Corso di Programmazione Concorrente

Memoria Condivisa

Valter Crescenzi

http://crescenzi.inf.uniroma3.it

Condivisione di Memoria tra Processi

- I processi possiedono un spazio di memoria privato non accessibile normalmente ad altri
- Un apposito meccanismo IPC permette a due o più processi di condividere segmenti di memoria.
- I processi possono:
  - accedere concorrentemente allo stessa memoria
  - instaurare interazioni basate sulla disponibilità di memoria condivisa

Modello della Memoria Paginata

- La gestione della memoria UNIX si basa sulla paginazione
- Un segmento di memoria consiste in una o più pagine di memoria
- Gli indirizzi utilizzati da un processo sono virtuali, e vengono tradotti in indirizzi fisici sulla base di un mapping gestito dal kernel
- Gli indirizzi virtuali utilizzati da processi possono venire tradotti negli stessi indirizzi fisici
- Per condividere memoria tra più processi, solo uno deve allocarne un segmento, tutti gli altri devono *agganciarsi* al segmento
  - consiste in un aggiornamento del mapping dei propri indirizzi virtuali in modo da raggiungere gli indirizzi fisici del segmento condiviso
  - processi distinti finiscono per vedere le stesse aree di memoria fisica ciascuno ai propri indirizzi virtuali (possibilmente anche diversi dagli indirizzi virtuali usati dagli altri)

I Diritti di un Processo su di un Segmento di Memoria Condivisa

- Un processo eredita dal processo che lo ha creato:
  - user identificator
  - group identificator
- L'autenticazione per shared memory è basata sull'identità del processo
- Ogni segmento condiviso è descritto da un set di diritti identico a quello usato per i file:

Spazio di Indirizzamento

- Per poter permettere a più processi di condividere una risorsa è necessario un riferimento alla risorsa univoco e noto a tutti
- Per processi “parenti”:
  - utilizzando lo sdoppiamento dello spazio di memoria se i processi sono uno figlio dell’altro
- Per processi che non sono “parenti”:
  - utilizzando uno spazio di indirizzamento *esterno* per creare un riferimento comune

`ftok()`: Chiavi per IPC

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

`man 3 ftok`

```
key_t ftok(const char *pathname, int proj_id);
```

- alcune chiamate di sistema per IPC, usano chiavi per identificare univocamente le risorse allocate
- le chiavi possono essere ottenute con `ftok()` e risultano univocamente associate ai due parametri:
  - un file esistente ed accessibile di percorso assoluto `pathname`
  - un identificativo di progetto `proj_id`
- restituisce `-1` in caso di errore

Shared Memory: Chiamate di Sistema

- `shmget`
  - allocazione
- `shmctl`
  - varie funzionalità; deallocazione
- `shmat`
  - agganciamento ai segmenti
- `shmdt`
  - rilascio dei segmenti condivisi

shmget (1)

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, int dim, int flags);
```

`man 2 shmget`

- `key` è un intero che rappresenta il segmento
  - in creazione si usa una chiave che identifichi il nuovo segmento
  - in accesso è necessario conoscere la chiave utilizzata
  - 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 specifica una dimensione
  - in accesso si può richiedere un sottosegmento

`shmget` (2)

- `flags` può essere settato bit per bit come disgiunzione di:
  - `IPC_CREAT` per creare un nuovo segmento
  - `IPC_EXCL` usato insieme ad `IPC_CREAT` causa il fallimento in caso già esista il segmento
  - 9 bit per i permessi
- 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

shmctl

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

`man 2 shmctl`

- `shmid` è un ID di segmento ritornato da `shmget()`
- `cmd` è un codice che specifica l'operazione da effettuare sul segmento condiviso
- La struttura `shmid_ds` viene utilizzata come parametro ingresso/uscita alla `shmctl`

Parametro `cmd` di `shmctl()`

- `cmd` è un codice di comando tra
  - `IPC_STAT` per ricevere informazioni sul segmento
  - `IPC_SET` per cambiare tali informazioni
  - `IPC_RMID` per marcare il segmento per la distruzione
  - `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)
- Tutti i comandi sono soggetti ai relativi controlli sui permessi

shmctl: Le Strutture shmid_ds e ipc_perm

```
struct shmid_ds {
  struct ipc_perm shm_perm;
  int shm_segz;
  unsigned short shm_cpid;
  unsigned short shm_lpid;
  short shm_nattch;
  … …
}
```

dimensione

processo creatore

ultimo processo modificatore

numero processi agganciati

```
struct ipc_perm {
  key_t key;
  ushort uid,gid;
  ushort cuid,cgid;
  ushort mode;
  ushort seq;
}
```

identificatore segmento (chiave)

uid e **gid** del possessore del segmento

uid e **gid** del creatore del segmento

permessi associati al segmento **rw-rw-rw**

modificabili da **IPC_SET**

`shmctl`: Possibili Errori

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

`man 2 shmat`

- Aggancia un segmento allo spazio di memoria del processo chiamante:
  - `id` è il descrittore di segmento fornito da `shmget()`
- `addr` è l'ind. da utilizzare per accedere al segmento
  - passando `NULL` il sistema sceglie un indirizzo non utilizzato
  - `flags` può essere
    - `SHM_RDONLY` per abilitare il segmento alla sola lettura
    - `SHM_RND` per arrotondare l'indirizzo ad un multiplo della lunghezza di pagina

shmat (2)

- ritorna un indirizzo valido (lo stesso contenuto in `addr` eventualmente) oppure `-1` in caso di errore
- la variabile `errno` può assumere i valori:
  - `EACCES` problema con i permessi
  - `EINVAL` problema con i parametri
  - `ENOMEM` memoria insufficiente per strutture di supporto all'aggancio

shmdt

man 2 shmdt

