

Meccanismi di Comunicazione Interprocesso (IPC)

- segnali
- pipe (anonymous o FIFO)
- ⇒ shared memory
- socket
- sincronizzazione tra processi
 - semafori
- credits:
<http://www.dia.uniroma3.it/~morgia>

Sommario

- Introduzione
- Schema dei diritti di processo
 - ? Esempio
- Primitive
 - ? shmget
 - ? shmctl
 - ? shmat/dt
- Esempio
- Vantaggi/svantaggi
- Conclusioni

Introduzione

- Meccanismo di comunicazione interprocesso
- Poche primitive
- Si usa come i normali puntatori
- Protezione basata sui diritti
- Gestito direttamente dal kernel
 - ? veloce
 - ? sicuro

Schema dei Diritti di Processo

- Processo attivato da un utente eredita:
 - ? user identifier
 - ? group identifier
- L'autenticazione per shared memory è basata sull'identità di processo
- Ogni segmento condiviso è descritto da un set di diritti identico a quello usato da UNIX per i file:

utente			gruppo			resto		
r	w	x	r	w	x	r	w	x

Esempio

- Processo A (uid=505 guid=100) crea un segmento in shared memory con diritti **rw-rw-r** (0662):
 - ? stesso user id = lettura/scrittura
 - ? stesso group id = lettura/scrittura
 - ? altri = solo lettura
- Processo B (uid=507 guid=100) accede al segmento:
 - ? scrittura ok
 - ? lettura ok
- Processo C (uid=510 guid=171) accede al segmento:
 - ? lettura ok

Primitive

- Insieme di primitive contenuto
- **shmget** - creazione/accesso identificatore di segmenti
- **shmctl** - controllo accesso ed informazioni
- **shmat** - attachment segmenti condivisi
- **shmdt** - rilascio segmenti condivisi

shmget (1)

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

man 2 shmget

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

```
int shmget(key_t key, int dim, int flags);
```

- `key` è un intero che rappresenta il segmento
 - ? in creazione si impone una chiave
 - ? in accesso è necessario conoscere la chiave imposta
 - ? se `key = IPC_PRIVATE` viene creato un segmento ex-novo
- `dim` è la dimensione del segmento arrotondata al multiplo intero superiore della costante **PAGE_SIZE**
 - ? in creazione si impone una dimensione
 - ? in accesso si può richiedere un sottosegmento

shmget (2)

- flags può essere:
 - ? **IPC_CREAT** per creare un nuovo segmento, oppure
 - ? **IPC_EXCL** che usato insieme ad **IPC_CREAT** causa il fallimento in caso già esista il segmento, oppure
 - ? 9 bit di maschera per i diritti **rx?-rw?-rx?**
- crea un descrittore di segmento ed inserisce **uid** e **gid** del creatore, insieme a dimensione e permessi
- restituisce
 - ? un intero utilizzabile come descrittore del segmento
 - ? in caso di errore -1

shmget (3)

- In caso di fallimento, possono trovarsi i seguenti valori nella variabile globale **errno**:
 - ? **EINVAL** se la dimensione non è valida
 - ? **EEXIST** se, specificando **IPC_CREAT** e **IPC_EXCL**, risultasse che il segmento già esiste
 - ? **EIDRM** se il segmento è marcato per la distruzione
 - ? **ENOSPC** mancano id liberi per shared memory
 - ? **ENOENT** se il segmento specificato dal parametro *key* non esiste (e non è stata richiesta la creazione)
 - ? **ENOACCES** se il richiedente non ha diritti
 - ? **ENOMEM** se è esaurita la memoria per i descrittori

Come Ottenere un Valore Unico per il parametro `key`

- Si è visto che i descrittori di file una volta ottenuti sono lo strumento per indicare un file alle primitive di gestione. Il valore restituito da `shmget ()` svolge un ruolo analogo per i segmenti di memoria condivisa

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

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

man 3 ftk

```
key_t ftok (char *pathname, char proj)
```

- restituisce un valore univocamente associato ad una coppia nome di file esistente, singolo carattere
- restituisce `-1` e (setta **`errno`**) in caso di errore

shmctl (1)

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

man 2 shmctl

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

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

- `shmid` è un ID di segmento ritornato da `shmget ()`
- `cmd` è un codice di comando tra
 - ? **IPC_STAT** per ricevere informazioni sul segmento (messe nella struttura preallocata e puntata da `buf`) se il processo ha diritti di lettura sul segmento
 - ? **IPC_SET** per cambiare tali informazioni (prelevabili dalla struttura puntata da `buf`) se il processo è il creatore o il possessore del segmento oppure il superuser
 - ? **IPC_RMID** per marcare il segmento per la distruzione (secondo gli stessi diritti di sopra)

shmctl (2)

