

Laboratorio su Programmazione Concorrente in C Posix Thread

Dall'ottava lezione di laboratorio in avanti

NOTA BENE:

1) Usare il comando **man** per avere informazioni sull'uso di una specifica funzione di libreria standard del linguaggio C

Ad esempio, per ottenere informazioni sull'uso della funzione **strerror_r**:

man strerror_r

man usleep

2) Ricordo inoltre che la funzione **pthread_create** restituisce un valore intero. In caso di errore tale risultato è diverso da zero e contiene un codice numerico che identifica l'errore.

Per avere una stringa che descrive in forma testuale tale errore occorre passare il risultato come argomento alla funzione **strerror_r**

3) PONETE PARTICOLARE ATTENZIONE AL MODO IN CUI TERMINATE L'ESECUZIONE DEL main
LA FUNZIONE **exit()** TERMINA IL PROCESSO E TUTTI I SUOI THREAD.

LA FUNZIONE **return()** CHIAMATA NEL MAIN TERMINA IL PROCESSO E TUTTI I SUOI THREAD.

LA FUNZIONE **pthread_exit()** CHIAMATA IN UN THREAD (O ANCHE NEL main) TERMINA IL THREAD MA NON IL PROCESSO, A MENO CHE QUEL THREAD NON FOSSE L'ULTIMO.

Lezione 8 in laboratorio usare pthread **tecnica di passaggio parametri, controllo d'errore e join**

☀ pthread, il disastro è dietro l'angolo



identificazione dell'errore in forma testuale (1)

PREMESSA all' Esercizio

usare_strerror_r

La funzione: `int strerror_r (int errnum, char *buf, size_t buflen);`

può essere utilizzata se:

a) si definisce il simbolo `_POSIX_C_SOURCE` con valore `>= 200112L`

b) poi si include il file `string.h`

La funzione vuole come primo argomento il codice di errore numerico di cui si cerca la stringa di caratteri che descrive l'errore.

Come secondo argomento l'indirizzo di inizio di una area di memoria in cui la funzione scrive la stringa di caratteri descrittiva dell'errore.

Come terzo argomento la dimensione in byte dell'area di memoria passata come secondo argomento.

IL risultato restituito è zero se tutto va a buon fine, ed in questo caso **nell'area di memoria passata come secondo argomento si trova la stringa null-terminata con il testo che descrive l'errore specificato dal codice numerico passato come primo argomento.**

identificazione dell'errore in forma testuale (2)

Esercizio **usare_strerror_r**

Scrivere un programma in cui il main genera un errore e invoca la funzione `strerror_r` per ottenere e poi stampare a video la stringa che descrive l'errore.

COME PRIMA ATTIVITA' PERO' SCRIVETE IL Makefile PER COMPILARE E LINKARE IL PROGRAMMA PRIMA DI COMINCIARE A SCRIVERE IL PROGRAMMA.

Dettagli:

Per generare l'errore, il main cerca di fare la `pthread_join` con se stesso, cioè chiama la `pthread_join` specificando come primo argomento il proprio identificatore di thread (suggerimento: usare la funzione `pthread_self()` per ottenere l'identificatore del thread main).

Verificare che la `pthread_join` così chiamata genera un errore e quindi usare la funzione `strerror_r` per ottenere la stringa descrittiva dell'errore.

Stampare a video l'errore, nella forma di intero e nella forma di stringa.

soluzione

in

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/usare_strerror_r.tgz

identificazione dell'errore in forma testuale (3)

SOLUZIONE Esercizio

usare_strerror_r

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
#include <string.h>          /* per strerror_r */

int main() {
    pthread_t tid;    int rc;    void *ptr;

    /* prendo il mio stesso identificatore di thread */
    tid = pthread_self();

