme non è garantito sia inizializzata. Tali strutture sono inizializzate quando è invocata la funzione **semctl()** con gli argomenti **SETVAL** o **SETALL**

# Operazioni sui semafori

- ```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop (int semid, struct sembuf *sops, unsigned nsops);
/* * ritorna 0 se OK, -1 in caso di errore */
```
- semop()** esegue operazioni su di un insieme di semafori. L'argomento **semid** è l'identificativo di semaforo ritornato da **semget()**
  - L'argomento **sops** è un puntatore ad un array di strutture, ciascuna contenente:
    - il numero di semaforo,
    - l'operazione da eseguire,
    - i flag di controllo, se necessari
  - nsops** specifica la lunghezza dell'array

# Operazioni su semafori

- La struttura **sembuf** specifica un'operazione su semaforo, come definito in **<sys/sem.h>**

```
struct sembuf {
    ushort  sem_num; /* numero di semaforo: primo = 0 */
    short   sem_op;  /* operazione sul semaforo */
    short   sem_flg; /* flag: IPC_NOWAIT, SEM_UNDO */
};
```
- sem\_num** è il numero del semaforo su cui eseguire l'operazione

# Operazioni su semafori

- L'operazione da eseguire su `sembuf.sem_num` è determinata da `sembuf.sem_op` come segue:
  - un valore > 0 incrementa il semaforo di quel valore (rilascio di risorse controllate dal semaforo)
  - un valore < 0 fa sì che il chiamante intenda aspettare fino a che il valore del semaforo diventa maggiore o uguale al valore assoluto di `sem_op`. Ciò corrisponde all'allocazione di risorse
    - Se `semval` (campo della struttura `sem`) è maggiore o uguale del valore assoluto di `sembuf.sem_op`, `sembuf.sem_op` è sottratto a `semval`
    - Se `semval` è minore del valore assoluto di `sembuf.sem_op`, il thread invocante è bloccato fino a quando `semval` diviene maggiore o uguale al valore assoluto di `sembuf.sem_op`
    - Se è specificato `IPC_NOWAIT` (nel membro `sem_flg` della struttura `sembuf`) il thread invocante non è bloccato e la funzione `semop()` restituisce l'errore `EAGAIN`

[ 13 ]

# Operazioni su semafori

- un valore = 0 mette il processo in attesa fino a quando il valore del semaforo ha raggiunto il valore 0
- I semafori Posix consentono solo le operazioni -1 (`sem_wait()`) e +1 (`sem_post()`)
- I semafori System V consentono di incrementare o decrementare i valori dei semafori di valori maggiori di 1 e di attendere fino a che il valore del semaforo è 0
  - Queste operazioni più generali, ed il fatto che i System V sono insiemi di semafori, complicano i semafori rispetto alla definizione dei semafori Posix

[ 14 ]

# Operazioni su semafori

- Ci sono due flag di controllo che possono essere usati nel campo `sem_flg` della struttura `sembuf`:
  - `IPC_NOWAIT` La funzione ritorna senza alterare nessun valore del semaforo se una qualsiasi operazione non può essere eseguita
  - `SEM_UNDO` Permette di annullare le operazioni eseguite sull'array quando il processo termina

[ 15 ]

# Operazioni su semafori

- `sops` è un puntatore ad un array di strutture di operazioni su semaforo
- Ciascuna struttura nell'array contiene i dati per eseguire un'operazione su semaforo. Se un'operazione fallisce, nessuno dei semafori è alterato
- Il processo si blocca (se non è stato specificato `IPC_NOWAIT`) fino a che:
  - le operazioni sui semafori sono tutte completate
  - il processo riceve un segnale
  - l'insieme dei semafori è rimosso
- Solo un processo alla volta può aggiornare un semaforo. Richieste simultanee eseguite da processi diversi vengono eseguite in un ordine arbitrario

[ 16 ]