- ? **IPC_LOCK** serve per non permettere lo swap del segmento e mantenerlo permanentemente in memoria centrale (solo superuser)
- ? **IPC_UNLOCK** serve per ammettere lo swap di un segmento precedentemente bloccato con **IPC_LOCK** (solo superuser)

shmctl (3)

Le Strutture shmids e ipc_perm

```
struct shmids {  
    struct ipc_perm shm_perm;  
    int shm_segz; ← dimensione  
    unsigned short shm_cpid; ← processo creatore  
    unsigned short shm_lpid; ← ultimo processo  
    short shm_nattch; ← modificatore  
    ... ..  
}; ← numero processi agganciati
```



```
struct ipc_perm {  
    key_t key; ← identificatore segmento (chiave)  
    ushort uid, gid; ← uid e gid del possessore del segmento  
    ushort cuid, cgid; ← uid e gid del creatore del segmento  
    ushort mode; ← permessi associati al segmento rw-rw-rw  
    ushort seq; ← modificabili da IPC_SET  
};
```

shmctl (4)

- In caso di fallimento di **shmctl()**, la variabile globale **errno** assume i seguenti valori:
- **EACCES** su **IPC_STAT** per processo senza diritto di lettura
- **EFAULT** il puntatore **buf** al contenitore non è valido
- **EINVAL** ID segmento o comando non validi
- **EIDRM** ID di segmento rimosso
- **EPERM** permessi insufficienti per l'operazione richiesta

shmat (1)

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

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

man 2 shmat

```
void *shmat(int id, const void *addr, int flags);
```

- Aggancia un segmento shared memory allo spazio di memoria del processo chiamante:
 - ? `id` è l'identificatore del segmento restituito da `shmget ()`,
 - ? `addr` è il puntatore utilizzabile per accedere al segmento stesso ed è un valore ritornato,
 - ? `flags` può essere **SHM_RDONLY** per ottenere un puntatore di sola lettura al segmento (per non violare i permessi)

shmat (2)

- Ritorna un indirizzo valido (lo stesso contenuto in `addr`) oppure `-1` in caso di errore e la variabile `errno` può assumere i seguenti valori:
 - ? **EACCES** problema con i permessi
 - ? **EINVAL** problema con i parametri
 - ? **ENOMEM** memoria insufficiente per allocare i descrittori interni

shmdt

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

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

```
int shmdt(const void *addr);
```

man 2 shmdt

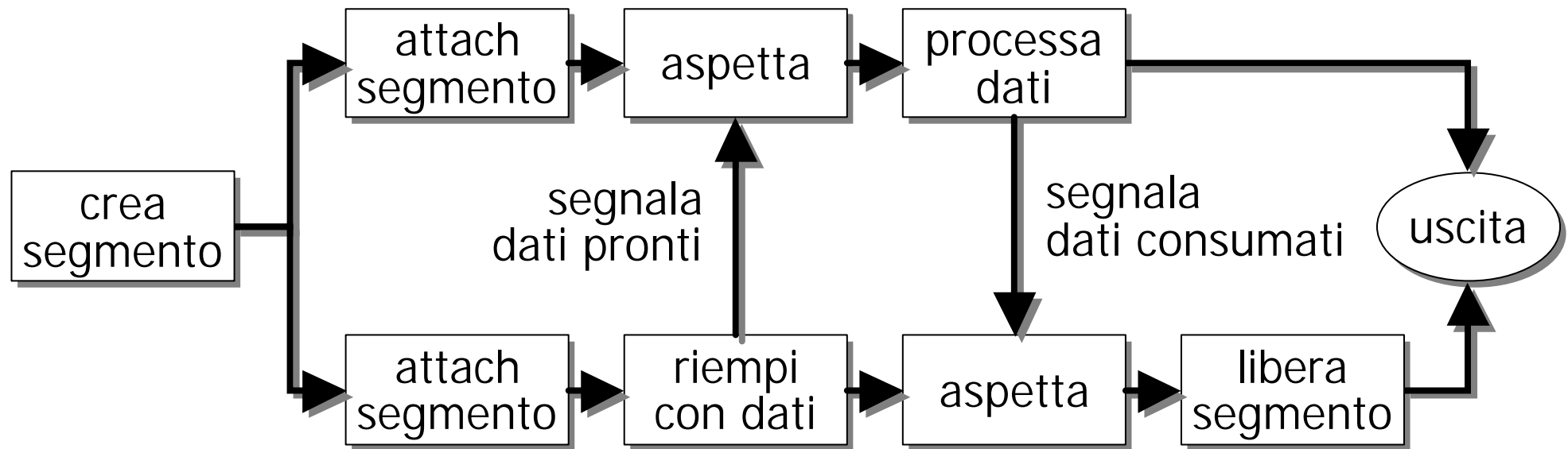
- Rilascia il segmento shared memory agganciato tramite `shmat ()` puntato dal parametro `addr`.
- Restituisce `-1` in caso di errore

Note sull'ereditarietà

- L'esecuzione di una `fork()` causa la creazione di un processo che **eredita** tutti i segmenti di shared memory attivi sul padre.
- L'esecuzione di `exec()` causa il **rilascio** (non la distruzione) di tutti i segmenti shared memory.
- Anche l'esecuzione di `exit()` causa il **rilascio** di tutti i segmenti shared memory.

