INTERPROCESS COMMUNICATION 27

- Le comunicazioni tra processi (IPC, Intreprocess Communication) sono realizzate mediante strutture dati rese disponibili dal kernel.

Sono disponibili 3 tipologie di comunicazioni tra processi:
- memoria condivisa (**shared memory segments**)
- semafori (**semaphore arrays**)
- code di messaggi (**message queues**)

- Ciascuna di queste strutture dati sono indicate da un identificatore, che ha lo stesso significato dell'identificatore di file aperto.
- Mediante tale identificatore i processi possono acquisire, utilizzare e rimuovere le strutture.
- Tale identificatore viene ottenuto da una system call, specificando alcuni parametri per permettere ad insiemi di processi diversi di condividere strutture diverse.

- Esiste un eseguibile **/usr/bin/ipcs** che permette di visualizzare le strutture allocate, e che mostra anche l'identificatore di ciascuna struttura e l'utente proprietario.

- Qualora i processi non abbiano rimosso le strutture allocate (ad es. in caso di terminazione anomala), per rimuoverle si può utilizzare il pro_ gramma **/usr/bin/ipcrm** per rimuovere una data struttura noto il suo identificatore.

Output del Programma ***ipcs***

```
[vittorio@poseidon prova]$ ipcs

------ Shared Memory Segments --------
key            shmid      owner      perms      bytes      nattch     status
0x00000001 3969       vittorio    600        100        0

------ Semaphore Arrays --------
key            semid      owner      perms      nsems      status

------ Message Queues --------
key            msqid      owner      perms      used-bytes  messages
```

Shared Memory Segments

Normalmente i processi Unix non condividono la memoria;
nemmeno i processi 'parenti':
infatti il figlio eredita UNA COPIA dei dati del padre;
se i due processi modificano quei dati dopo la fork, ognuno modifica la propria copia.
La possibilita' di condividere memoria e' stata aggiunta (come le altre funzioni IPC) nella versione System V e poi utilizzata anche nelle altre versioni di Unix.

Il segmento di memoria condivisa è caratterizzato da un identificatore (detto chiave, key).

Quindi se piu' processi vogliono usare lo stesso segmento di memoria condivisa, questi dovranno richiederla esplicitamente, specificando tutti la stessa key (devono sapere il valore di key).

Oppure, se i processi che devono condividere il segmento di memoria sono tutti figli di uno stesso processo, il padre prima di creare i figli potrà mappare la memoria condivisa, ed i figli avranno automaticamente una copia della chiave di accesso alla memoria condivisa.

Questi due modalità sono realizzate con un uso diverso delle stesse system call.

Shared Memory Segments

Ad un segmento di memoria condivisa si accede tramite le seguenti system calls

int **shmget** (key_t **key**, int **size**, int **shmflg**);
void ***shmat** (int **shmid**, void ***shmaddr**, int **shmflg**);

definiti negli header

sys/types.h , sys/ipc.h e sys/shm.h

shmget restituisce un identificatore di un segmento di memoria condivisa di dimensione size;

a seconda del valore di key, il segmento viene creato o no.
Il segmento viene CREATO se
- key vale IPC_PRIVATE (0),

oppure se
- non esiste il segmento associato a key, e semflg&IPC_CREAT e' vero,

Quindi se si vuole che piu' processi acquisiscano la stessa struttura si puo':

1) far partire indipendentemente diversi processi e fare chiamare a tutti la *semget* con la stessa chiave key diversa da IPC_PRIVATE.
   Tale key deve essere DIVERSA da chiavi usate da altri processi anche di altri utenti
2) far acquisire le strutture da un processo padre e farle 'ereditare' dai processi figli, che quindi non devono fare semget, ma solo usare l'identificatore restituito dalla semget al padre. In questo caso il padre puo' usare IPC_PRIVATE come chiave.

il parametro shmflg definisce i permessi di lettura scrittura

La system call

void ***shmat** (int **shmid**, void ***shmaddr**, int **shmflg**);

collega shmget restituisce un identificatore di un segmento di memoria

- shmat esegue il collegamento del segmento di memoria condivisa, ottenuto dalla chiamata alla shmget, ad un indirizzo nello spazio di indirizzi del processo chiamante.
- Praticamente, restituisce un indirizzo che punta all'area di memoria condivisa, che potrà essere utilizzato per i successivi accessi.
- Se il parametro shmaddr e' NULL (consigliato), l'indirizzo viene scelto dal sistema, altrimenti dall'utente.
- Il parametro shmflg, se posto a zero, assegna capacità di leggere e scrivere nel segmento di memoria condivisa.

La system call

int **shmdt** (char *shmaddr);

scollega l'area di memoria condivisa puntata da shmaddr dallo spazio di memoria indirizzabile dall'utente, ma non rimuove l'area condivisa.

La system call

int **shmctl** (int shmid, int cmd, struct shmid_ds

serve per rimuovere un segmento di memoria condivisa.
Si specifica cmd = IPC_RMID
e terzo parametro nullo.

Shared Memory: Esempio Padre/Figlio

```
#include <malloc.h>  #include <stdio.h> #include <sys/types.h>
#include <sys/ipc.h> #include <sys/sem.h>
main()  {
int shmid1; char *addr1;
/* alloca memoria condivisa per un array di 100 caratteri */
shmid1 = shmget(IPC_PRIVATE,100*sizeof(char),0600);
if (shmid1 == -1)
      { perror("Creazione memoria condivisa"); exit(1); }
// ottengo un puntatore all'area condivisa
addr1 = (char *) shmat ( shmid1, 0, 0);
/*    E SE AVESSI ALLOCATO in maniera canonica ?
      addr1= malloc(100);
      if(addr1==NULL) printf("errore nella malloc"); exit(0);
      */
// assegno un valore al primo carattere
*addr1='A';
if (fork() == 0)
  {   // processo figlio, cambia valore al primo carattere
      *addr1='C';
      shmdt(addr1);
      printf("figlio: carattere %c\n", *addr1);
   }
else  {  // processo padre, attendo che il figlio abbia scritto
      sleep(5);
      printf("padre: carattere: %c\n", *addr1);
      // attendo che il figlio termini
      wait(NULL);
      shmctl(shmid1,IPC_RMID,NULL);
   }
}
```

lancio prima il processo 1, e immediatamente il processo 2

```
// processo 1
main()  { int shmid1; char *addr1;
        int key=1;
        shmid1 = shmget(key,100*sizeof(char),0600);
        addr1 = (char *) shmat ( shmid1, 0, 0);
        // assegno un valore al primo carattere
        *addr1='A';
        printf("proc1: prima, carattere: %c\n", *addr1);
        // attendo che l'altro processo scriva
        sleep(10);
        // dovrei stampare B
        printf("proc1: dopo, carattere: %c\n", *addr1);
        shmctl(shmid1,IPC_RMID,NULL);
}

// processo 2
main()  {
        int shmid1; char *addr1;
        int key=1;
        shmid1 = shmget(key,100*sizeof(char),0600);
        addr1 = (char *) shmat ( shmid1, 0, 0);
        // LEGGO
        printf("proc2: prima carattere: %c\n", *addr1);
        *addr1='B';
        shmdt(addr1);
}
```