# Controllo dei semafori

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semctl (int semid, int semnum, int cmd, union semun
    arg);
```

- **semctl()** esegue varie operazioni di controllo sui semafori
- La funzione deve essere chiamata con un identificativo di semaforo **semid** valido (ritornato da **semget()**)
- Il valore di **semnum** permette di selezionare un semaforo in un array attraverso il suo indice (usato solo per **GETVAL**, **SETVAL**, **GETNCNT**, **GETZCNT**, **GETPID**)

( 17 )

# Controllo dei semafori

- Il quarto argomento **arg** è opzionale, e dipende dall'operazione che si compie (il terzo argomento **cmd**)
- Se richiesto, è di tipo **union semun**, che deve essere esplicitamente dichiarata nell'applicazione come:

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

( 18 )

# Controllo dei semafori

- L'argomento **cmd** è uno dei seguenti flag (se non **definito altrimenti**, un valore di ritorno della funzione pari a 0 indica successo, -1 errore):
- **GETVAL** ritorna il valore di un singolo semaforo come **valore della funzione**
- **SETVAL** assegna il valore di un singolo semaforo (preso da **arg.val**)
- **GETALL** ritorna il valore di tutti i semafori. I valori sono ritornati attraverso il puntatore **arg.array**
- **SETALL** inizializza il valore di tutti i semafori in un insieme. I valori sono specificati in **arg.array**
- **GETPID** ritorna il valore del pid del processo che ha eseguito l'ultima operazione sul semaforo come **valore della funzione**
- **GETNCNT** ritorna il **numero di processi** che attendono che il valore del semaforo incrementi
- **GETZCNT** ritorna il **numero di processi** che attendono che il valore del semaforo sia zero

( 19 )

# Controllo dei semafori

- **IPC\_STAT** ritorna le informazioni sullo stato del semaforo e le pone nella struttura puntata da **arg.buf**, un puntatore ad un buffer di tipo **semid\_ds**
- **IPC\_SET** inizializza gli user e group ID e permessi della struttura **semid\_ds** del semaforo con i valori dei corrispondenti membri della struttura puntata da **arg.buf**
- **IPC\_RMID** rimuove il set di semafori specificato da **semid**

( 20 )

# Esempio

- I semafori System V hanno persistenza di kernel
- Verifichiamone l'uso con un insieme di piccoli programmi
- I valori dei semafori saranno mantenuti dal kernel da uno dei programmi al successivo

[ 21 ]

# semcreate

- Crea un insieme di semafori System V
- L'opzione **-e** da riga di comando consente di specificare il flag **IPC\_EXCL**
- Il numero di semafori dell'insieme è definito da riga di comando

[ 22 ]

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ipc.h>
#include <sem.h>

int
main(int argc, char **argv)
{
    int c, oflag, semid, nsems;

    oflag = 0666 | IPC_CREAT;
    while ( (c = getopt(argc, argv, "e")) != -1) {
        switch (c) {
            case 'e':
                oflag |= IPC_EXCL;
                break;
        }
    }
    if (optind != argc - 2) {
        fprintf(stderr, "usage: semcreate [ -e ] <pathname> <nsems>");
        exit(-1);
    }
    nsems = atoi(argv[optind+1]);

    semid = semget(ftok(argv[optind], 1), nsems, oflag);
    exit(0);
}
```

# semcreate

[ 23 ]

# semrmid

- Rimuove un insieme di semafori dal sistema
- Viene eseguito il comando **IPC\_RMID** attraverso la funzione **semctl()** per rimuovere l'insieme

[ 24 ]

## semrmid

```
#include ...

int
main(int argc, char **argv)
{
    int        semid;

    if (argc != 2){
        fprintf(stderr, "usage: semrmid <pathname>");
        exit(-1);
    }
    semid = semget(ftok(argv[1], 1), 0, 0);
    semctl(semid, 0, IPC_RMID);

    exit(0);
}
```

