Laboratorio di Sistemi Operativi

Marzo-Giugno 2008
Matricole congrue 0 mod 3

IPC:
Memoria condivisa
e
Semafori

Memoria Condivisa

- Permette a due o più processi di condividere una zona di memoria
- È l'IPC più veloce perché in questo caso non è necessario copiare i dati tra server e client
- L'unico problema è la sincronizzazione per l'accesso (spesso i semafori sono usati per effettuare tale sincronizzazione)

Struttura associata

```
struct shmid_ds {
   struct ipc_perm  shm_perm; /* see Section 14.6.2 */
   struct anon_map *shm_amp;  /* pointer in kernel */
   int      shm_segsz;   /* size of segment in bytes */
   ushort   shm_lkcnt;   /* number of times segment is being locked */
   pid_t    shm_lpid;    /* pid of last shmop() */
   pid_t    shm_cpid;    /* pid of creator */
   ulong    shm_nattch;  /* number of current attaches */
   ulong    shm_cnattch; /* used only for shminfo */
   time_t   shm_atime;   /* last-attach time */
   time_t   shm_dtime;   /* last-detach time */
   time_t   shm_ctime;   /* last-change time */
};
```


Limiti di sistema

Name	Description	Typical Value
SHMMAX	The maximum size in bytes of a shared memory segment.	33554432
SHMMIN	The minimum size in bytes of a shared memory segment.	1
SHMMNI	The maximum number of shared memory segments, systemwide.	4096
SHMSEG	The maximum number of shared memory segments, per process.	4096

Funzione shmget

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

int shmget(key_t key, int size, int flag );
```

Restituisce: ID del segmento di memoria condivisa se OK,
-1 in caso di errore

shmget

- Quando si crea un nuovo segmento alcuni dei membri di shmid_ds sono settati
  - si inizializza ipc_perm con i permessi specificati in *flag*
  - shm_lpid, shm_nattach, shm_atime sono settati a 0
  - shm_ctime è settato all'istante di creazione

Es: shmid = shmget (key, 1000, IPC_CREAT|IPC_EXCL|0666);

- Quando ci si riferisce ad un'area di memoria condivisa esistente si usa size=0

Es: shmid = shmget (key, 0, 0);

Funzione shmctl

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

int shmctl(int shmid, int cmd, struct shmid_ds *buf );
```

Restituisce: 0 se OK
-1 in caso di errore

argomento *cmd*

- *cmd* rappresenta il comando da eseguire su *shmid*

IPC_STAT = recupera la struttura shmid_ds e la conserva nella struttura puntata da *buf*

IPC_SET = setta il campo shm_perm in shmid_ds secondo quanto contenuto nella struttura puntata da *buf*

IPC_RMID = rimuove la memoria condivisa dal sistema

SHM_LOCK = mette un "lock" sulla memoria condivisa; può essere eseguito solo da superuser

SHM_UNLOCK = disattiva il "lock" sulla memoria condivisa; può essere eseguito solo da superuser

Funzione shmat

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

void *shmat(int shmid, void *addr, int flag );
```

- Il processo che chiama shmat collega la memoria condivisa al suo spazio degli indirizzi (al primo indirizzo utile se *addr* =0; in genere è così)
- Se *flag*=SHM_RDONLY si potrà solo leggere il segmento
- Restituisce il puntatore al segmento se OK, -1 in caso di errore

Funzione shmdt

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

int shmdt(void *addr);
```

- Scollega la memoria condivisa dal proprio spazio di memoria (nota: non rimuove la memoria e le sue strutture dal sistema; la rimozione avviene con shmctl ed il comando IPC_RMID)
- Restituisce 0 se OK, -1 in caso di errore

esempio d'uso

```
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 shared memory (di solito il
                           \\server)
```


esempio: scrivi su memoria condivisa

```
#include <sys/ipc.h>
#include <stdio.h>
#include <sys/shm.h>
#define SHMSIZE 1000
int main() {
    int tmp, shmid; char* shptr; char buffer[SHMSIZE];
    shmid=shmget(IPC_PRIVATE,SHMSIZE,IPC_CREAT|0600);
    printf("\n L'id del segmento di shm creato e': %d\n", shmid);
    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: scrivi su memoria condivisa