```
#include <sys/types.h>
#include <sys/shm.h>
int shmdt(const void *addr);
```

- rilascia il segmento di indirizzo `addr` e lo stacca dallo spazio di memoria del processo chiamante
- restituisce `-1` in caso di errore

shm.c (1)

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
int main () {
  int segment_id;
  char* shared_memory;
  struct shmid_ds shmbuffer;
  int segment_size;
  const int shared_segment_size = 0x6400;

  /* Allocate a shared memory segment.  */
  segment_id = shmget(IPC_PRIVATE, shared_segment_size,
                      IPC_CREAT|IPC_EXCL|S_IRUSR|S_IWUSR);

  /* Attach the shared memory segment.  */
  shared_memory = (char*) shmat (segment_id, 0, 0);
  printf ("shared memory attached at address %p\n",
          shared_memory);
  …
```

shm.c (2)

```
… /* Determine the segment's size.  */
  shmctl (segment_id, IPC_STAT, &shmbuffer);
  segment_size = shmbuffer.shm_segsz;
  printf ("segment size: %d\n", segment_size);
  sprintf (shared_memory, "Hello, world.");
  shmdt (shared_memory);  // Detach

  // Reattach the segment at a different address
  shared_memory = (char*) shmat(segment_id,
                               (void*) 0x5000000, 0);
  printf ("shared memory reattached
                  at address %p\n", shared_memory);
  printf ("%s\n", shared_memory);
  shmdt (shared_memory);              // Detach
  shmctl (segment_id, IPC_RMID, 0); // Deallocate
  return 0;
}
```

shm.c: Esempio di Esecuzione

```
$ gcc shm.c –o shm
$ ./shm
shared memory attached at address 0x40015000
segment size: 25600
shared memory reattached at address 0x5000000
Hello, world.
$
```

Ereditarietà dei Segmenti

- L'esecuzione di una `fork()` causa la creazione di un processo che **eredita** tutti i segmenti di memoria condivisa del padre
- L'esecuzione di `exec()` causa il **rilascio** (non la distruzione) di tutti i segmenti
- Anche l'esecuzione di `exit()` causa il rilascio (non la distruzione) di tutti i segmenti
- Un segmento di cui si richiede la deallocazione verrà effettivamente rimosso solo quando tutti i processi che vi erano agganciati lo rilasciano

Shared Memory con `fork()`: `shm_fork.c` (1)

```
#include <stdio.h>
#include <sys/shm.h>
#include <sys/stat.h>
#include <sys/types.h>

int main() {
  pid_t pid;
  int segment_id;
  char* shared_memory;
  struct shmid_ds shmbuffer;
  int segment_size;
  const int shared_segment_size = 0x6400;

  /* Allocate a shared memory segment.  */
  segment_id = shmget(IPC_PRIVATE, shared_segment_size,
                      IPC_CREAT|IPC_EXCL|S_IRUSR|S_IWUSR);
…
```

Shared Memory con `fork()`: `shm_fork.c` (2)

```
… /* Attach the shared memory segment.  */
  shared_memory = (char*) shmat (segment_id, 0, 0);
  printf ("shared memory attached at address %p\n",
          shared_memory);
  /* Determine the segment's size.  */
  shmctl (segment_id, IPC_STAT, &shmbuffer);
  segment_size = shmbuffer.shm_segsz;
  printf ("segment size: %d\n", segment_size);
  // Write a string to the shared memory segment
  sprintf (shared_memory, "Hello, world.");
  /* Detach the shared memory segment.  */
  shmdt (shared_memory);
…
```

Shared Memory con fork(): shm_fork.c (3)

```
… if ((pid=fork())==0) {
        //Child: attach the shared memory segment
    shared_memory =

(char*)shmat(segment_id,NULL,SHM_RDONLY);
    printf("Child: shared memory attached”);
    printf(“ at address %p\n", shared_memory);
    /* Print out the string from shared memory.  */
    printf (“Read: %s\n", shared_memory);
    shmdt (shared_memory); // Detach
    return 0;
  }
  else { // parent
        wait();
    shmctl (segment_id, IPC_RMID, 0);
  }
}
```

Shared Memory con `fork()`: Output

```
$ ./shm_fork
shared memory attached at address 0x40015000
segment size: 25600
Child: shared memory attached at address 0x40015000
Read: Hello, world.
$
```

Esercizio: scrivere uno o più programmi equivalenti che utilizzino la exec().

Vantaggi/Svantaggi

- Vantaggi
  - efficienti, il più efficiente meccanismo IPC disponibile
  - facilità di gestione
  - semplice: assimilabile alla normale memoria
- Svantaggi
  - utilizzabile su una singola macchina
  - basso livello d'astrazione
  - lascia il compito di sincronizzare gli accessi completamente a carico del programmatore

Da Shell: `ipcs`

- I segmenti di memoria condivisa possono essere rilasciati anche utilizzando un apposito comando 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 (`shmid`)
  - uid possessore
  - permessi
  - lunghezza segmento e numero processo agganciati

Da Shell: ipcrm

```
$ ipcs -m

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

$ ipcs -mp

------ Shared Memory Creator/Last-op --------
shmid      owner      cpid       lpid
0          root       575        10539

26881      crescenz   10476      10495
```

- `ipcrm` consente di rimuovere segmenti e semafori:
  - `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 x 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”.