[ 25 ]

## semsetvalues

- Imposta tutti i valori in un insieme di semafori
- Dopo aver ottenuto l'ID del semaforo con `semget()`, viene passato `IPC_STAT` a `semctl()` per prelevare la struttura `semid_ds` associata al semaforo
  - Il membro `sem_nsems` è il numero di semafori nell'insieme
- Si alloca memoria per un array di `ushort`, uno per ciascun membro dell'insieme, e si copiano i valori dalla linea di comando nell'array
  - Un comando `SETALL` a `semctl()` imposta tutti i valori nell'insieme dei semafori

[ 26 ]

## semsetvalues ...

```
#include ...

int main(int argc, char **argv)
{
    int        semid, nsems, i;
    struct semid_ds  seminfo;
    unsigned short *ptr;
    union semun arg;
    if (argc < 2){
        fprintf(stderr, "usage: semsetvalues <pathname> [ values ... ]");
        exit(-1);
    }
    /* prima prende il numero di semafori nell'insieme */
    semid = semget(ftok(argv[1], 0), 0, 0);
    arg.buf = &seminfo;
    semctl(semid, 0, IPC_STAT, arg);
    nsems = arg.buf->sem_nsems;
```

[ 27 ]

## ... semsetvalues

```
/* poi prende i valori da linea di comando */
if (argc != nsems + 2){
    fprintf(stderr, "%d semaphores in set, %d values specified", nsems, argc-2);
    exit(-1);
}
/* alloca memoria per mantenere tutti i valori nell'insieme, e memorizza */
ptr = calloc(nsems, sizeof(unsigned short));
arg.array = ptr;
for (i = 0; i < nsems; i++){
    ptr[i] = atoi(argv[i + 2]);
}
semctl(semid, 0, SETALL, arg);

exit(0);
}
```

[ 28 ]

# semgetvalues

- Preleva e stampa tutti i valori in un insieme di semafori
- Dopo aver ottenuto l'ID dell'insieme con `semget()`, si invia il comando `IPC_STAT` a `semctl()` per prelevare la struttura `semid_ds`. Il membro `sem_nsems` è il numero di semafori nell'insieme
- Si alloca un array di `ushort`, uno per ciascun elemento dell'insieme, e si invia un comando `GETALL` a `semctl()` per prelevare tutti i valori nell'insieme dei semafori
  - Ciascun valore viene stampato

[ 29 ]

Cdl in Informatica - Laboratori di ISO - AA 2015/2016 - Prof. Antonino Staiano

# semgetvalues ...

```
#include ...
int
main(int argc, char **argv)
{
    int      semid, nsems, i;
    struct semid_ds  seminfo;
    unsigned short   *ptr;
    union semun arg;

    if (argc != 2){
        fprintf(stderr, "usage: semgetvalues <pathname>");
        exit(-1);
    }

    /* prende prima il numero di semafori nell'insieme */
    semid = semget(ftok(argv[1], 0), 0, 0);
    arg.buf = &seminfo;
    semctl(semid, 0, IPC_STAT, arg);
    nsems = arg.buf->sem_nsems;
```

[ 30 ]

Cdl in Informatica - Laboratori di ISO - AA 2015/2016 - Prof. Antonino Staiano

# ... semgetvalues

```
/* alloca memoria per tenere tutti i valori
dell'insieme */
ptr = calloc(nsems, sizeof(unsigned short));
arg.array = ptr;

/* preleva i valori e stampa */
semctl(semid, 0, GETALL, arg);
for (i = 0; i < nsems; i++)
    printf("semval[%d] = %d\n", i, ptr[i]);

exit(0);
}
```

[ 31 ]

