

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

IPC: semafori

Claudio Vicari

Caratteristiche comuni delle IPC

- ➔ Questi tre tipi di IPC sono denominate IPC POSIX:
 - semafori
 - code di messaggi
 - memoria condivisa
- ➔ sono dotate di alcune proprietà in comune nel modo in cui vengono manipolate:
 - hanno una funzione *open con la possibilità di specificare un pathname e dei permessi
 - hanno una funzione *unlink per rimuovere la struttura
 - hanno funzioni per manipolare le strutture

Semafori POSIX

- ➔ un semaforo è un meccanismo utilizzato per la sincronizzazione fra processi (o thread) differenti
- ➔ i semafori sono dotati di un valore, che si può leggere e modificare
- ➔ si possono effettuare tre operazioni fondamentali:
 - creare un semaforo
 - aspettare (*wait*) un semaforo: l'esecuzione si blocca finché il semaforo è minore o uguale a 0, dopodiché viene decrementato
 - fare *post* verso un semaforo: questa operazione corrisponde ad incrementarne il valore, svegliando anche eventuali processi in stato di wait

Esempio: problema produttore/consumatore

➡ produttore:

```
semaforo get = 1  
semaforo put = 0
```

```
for(;;) {  
    sem_wait(&put);  
    scrivi in un buffer;  
    sem_post(&get);  
}
```

➡ consumatore:

```
for(;;) {  
    sem_wait(&get);  
    leggi il buffer;  
    sem_post(&put);  
}
```

- Il consumatore aspetta che il produttore abbia scritto
- Il produttore aspetta che il consumatore abbia letto

Semafori e lock

- ⇒ i semafori quindi sono utilizzati in modo simile ai lock, però sono più flessibili:
 - un lock può essere rilasciato solo dal processo che lo ha preso, mentre chiunque può effettuare l'operazione di *post* in un semaforo
 - un semaforo ha anche uno stato associato, che indica quanti processi hanno fatto post

Tipi di semafori

- ➔ i semafori possono essere creati con nome o senza nome
 - i semafori con nome possono essere utilizzati da processi del tutto indipendenti
 - i semafori senza nome possono essere usati solo da processi (o thread) che hanno un parent in comune
- ➔ le operazioni su questi tipi di semafori sono identiche, solo le funzioni di creazione e distruzione sono diverse

Semafori con nome: la funzione sem_open

```
#include <semaphore.h>
sem_t *sem_open(const char *name, int oflag,
    ...);
```

- ➔ possiamo specificare il nome del semaforo
- ➔ se `oflag==O_CREAT`, allora il semaforo viene creato, altrimenti (`oflag==0`) si apre un semaforo già esistente
- ➔ opzionalmente, come terzo argomento si può specificare un modo (come per `open`)
- ➔ come quarto argomento si può specificare un valore iniziale

Semafori con nome: la funzione `sem_close`

```
int sem_close( sem_t * sem );  
int sem_unlink( const char * name );
```

- ➔ `sem_close` chiude il semaforo
- ➔ questa operazione avviene comunque al termine del processo
- ➔ il semaforo però non viene rimosso dal sistema (un altro processo può ancora accederlo)
- ➔ `sem_unlink`, invece, è in grado di rimuovere il semaforo, ma solo nel caso in cui non sia aperto da nessuno

Semafori: funzioni di attesa

```
int sem_wait(sem_t * sem);  
int sem_trywait(sem_t * sem);
```

- ➔ `sem_wait` controlla il valore del semaforo
 - se è maggiore di 0, ritorna immediatamente e lo decrementa
 - altrimenti, mette il thread corrente in sleep, attendendo che il semaforo abbia un valore positivo prima di decrementarlo e uscire
- ➔ `sem_trywait` non si blocca mai, ma ritorna `EAGAIN` se il semaforo vale 0

Semafori: funzioni di attesa

```
int sem_post(sem_t * sem);  
int sem_getvalue(sem_t * sem, int * sval);
```

- ➔ `sem_post` incrementa il valore del semaforo e sveglia i thread in attesa
- ➔ `sem_getvalue` pone in `sval` il valore attuale del semaforo

Semafori: esempio

```
main() {
    sem_t sem1;
    int pid;
    int ret;

    ret = sem_init( &sem1, 0, 0 );
    if( ret == -1 ) {
        puts("error in semaphore creation");
        exit(0);
    }

    while( -1 ) {
        int ret2;
        sem_getvalue( &sem1, &ret2 );
        printf("getvalue=%d\n", ret2 );
        sem_post( &sem1 );
        sleep(1);
    }
}
```

Semafori: altri esempi

- ⇒ esempi presi da Unix Network Programming 2nd edition
- ⇒ provare a creare semafori:
 - `semcreate` semaforo
- ⇒ aspettare un valore:
 - `semwait` semaforo
- ⇒ fare post:
 - `sempost` semafor

Semafori senza nome: costruzione e distruzione

```
int sem_init(sem_t *sem, int pshared,  
             unsigned int value);  
int sem_destroy (sem_t * sem);
```

- ➔ `sem_init` consente di specificare se il semaforo può essere condiviso fra processi
 - un semaforo con nome è sempre condiviso
- ➔ notare anche che utilizzano un puntatore ad una struttura di tipo `sem_t`, che quindi deve essere allocata esplicitamente

Semafori senza nome

```
sem_t mysem;  
Sem_init(&mysem, 1, 0);  
  
if( fork==0 ) { //child  
    sem_post(mysem);  
} else {  
    sem_wait(&mysem);  
}
```

- ➔ questo codice non funziona come ci aspettiamo!
Infatti, mysem non risiede nella memoria condivisa, quindi le due copie di mysem si riferiscono a semafori differenti

Limiti per i semafori

- ➔ Posix definisce
 - SEM_NSEMS_MAX per il massimo numero di semafori aperti da un singolo processo (non meno di 256)
 - SEM_VALUE_MAX per il massimo valore permesso in un semaforo (almeno 32767)
 - sono definite in <unistd.h>
 - possono essere esaminate con la funzione sysconf