

Sviluppo di software di rete con l'utilizzo di sistemi UNIX

IPC: memoria condivisa

Claudio Vicari

Definizioni

- ➔ la memoria condivisa è un meccanismo delle IPC (sia Posix che SystemV) per condividere dati fra processi, trattandoli come se fossero normali variabili
- ➔ nella maggior parte dei casi, richiede un meccanismo di controllo e sincronizzazione per operare correttamente
- ➔ è il meccanismo di condivisione più rapido, perché richiede un coinvolgimento minimo da parte del kernel...

Esempio: client server con FIFO

- ➡ riconsideriamo l'esempio (visto con le FIFO) di comunicazione fra client e server
 - 1 il server legge un file
 - 2 il server copia il file in una FIFO
 - 3 il client legge i dati dalla FIFO
 - 4 il client li stampa a video
- ➡ Quindi, i dati sono su disco e vengono copiati nella memoria del server (passi 1 e 2)
- ➡ il server li copia in una FIFO (passo 2)
- ➡ il client li copia nella propria memoria, per poterli stampare
- ➡ Quindi, abbiamo quattro copie degli stessi dati! Più quella che sarà stampata a video...

Esempio non funzionante: uso di un numero di sequenza

incr1.c

- ➡ questo esempio utilizza un semaforo con nome, ma poiché il contatore count non è condiviso, i due processi ne hanno due copie differenti

La funzione mmap

```
#include <sys/mman.h>
void * mmap(void *start, size_t length,
            int prot , int flags, int fd,
            off_t offset);
```

- ➔ questa funzione è in grado di creare una corrispondenza (map) fra un'area di memoria ed il contenuto di un file
- ➔ si parte dall'offset specificato nel file, per un totale di *length* byte
- ➔ il file è già stato aperto, il suo file descriptor è *fd*

La funzione mmap /2

- ➔ il parametro **start* è un puntatore ad un'area di memoria in cui fare il *mapping* – ma è solamente un suggerimento al kernel
 - comunemente, si imposta a 0
 - mmap restituisce proprio un puntatore alla locazione di memoria da utilizzare
- ➔ il parametro *prot* indica la protezione da specificare
 - non deve andare in conflitto con il modo di apertura del file
 - può essere l'or di: PROT_EXEC, PROT_READ, PROT_WRITE, PROT_NONE (pagine non accessibili!)

La funzione mmap /3

- ➔ in flags si deve specificare uno fra MAP_SHARED (le modifiche vengono viste da tutti i processi e scritte sull'oggetto) e MAP_PRIVATE (visibili solo dal processo stesso)
- MAP_FIXED impone l'uso dell'indirizzo specificato, per portabilità non deve essere usato

La funzione munmap

```
int munmap(void *start, size_t length);
```

- ➔ `start` e `length` devono identificare la memoria restituita da `mmap`
- ➔ altri riferimenti a quegli indirizzi porterebbero ad un SIGSEGV
- ➔ fa un flush dei dati in memoria nel file
- ➔ se `mmap` era stato chiamato con `MAP_PRIVATE`, i dati sono scartati

La funzione msync

```
int msync(void *start, size_t length,  
          int flags);
```

- ➔ fa un flush esplicito dei dati in memoria nel file
- ➔ altrimenti, il processo non ha una garanzia che siano scritti fino alla chiamata di munmap
- ➔ flag:
 - con MS_ASYNC accodiamo l'operazione nel kernel, ma la funzione ritorna subito
 - con MS_SYNC invece la funzione ritorna solo dopo l'effettiva scrittura
 - specificando anche MS_INVALIDATE forziamo tutti gli altri processi che mappano quel file a rileggerlo dal disco

La funzione msync

```
int msync(void *start, size_t length,  
          int flags);
```

- ➡ fa un flush esplicito dei dati in memoria nel file
- ➡ altrimenti, il processo non ha una garanzia che siano scritti fino alla chiamata di munmap
- ➡ flag:
 - con MS_ASYNC accodiamo l'operazione nel kernel, ma la funzione ritorna subito
 - con MS_SYNC invece la funzione ritorna solo dopo l'effettiva scrittura
 - specificando anche MS_INVALIDATE forziamo tutti gli altri processi che mappano quel file a rileggerlo dal disco

Perché usare mmap?

- ➡ mmap può permettere di semplificare la gestione dei file, perché l'I/O è nascosto e quindi evitiamo di usare le operazioni di read, write, lseek
 - non può essere usata con tutti i tipi di file: per esempio non con socket o con terminali
- ➡ ci consente di realizzare una memoria condivisa fra processi non correlati

Perché usare mmap?

- ➡ mmap può permettere di semplificare la gestione dei file, perché l'I/O è nascosto e quindi evitiamo di usare le operazioni di read, write, lseek
 - non può essere usata con tutti i tipi di file: per esempio non con socket o con terminali
- ➡ ci consente di realizzare una memoria condivisa fra processi non correlati

Esempio: uso di un numero di sequenza con mmap

incr2.c

- ➡ in questo esempio, apriamo un file qualsiasi, e usiamo mmap su di esso
- ➡ i mapping sono mantenuti anche nei child: quindi i cambi sono visibili reciprocamente
- ➡ scriviamo il valore del contatore direttamente nella memoria fornitaci
- ➡ le due copie di *ptr sono in sincronia fra i due processi
- ➡ proviamo ad esaminare il contenuto del file prodotto!

Esempio: con semaforo condiviso

incr3.c

- ➔ in questo esempio, anche il semaforo è nella memoria condivisa
- ➔ così facendo, possiamo usare un semaforo non dotato di nome, condividendolo fra due processi

Memoria condivisa Posix

- ➡ con altre tecniche, possiamo condividere memoria anche fra processi non correlati
- ➡ per la memoria da condividere con mmap, possiamo usare sia dei file, che quello che possiamo ottenere da una chiamata a `shm_open`

La funzione shm_open

```
int shm_open(const char *name, int oflag,  
             mode_t mode);
```

- ➔ restituisce un file descriptor in caso di successo (possiamo usarlo per mmap)
- ➔ oflag deve contenere O_RDONLY oppure O_RDWR, opzionalmente O_CREAT, O_EXCL, O_TRUNC
- ➔ mode specifica i bit di autorizzazione, è usato solo con oflag

La funzione shm_unlink

```
int shm_unlink(const char *name);
```

- ➔ consente di rimuovere la memoria condivisa
- ➔ come al solito, l'operazione avrà luogo solamente quando la memoria sarà stata chiusa da tutti i processi che la usano
- ➔ prima della deallocazione, comunque, nessun altro potrà riaprire la stessa memoria

Esempi

⇒ piccoli esempi:

- shmcreate
- shmopen
- shmunlink
- shmwrite (scrive un pattern fisso)
- shmread

Uso di un contatore con la memoria condivisa

client

server

- ➔ `export PX_IPC_NAME=/` (per impostare la locazione delle IPC)
- ➔ `b> server1 mm1 sem1`
- ➔ `a> client1 mm1 sem1`