Cdl in Informatica - Laboratori di ISO - AA 2015/2016 - Prof. Antonino Staiano

# semops

- Esegue un array di operazioni su un insieme di semafori
- L'opzione `-n` specifica il flag `IPC_NOWAIT` per ogni operazione ed un'opzione `-u` specifica il flag `SEM_UNDO` per ogni operazione
- Dopo aver aperto l'insieme dei semafori con `semget()`, viene allocato un array di strutture `sembuf`, un elemento per ogni operazione specificata da riga di comando
- `semop()` esegue l'array di operazioni sull'insieme dei semafori

[ 32 ]

Cdl in Informatica - Laboratori di ISO - AA 2015/2016 - Prof. Antonino Staiano



## semops ...

```
#include ...
int
main(int argc, char **argv)
{
    int c, i, flag, semid, nops;
    struct sembuf *ptr;
    flag = 0;
    while ( (c = getopt(argc, argv, "nu")) != -1) {
        switch (c) {
            case 'n':
                flag |= IPC_NOWAIT; /* per ogni operazione */
                break;

            case 'u':
                flag |= SEM_UNDO; /* per ogni operazione */
                break;
        }
    }
}
```

[ 33 ]

## ...semops

```
if (argc - optind < 2){ /* argc - optind = #args rimanenti */
    fprintf(stderr, "usage: semops [ -n ] [ -u ] <pathname>
    operation ...");
    exit(-1);
}
semid = semget(ftok(argv[optind], 0), 0, 0);
optind++;
nops = argc - optind;

/* alloca memoria per mantenere le operazioni, memorizzare,
ed eseguire */
ptr = calloc(nops, sizeof(struct sembuf));
for (i = 0; i < nops; i++) {
    ptr[i].sem_num = i;
    ptr[i].sem_op = atoi(argv[optind + i]); /* <0, =0, >0 */
    ptr[i].sem_flg = flag;
}
if(semop(semid, ptr, nops)<0)
    perror("semop");
exit(0);
}
```

[ 34 ]

## Esempio di esecuzione

```
$ touch /tmp/staiano
$ semcreate -e /tmp/staiano 3
$ semsetvalues /tmp/staiano 1 2 3
$ semgetvalues /tmp/staiano
semval[0] = 1
semval[1] = 2
semval[2] = 3
```

- Creiamo prima un file chiamato **/tmp/staiano** usato da **ftok()** per identificare l'insieme dei semafori
- **semcreate** crea un insieme con tre membri
- **semsetvalues** imposta i valori a 1, 2 e 3; questi valori sono stampati da **semgetvalues**

[ 35 ]

## Esempio di esecuzione

- Dimostriamo l'atomicità dell'insieme delle operazioni quando eseguite su di un insieme di semafori

```
$ semops -n /tmp/staiano -1 -2 -4
semctl error: Resource temporarily unavailable
$ semgetvalues /tmp/staiano
semval[0] = 1
semval[1] = 2
semval[2] = 3
```

- Specifichiamo il flag **(-n)** di non blocco e tre operazioni, ognuna delle quali decrementa un valore nell'insieme
  - La prima op. è OK, la seconda è OK, ma la terza non può essere eseguita (non possiamo sottrarre 4 dal terzo membro poiché il suo valore è 3)
  - Riceviamo un errore **EAGAIN** (abbiamo specificato modalità non bloccante)

[ 36 ]

## Esempio di esecuzione

- Dimostriamo ora la proprietà SEM\_UNDO dei semafori System V

```
$ semsetvalues /tmp/staiano 1 2 3
$ semops -u /tmp/staiano -1 -2 -3 /* SEM_UNDO */
$ semgetvalues /tmp/staiano
semval[0] = 1      /* tutte le modifiche annullate */
semval[1] = 2      /* quando semops è terminato */
semval[2] = 3
$ semops /tmp/staiano -1 -2 -3
$ semgetvalues /tmp/staiano /* senza SEM_UNDO */
semval[0] = 0
semval[1] = 0
semval[2] = 0
```

[ 37 ]

