Pietro Pino (pepy91)

GESTIONE DEI PROCESSI E DEI THREADS IN UN SISTEMA LINUX (PARTE 4)

16 April 2015

Salve amici di Electroyou. Poiché in questi mesi sono stato parecchio impegnato dall'università, tra esami e professori, non ho potuto concludere la serie di articoli riguardante l'argomento specificato dal titolo. Finalmente oggi ho trovato un po' di tempo da dedicare a questo sito meraviglioso, che, ripeto, tanto mi ha aiutato e spero continuerà a farlo. Buona lettura!

I SEMAFORI GENERALIZZATI

Se i thread lavorano troppo velocemente, la coda dei processi sarà vuota e i thread usciranno. Se nuovi processi vengono messi in coda più tardi, non ci saranno thread per trattarli.Abbiamo bisogno dunque di un meccanismo per bloccare i thread quando la coda è vuota finché nuovi processi sono disponibili. I ***semafori*** sono la soluzione. Un *semaforo* è un contatore che può essere usato per sincronizzare più thread. Cosi come per un mutex, GNU/Linux garantisce che verificare o modificare il valore di un semaforo può essere fatto senza pericolo, senza creare cioè una race condition.

Ogni semaforo ha un **valore contatore**, che è un intero non negativo. Un semaforo supporta due operazioni di base:

1. un'operazione **wait** decrementa il valore del semaforo di 1. Se il valore è già zero, si sospende l'operazione finché il valore del semaforo diventa positivo. Quando ciò avviene, il valore viene decrementato di 1 e l'operazione *wait* ritorna;
2. un'operazione **post** incrementa il valore del semaforo di 1. Se il semaforo è a zero e altri thread sono bloccati nell'operazione wait su quel semaforo, uno di quei thread (a caso) viene sbloccato e la sua operazione wait termina (riportando il valore del semaforo a zero).

Per usare i semafori, dobbiamo includere l' header file **<semaphore.h>**. Un semaforo è rappresentato da una variabile di tipo **sem_t**. Prima di usarlo, bisogna inizializzarla con questa chiamata:

```
int sem_init (sem_t *sem, int type, int val);
```

- Il primo parametro è un puntatore alla variabile sem_t;
- Il secondo parametro deve essere 0 (GNU/Linux non supporta condivisione tra processi);
- Il terzo parametro è il valore iniziale del semaforo.

Se il semaforo non serve più, è meglio deallocarlo e liberare le risorse ad esso associate con la chiamata

```
int sem_destroy (sem_t *sem);
```

Per aspettare su un semaforo, utilizziamo la chiamata *wait*:

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

Per l'operazione *post*,invece, utilizziamo la chiamata:

```
int sem_post (sem_t *sem);
```

GNU/Linux fornisce una funzione per ripristinare il valore corrente di un semaforo:

```
int sem_getvalue (sem_t *sem, int *value);
```

dove il secondo argomento è il valore nella variabile puntatore a int.

Ritornando al nostro esempio della coda dei processi, possiamo usare un semaforo per contare il numero di processi in attesa nella coda. Vediamo il codice corrispondente:

```
CODA DEI PROCESSI CONTROLLATA DA UN SEMAFORO

#include <malloc.h>

#include <pthread.h>

#include <semaphore.h>

struct job {

       struct job *next;
}

/* Una lista concatenata per processi in sospeso */

struct job *job_queue;


/* Un mutex che protegge job_queue */

pthread_mutex_t job_queue_mutex;

pthread_mutex_init (&job_queue_mutex, NULL);
```

```
/* Un semaforo che conta il numero di processi nella coda */
sem_t job_queue_count;

/* Compie una volta l'inizializzazione della coda dei processi */
void initialize_job_queue () {
        /* La coda è inizialmente vuota*/
        job_queue = NULL;

        /* Inizializza il semaforo che conta i processi nella coda. Il su
        sem_init (&job_queue_count, 0, 0);
        }
 /* Trattamento processi in coda finché la coda è vuota */
 void *thread_function (void *arg){
        while (1){
                struct job *next_job;
                /* Operazione wait sul semaforo della coda dei processi
                indicato dal fatto che la coda non è vuota, decrementa
                finché viene accodato un nuovo processo */
                sem_wait (& job_queue_count);

                /* Chiude il mutex sulla coda dei processi */
                phtread_mutex_lock (&job_queue_mutex);

                /* Grazie al semaforo, sappiamo che la coda non è vuota
```

```
                next_job = job_queue;

                /* Rimuove questo processo dalla lista */
                job_queue = job_queue -> next;

                /* Apre il mutex sulla coda dei processi */
                pthread_mutex_unlock (&job_queue_mutex);

                /* Esegue il lavoro */
                process_job (next_job);

                /* Pulisce */
                free (next_job);
                }
        return NULL;
    }

/* Aggiunge un nuovo processo alla coda */
void enqueue_job (/* Passiamo come argomento i dati specifici del processo */)
        struct job *new_job;
        /* Alloca un nuovo processo */
        new_job = (struct job *) malloc (sizeof (struct job));

        /* Imposta qui gli altri campi della struttura del processo */
```

```
                    /* Chiude il mutex sulla coda dei processi prima di accedervi */

                    pthread_mutex_lock (&job_queue_mutex);


                    /* Inserisce il nuovo processo alla testa della coda */

                    new_job -> next = job_queue;

                    job_queue = new_job;


                    /* Operazione post al semaforo per indicare che un altro processo
                    aspettando sul semaforo, uno si sbloccherà cosi potrà trattare il

                    sem_post(&job_queue_count);


                    /* Apre il mutex della coda dei processi */

                    pthread_mutex_unlock (&job_queue_mutex);
            }
```

Prima di prendere un processo dalla coda, ogni thread aspetta prima sul semaforo. Se il valore del semaforo è zero, indicato con la coda vuota, il thread sospenderà finché il valore del semaforo diventa positivo, cioè quando un nuovo processo è in coda. La funzione **enqueue_job** aggiunge un processo alla coda. Cosi come nel **thread_function**, bisogna chiudere il mutex della coda prima di modificarne il valore. Dopo averlo aggiunto, viene postato al semaforo, indicando che c'è un nuovo processo, altrimenti i thread si bloccano su **sem_wait**.

Si conclude cosi questa serie di articoli. Spero che qualcuno possa trovarli utili o interessanti. Vi ringrazio e cercherò in qualche modo di trovare il tempo per inserire altre nozioni e argomenti che sto trattando nel mio corso di Laurea.

Estratto da "http://www.electroyou.it/mediawiki/index.php?title=UsersPages:Pepy91:gestione-dei-processi-e-dei-threads-in-un-sistema-linux-parte-4"