Esempio

Shared Memory con `fork()`



Esempio: Shared Memory con `fork()`

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

#define FILENAME "shmfork.c"
#define PROJ 'X'

int main(int argc, char **argv) {
    int *addr, *child_addr, i, shmid;
    pid_t pid;
    key_t key = ftok(FILENAME, PROJ);
    long somma;
    shmid=shmget(key,1024,IPC_CREAT|0666);
    if (shmid==-1) {
        perror("shmget");
        exit(2);
    }
    pid=fork();
    switch(pid) {
        case -1: perror("fork()");
                shmctl(shmid,IPC_RMID,0);
                exit(3);
                break;
        ...
    }
```

Esempio: Shared Memory con `fork()`

...

```
case 0: child_addr=shmat( shmid, NULL, 0 );
        if (child_addr==(void *)-1) {
            perror( "shmat(child)" );
            exit(4);
        }
        puts( "Child waiting...." );
        while (child_addr[1022]!=12) sleep(1);
        puts( "Child: shared segment available,
                processing..." );
        for (i=0; i<1024; i++)
            somma+=child_addr[i];
        child_addr[1023]=0;
        puts( "Child: disposing" );
        shmdt( child_addr );
        break;
```

...

Esempio: Shared Memory con `fork()`

```
...    default: printf("Child pid is %d\n",pid);
        addr=shmat(shmid,NULL,0);
        if (addr==(void *)-1) {
            perror("shmat(parent)");
            shmctl(shmid,IPC_RMID,0);
            exit(5);
        }
        puts("Server: filling with data...");
        for (i=0 ; i<1024 ; i++)
            addr[i]=i;
        puts("Server: sending signal");
        addr[1022]=12;
        puts("Server waiting...");
        while (addr[1023]!=0) sleep(1);
        puts("Server: disposing...");
        shmdt(addr);
        shmctl(shmid,IPC_RMID,NULL);
        break;
    }
    return 0;
}
```

Shared Memory con `fork()`: Output

```
[crescenz@diadema prove]$ ./shmfork
Child pid is 11280
Child waiting...
Server: filling with data..
Server: sending signal
Server waiting...
Child: shared segment available, processing..
Child: disposing
Server: disposing...
```

Esercizio: scrivere un programma equivalente che sincronizzi i due processi tramite segnali.

Vantaggi/Svantaggi

- Vantaggi

- ? Efficienti, il meccanismo IPC più efficiente
- ? Facilità di gestione
- ? Facilmente comprensibile perché assimilabile ai puntatori C

- Svantaggi

- ? Vincolato al singolo host
- ? Basso livello d'astrazione
- ? Necessità di sincronizzazione (semafori, ecc.)

Da Shell: ipcs

- Se lo shared memory segment non viene rilasciato nel codice, va fatto da shell.
- `ipcs` serve per ottenere informazioni sullo stato di:
 - ? shared memory [-m]
 - ? code di messaggi [-q]
 - ? semafori [-s]
 - ? tutto [-a]
- le informazioni ottenute sono:
 - ? indirizzo segmento
 - ? identificatore (`shm_id`)
 - ? uid possessore
 - ? permessi
 - ? lunghezza segmento e numero processo agganciati

Da Shell: ipcrm

```
[crescenz@diadema prove]$ ipcs -m
```

```
----- Shared Memory Segments -----
```

key	shmid	owner	perms	bytes	nattch	status
0x00280267	0	root	644	1048576	0	
0x7b4532d5	26881	crescenz	600	1024	4	

```
[crescenz@diadema prove]$ ipcs -mp
```

```
----- Shared Memory Creator/Last-op -----
```

shmid	owner	cpid	lpid
0	root	575	10539
26881	crescenz	10476	10495

- `ipcrm` serve per rimuovere segmenti, semafori o code:

? segmenti: `ipcrm [shm|msg|sem] <id>`

Esercizi

Esercizio: implementare un programma che simuli una pipe tramite memoria condivisa e segnali. In particolare un processo “client” emette una sequenza di numeri interi casuali in un range $[-n, n]$, dove n è un intero inserito dall’utente, ed un processo “server” li preleva per stamparne il quadrato.

Esercizio: simulare il gioco “battaglia navale” tra due processi che emettono coordinate casuali in una matrice $n \times n$ dove n è un intero predefinito. Si utilizzano i meccanismi IPC che più si ritengono opportuni per la sincronizzazione delle giocate, il rilevamento della fine di una partita, e per lo scambio delle coordinate delle celle “colpite”.