## Allocare e Deallocare Semafori

- Volendo impiegare le funzioni per operare con i semafori System V per la creazione di un semaforo binario potremmo creare le seguenti funzioni

```
int binary_semaphore_allocation(key_t key, int sem_flags)
{
    return semget(key, 1, sem_flags);
}
/* restituisce -1 in caso di errore */
int binary_semaphore_deallocate(int semid)
{
    return semctl(semid, 0, IPC_RMID);
}
```

[ 38 ]

## Inizializzare Semafori

```
int binary_semaphore_initialize(int semid, int initvalue)
{
    union semun argument;
    unsigned short values[1];
    values[0] = initvalue; /* Valore iniziale */
    argument.array = values;
    return semctl (semid, 0, SETALL, argument);
}
```

[ 39 ]

## Eseguire una wait

```
/* Attende, eventualmente con una sospensione del
processo chiamante in attesa passiva, che il
semaforo torni positivo. Quindi lo decrementa */
int binary_semaphore_wait(int semid)
{
    struct sembuf operations[1];
    /* Utilizziamo il primo ed unico semaforo */
    operations[0].sem_num = 0;
    /* L'op. decrementa il valore del semaforo */
    operations[0].sem_op = -1;
    /* UNDO automatico all'uscita */
    operations[0].sem_flg = SEM_UNDO;
    return semop(semid, operations, 1);
}
```

[ 40 ]

# Eseguire una Post

```
/* Incrementa il semaforo, ritorna
immediatamente*/
int binary_semaphore_post(int semid)
{
    struct sembuf operations[1];
    /* Utilizziamo il primo ed unico semaforo */
    operations[0].sem_num= 0;
    /* L'op. incrementa il valore del semaforo */
    operations[0].sem_op = 1;
    /* UNDO automatico all'uscita */
    operations[0].sem_flg = SEM_UNDO;
    return semop(semid,operations,1);
}
```

[ 41 ]

# Esempio

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

int wait(int semid, int semnum) /* operazione wait sulla componente
semnum di semid */
{
    struct sembuf operations;
    operations.sem_num=semnum;
    operations.sem_op=-1;
    operations.sem_flg=0;
    return semop(semid,&operations,1);
}

int post(int semid, int semnum) /* operazione post sulla componente
semnum di semid */
{
    struct sembuf operations;
    operations.sem_num=semnum;
    operations.sem_op=1;
    operations.sem_flg=0;
    return semop(semid,&operations,1); }
```

[ 42 ]

# Esempio (cont.)

```
int seminit(int semid, int semnum, int initval)
/* inizializzazione con initval della
componente semnum di semid */
{
    union semun
    {
        int val;
        struct semid_ds *buf;
        unsigned short *array;
    } arg;

    arg.val=initval;
    return semctl(semid,semnum,SETVAL,arg);
}
```

[ 43 ]

# Esempio (cont.)