```
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: leggi da memoria condivisa

```
#include <sys/ipc.h>
#include <sys/shm.h>
#define SHMSIZE 100
int main(int argc, char* argv[]) {
   int shmid; char* shptr; char buffer[SHMSIZE];
   if (argc != 2) { printf("Mi serve l'id \n"); exit(1);}
   shmid = atoi(argv[1]);
   shptr = (char*)shmat(shmid,0,0);
   strncpy(buffer,shptr,SHMSIZE);
   buffer[100]=0;
   printf("Nella memoria condivisa c'e':\n%s\n",buffer);
   shmdt(shptr); exit(0);
}
```

esempio: leggi da memoria condivisa

```
  shptr = (char*)shmat(shmid,0,0);

    strncpy(buffer,shptr,SHMSIZE);
    buffer[100]=0;
    printf("Nella memoria condivisa c'e':\n%s\n",buffer);

  shmdt(shptr);
    exit(0);
}
```


esempio 2: uso memoria condivisa

```
#include <sys/ipc.h>
#include <stdio.h>
#include <sys/shm.h>
int main( ) {
    key_t my_key=1234;
     int* ptr;

    shid=shmget(my_key,sizeof(int),IPC_CREAT|0666);

     ptr=(int*)shmat(shid,0,0);
     *ptr=0;              \\ si può utilizzare lo spazio di memoria come si utilizza
                    \\ in C un qualunque spazio di memoria puntato da un puntatore
     shmdt(ptr);

exit(0);
}
```

Mem Condivisa ed ereditarieta'

fork: il figlio eredita ogni collegamento a memoria condivisa

exec: ogni collegamento a memoria condivisa viene chiuso

esempio 3: ereditarieta'

```
#include <sys/ipc.h>
#include <unistd.h>
#include <sys/shm.h>
int main( ) {
      key_t my_key=1234;
      int pid, sh_id;
      int* ptr;
      sh_id=shmget(IPC_PRIVATE,sizeof(int),IPC_CREAT|0666);
      ptr=shmat(sh_id,0,0);        *ptr=0;

      if(pid = fork()< 0){printf("error"); exit(0);}
      if(pid !=0){ *ptr = 7;
            ...          \\ padre
      shmdt(ptr);}
      else{sleep(2); printf("\n %d \n", *ptr);}   \\ figlio

      exit(0);
}
```

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

- forma più complessa:

  inizializzato ad un valore positivo indicante il numero di risorse che sono a disposizione per essere condivise

semafori

- Secondo XSI:

  un semaforo e' in realta' un insieme di semafori --->bisogna specificarne il numero

Utilizzeremo:

> ipersemaforo per riferirci all'insieme

> semaforo per riferirci agli elementi dell'insieme

(iper)semafori: Strutture associate

```
struct semid_ds  {
   struct ipc_perm    sem_perm;  /* vedi code di messaggi*/
   struct sem  *sem_base; /* ptr al primo sem. Nell'Ipers.*/
   ushort      sem_nsem;  /* # semafori nell'Ipersemaforo*/
   time_t      sem_otime; /* tempo ultima op. sull'Ipers. */
   time _t     sem_ctime; /* tempo ultima modifica */
```

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

```
struct sem {
   ushort semval /* valore del semaforo*/
   pid_t  sempid; /* pid ultima operazione*/
   ushort semcnt; /* #proc. in attesa di semval > currval */
   ushort semzcnt; /* #proc. in attesa di semval =0 */
```

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 (Iper)semaforo se OK,
-1 in caso di errore

semget

- Quando un (iper)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* (# semafori nell'ipersemaforo)

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

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

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

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);
```

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


argomento *cmd*

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

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

- *cmd* rappresenta il comando da eseguire sul (iper)semaforo identificato da *semid*

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

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

IPC_RMID = rimuove il semaforo (l'insieme)

GETVAL = ritorna il valore del semaforo numero *semnum* nell'Ipersemaforo *semid*

argomento *cmd*

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

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

SETVAL = setta il contatore del semaforo numero *semnum* nell'ipersemaforo *semid,* al valore *arg.val*

GETALL = recupera i valori dei semafori presenti nell'Ipersemaforo *semid* e li mette *arg.array*

SETALL = Per ogni j = 0...semnum-1, setta il valore del semaforo j nell'ipersemaforo *semid* al valore *arg.array[j]*

union semun

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

- il campo *val* si usa con SETVAL per settare il valore del semaforo *semnum* dell'ipersemaforo identificato da *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 l' ipersemaforo
- semctl (semid, 0, SETVAL, 1) ;
  - setta il valore del (primo) semaforo ad 1
- i =semctl (semid, 2, GETVAL, 0)
  - i contiene il valore del terzo semaforo
  - il quarto argomento è ignorato

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,* sull'ipersemaforo 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 semaforo
- 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;
sem_lock.sem_num=0;
sem_lock.sem_op=-1;
sem_lock.sem_flg=0;
```

esempio d'uso (1)

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

struct sembuf sem_unlock;
sem_lock.sem_num=0;
sem_lock.sem_op=1;
sem_lock.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 bloccato\n");

// utilizzo della risorsa gestita dall'unico semaforo nell'Ipersemaforo

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


esempio d'uso (2)

```
struct sembuf sem_lock_0={0, -1, 0};
struct sembuf sem_unlock_0={0, 2, 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);
semop(semid,&sem_lock_0,1);
printf("sem 0 bloccato\n");

   // utilizzo della risorsa gestita dal semaforo #0 nell'ipersemaforo

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