```
int main() {
    int semid,val;
    long i;
    semid = semget(IPC_PRIVATE,1,0600);
    printf("Valore semid: %d\n",semid); fflush(stdout);
    if (semid == -1){
        fprintf(stderr,"Creazione semaforo");
        exit(-1); }
    seminit(semid,0,0); // args(int semid,int semnum,int initval)
    val = semctl(semid,0,GETVAL);
    if (val==-1) {
        fprintf(stderr,"Lettura valore semaforo");
        exit(-1); }
    printf("Valore semaforo %d: %d\n",semid,val); fflush(stdout);
    if (fork() == 0) {
        for (i=0; i< 10000000;i++){
            printf("Do via libera\n");
            post(semid,0); } //args (int semid, int semnum)
    else {
        wait(semid,0); //args (int semid, int semnum)
        printf("Ricevuto via libera\n");
        for (i=0; i< 10000000;i++){
            if (semctl(semid,0,IPC_RMID)==-1) {
                fprintf(stderr, "Rimozione semaforo");
            }exit(0); }
```

[ 44 ]

## Esercizio

- Implementare i problemi:

- del produttore e del consumatore
- dei produttori e del consumatore

visti per i semafori Posix con i semafori System V

[ 45 ]

## Cenni di memoria condivisa (System V)

[ 46 ]

## Memoria Condivisa

- Altra forma di IPC
  - Consente a due processi qualsiasi di accedere alla stessa memoria logica
- La memoria condivisa è un modo molto efficiente di trasferire i dati tra due processi in esecuzione
- La memoria condivisa è uno speciale range di indirizzi creato dall'IPC per un processo e appare nello spazio degli indirizzi di quel processo
  - Altri processi possono "attaccare" lo stesso segmento di memoria condivisa nel proprio spazio degli indirizzi
  - Tutti i processi possono accedere alle locazioni di memoria come se la memoria fosse stata allocata con una **malloc()**
  - Se un processo scrive nella memoria condivisa, le modifiche divengono immediatamente visibili a qualsiasi altro processo che ha accesso alla stessa memoria condivisa

[ 47 ]

## Memoria Condivisa

- La memoria condivisa di per se non fornisce meccanismi di sincronizzazione
  - Non ci sono meccanismi automatici per prevenire che un secondo processo inizi a leggere nella memoria condivisa prima che il primo processo ha finito di scriverci
  - E' responsabilità del programmatore sincronizzare gli accessi

[ 48 ]

# Funzioni per la memoria condivisa

- Le funzioni per la memoria condivisa somigliano a quelle per i semafori:

```
#include <sys/shm.h>
int shmget(key_t key, size_t size, int shmflg);
void *shmat(int shm_id, const void * shm_addr, int shmflg);
int shmctl(int shm_id, int cmd, struct shmid_ds *buf);
int shmdt(const void *shm_addr);
```

- Come per i semafori, sono richiesti anche `<sys/types.h>` e `<sys/ipc.h>`

[ 49 ]

# Memoria condivisa

- Il kernel mantiene una struttura con almeno i seguenti membri per ogni segmento di memoria condivisa:

```
struct shmid_ds {
    struct ipc_perm  shm_perm;
    size_t          shm_segsz; /* dim. del segmento in byte */
    pid_t           shm_lpid;   /* pid ultima shmop() */
    pid_t           shm_cpid;   /* pid del creatore */
    shmatt_t        shm_nattach; /* num. di attacchi */
    time_t          shm_atime;  /* istante ultimo attach */
    time_t          shm_dtime;  /* istante ultimo detach */
    time_t          shm_ctime;  /* istante ultima modifica */
    ...
}
```

[ 50 ]

## shmget

- La prima funzione chiamata, per ottenere un identificatore di memoria condivisa, è

```
#include <sys/shm.h>
int shmget(key_t, size_t size, int flag);
/* restituisce ID memoria condivisa se OK, -1 in caso di errore */
```

- Parametro **key** stesso significato per i semafori
- Parametro **size**: dimensione del segmento di memoria condivisa espresso in byte
  - Se stiamo creando un nuovo segmento è necessario specificare **size**
  - Se stiamo facendo riferimento ad un segmento esistente, **size = 0**

[ 51 ]

## shmat

- Una volta creato un segmento di memoria condivisa, un processo lo concatena al proprio spazio di indirizzi chiamando **shmat()**

```
#include <sys/shm.h>
void *shmat(int shm_id, const void *addr, int flag);
/* restituisce un puntatore al segmento di memoria condivisa se OK, -1 in caso di errore */
```

- L'indirizzo nel processo chiamante a cui il segmento è attaccato dipende dall'argomento **addr** e se il bit **SHM\_RND** è specificato in flag (quest'ultimo può anche assumere il valore **SHM\_RDONLY**)
  - Se **addr** è 0, il segmento è attaccato al primo indirizzo disponibile selezionato dal kernel (raccomandato)
  - Se **addr** è diverso da 0 e **SHM\_RND** non è specificato, il segmento è attaccato all'indirizzo dato da **addr**
  - Se **addr** è diverso da 0 e **SHM\_RND** è specificato, il segmento è attaccato all'indirizzo **addr** approssimato per difetto al più vicino multiplo di **SHMLBA** (low boundary address)
- Il valore ritornato **shmat()** è l'indirizzo a cui il segmento è attaccato

[ 52 ]

# shmctl

- La funzione **shmctl()** controlla le varie operazioni sulla memoria condivisa

```
#include <sys/shm.h>
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
/* restituisce 0 se OK, -1 in caso di errore */
```

- cmd**: specifica uno dei seguenti comandi da eseguire sul segmento specificato da **shmid**:
  - IPC\_STAT** preleva la struttura **shmid\_ds** per il segmento e lo memorizza nella struttura puntata da **buf**
  - IPC\_SET** imposta i campi **shm\_perm.uid**, **shm\_perm.gid**, **shm\_perm.mode**
  - IPC\_RMID** rimuove il segmento di memoria condivisa dal sistema

[ 53 ]

# shmdt

- Terminato l'utilizzo del segmento di memoria condivisa, lo deallochiamo con **shmdt()**

```
#include <sys/shm.h>
int shmdt(void *addr);
/* restituisce 0 se OK, -1 in caso di errore */
```

- Osserviamo che non viene rimosso dal sistema l'identificatore e la sua struttura dati associata
  - L'identificatore resta nel sistema fino a che qualche processo lo rimuove specificamente chiamando **shmctl()** con un comando di **IPC\_RMID**
- L'argomento **addr** è il valore ritornato dalla chiamata precedente a **shmat()**

[ 54 ]

## Esempio: uso memoria condivisa

```
shmid=
shmget(IPC_PRIVATE, SEGSIZE, IPC_CREAT|0666);
char *segptr;
segptr=shmat(shmid, 0, SHM_RDONLY);
```

\\si utilizza la memoria condivisa

```
shmdt(segptr); /* non rimuove l'id e la
struttura dati associata dal sistema,
semplicemente non è più a disposizione */
```

```
shmctl(shmid, IPC_RMID, 0);
\\ rimuove la sh. memory
```

[ 55 ]

## Esempio: Scrittura

```
#include <sys/ipc.h>
#include <stdio.h>
#include <sys/shm.h>
#define SHMSIZE 1000
int main(int argc, char* argv[]) {
    key_t chiaveipc;
    int tmp, shmid;
    char* shptr;
    char buffer[SHMSIZE];
    if (argc != 2) { printf("Mi serve la chiave IPC\n");
        exit(1); }
    chiaveipc = atoi(argv[1]);
    shmid=shmget(chiaveipc, SHMSIZE, IPC_CREAT|0600);
```

[ 56 ]

## Esempio: Scrittura

```
shptr = (char*) shmat(shmid, 0, 0);
printf("Immetti una stringa:\n");
fgets(buffer, SHMSIZE, stdin);
strncpy((char*) shptr, buffer, SHMSIZE);
printf("Fatto.\n");
shmdt(shptr);
exit(0);
}
```

## Esempio: Lettura

```
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHMSIZE 100
int main(int argc, char* argv[]) {
    key_t chiaveipc;
    int shmid;
    char* shptr;
    char buffer[SHMSIZE];
    if (argc != 2) { printf("Mi serve la chiave IPC\n");
        exit(1); }
    chiaveipc = atoi(argv[1]);
    shmid = shmget(chaveipc, 0, 0);
```

## Esempio: Lettura

```
shptr = (char*) shmat(shmid, 0, 0);
strncpy(buffer, shptr, SHMSIZE);
buffer[SHMSIZE] = '\0';
printf("Nella memoria condivisa : \n%s\n", buffer);
shmdt(shptr);
exit(0);
}
```