    /* tento join con me stesso ma provoca errore perché mi bloccherei aspettando la mia morte */
    rc = pthread_join( tid , &ptr );
    if (rc){
        char msg[100]; int ret;
        ret=strerror_r(rc,msg,100);
        if( ret != 0 ) { /* anche la funzione strerror_r potrebbe causare errore */
            printf("strerror failed: errore %i \n", ret );    fflush(stdout);
        }
        /* stampo il messaggio di errore provocato dalla join */
        printf("pthread_join failed: %i %s\n", rc, msg );
    }
    return(0);
}
```

identificazione dell'errore in forma testuale (4)

SOLUZIONE **Esercizio** **usare_strerror_r**

MAKEFILE

CFLAGS=-ansi -Wpedantic -Wall -D_REENTRANT -D_THREAD_SAFE -D_POSIX_C_SOURCE=200112L
LIBRARIES=-lpthread

all: usare_strerror_r.exe

usare_strerror_r.exe: usare_strerror_r.o

gcc -o usare_strerror_r.exe usare_strerror_r.o \${LIBRARIES}

usare_strerror_r.o: usare_strerror_r.c

gcc \${CFLAGS} -c usare_strerror_r.c

.PHONY: clean

clean:

rm -f usare_strerror_r.o usare_strerror_r.exe

Esercizio 000: es000_infiniti_thread_senza_join.c

Pthread

Partendo dall'esempio delle dispense **con_trucco.c** (reperibile a questo indirizzo http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/PTHREAD/CON_TRUCCO/con_trucco.c e

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/PTHREAD/CON_TRUCCO/Makefile)

e dall'esempio d'uso della pthread_join visto a lezione, costruire un programma in cui il main crea infiniti thread, a ciascuno dei quali passa un indice crescente, senza fare mai la pthread_join.

Ciascun thread stampa l'indice passatogli e termina se stesso.

Inserire il controllo sugli errori restituiti da ciascuna funzione di libreria, stampando a video la stringa che descrive il tipo di errore accaduto. A tal fine usare la funzione strerror_r().

Verificare cosa **accade** in esecuzione. **A lungo andare, dovrebbe capitare un errore, dovuto al mancato rilascio degli identificatori dei pthread.**

Iniziate l'esercizio scrivendo il Makefile.

soluzioni in http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es000_infiniti_thread_senza_join.tgz

Esercizio 00: es00_infiniti_thread_con_join.c

Come nell'esercizio precedente, ma qui il main crea 1000 thread e poi fa la join per quei 1000 thread, poi crea altri 1000 thread e fa la join per quei 1000, e così via all'infinito.

Iniziate l'esercizio scrivendo il Makefile.

soluzioni in http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es00_infiniti_thread_con_join.tgz

Pthread

Esercizio 0: es0_premortem.c

Il main è il primo thread, ha indice 0, aspetta 1000 microsecondi, crea un thread figlio passandogli (anche) il valore dell'indice incrementato di uno, e poi termina.

Ciascun altro thread deve: stampare il proprio indice, aspettare 1000 microsecondi, creare un altro thread passandogli (anche) il valore dell'indice incrementato di uno, aspettare che il thread che lo ha creato termini, e infine terminare esso stesso.

I thread non possono usare variabili globali per passarsi informazioni.

Se necessario, i thread possono passare al loro figlio anche altre informazioni.

Usare la funzione **usleep()** per realizzare l'attesa richiesta.

Nota bene: Per usare la funzione **usleep()** occorre definire il simbolo **_DEFAULT_SOURCE** prima di includere il file **unistd.h** che ne definisce il prototipo.

Se preferite, invece della funzione **usleep()** potete usare la funzione **nanosleep()**.

Iniziate l'esercizio scrivendo il Makefile.

soluzione con **usleep** in

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es0_premortem.tgz

soluzione con **nanosleep** in

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es0_premortem_nano.tgz

Pthread Complicato

Esercizio 0000: es0000_strutture.c

Un main lancia in esecuzione 4 thread passando a ciascuno di essi una struttura dati chiamata Struttura che è formata da 3 campi:

un intero N che dice al pthread quanti altri thread deve creare; una stringa Str di 100 caratteri; un intero Indice con un valore compreso tra 0 ed N-1, diverso per ciascun thread creato da uno stesso thread.

Ciascun thread, iniziando la sua esecuzione, aspetta 1 secondo, poi legge il contenuto della struttura dati ricevuta, in particolare il valore di N. Sia M il valore contenuto in N.

Se $M > 1$ allora il thread lancia in esecuzione M-1 thread passando a ciascuno una copia della struttura dati che nel campo N contiene il valore M-1, e nel campo Indice contiene un valore compreso tra 0 ed (M-1)-1, diverso per ciascun thread creato dal nostro thread.

Ciascun thread deve restituire una struttura dati simile a quella che è stata avuta come argomento. Nel campo stringa Str della struttura deve essere collocato in formato testuale il valore di N ricevuto poi uno spazio e poi il valore di Indice ricevuto.

Prima di terminare, ciascun thread deve aspettare la terminazione di ciascun thread che lui ha creato e stampare a video la stringa Str ricevuta da ciascuno di quei thread.

Iniziate l'esercizio scrivendo il Makefile.

Soluzioni

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es0000_strutture.tgz

Lezione 9 in laboratorio

usare pthread, **mutua esclusione**

- ✿ pthread, mutua esclusione
- ✿ Gli esercizi 1 e 2 mettono in evidenza la possibilità e la convenienza, dal punto di vista delle prestazioni, di eseguire in parallelo diversi tipi di operazioni che, agendo su gruppi distinti di variabili globali, possono essere protette in mutua esclusione da variabili mutex diverse.
- ✿ Gli esercizi 3 e 3bis mettono in evidenza che, qualora debba essere preso possesso di un gruppo di risorse, una alla volta, e quindi non atomicamente, occorre rispettare lo stesso ordine nel prendere le risorse, per evitare deadlock.

Pthread – Programmazione Concorrente

Esercizio 1: `es1_banche` **mutua esclusione semplice**

Un programma su un server gestisce le operazioni di cassa su 3 diverse banche chiamate B1, B2 e B3. Per ciascuna banca Bj esiste una variabile globale Tj che contiene il totale del denaro posseduto dalla banca Bj. Per ciascuna banca Bj esiste una variabile globale Nj che contiene il numero di operazioni (depositi e prelievi) fatte della banca Bj. Per ciascuna banca esistono 5 pthread Depositi e 4 pthread Prelievi.

Ciascuno di questi pthread esegue un loop. Il loop e' composto da una parte iniziale **fuori dalla** mutua esclusione e da una sezione critica **in** mutua esclusione.

La parte iniziale fuori dalla mutua esclusione realizza l'attesa di 1 secondo. Nella parte di sezione critica si modifica il totale della banca di +10 per i Depositi e di -9 per i Prelievi. Prima di uscire dalla sezione critica si deve attendere 2/10 di secondo.

Esiste poi un thread BancaDiItalia che esegue un loop in cui cerca di stampare a video la somma dei totali delle 3 banche e la somma del numero di operazioni eseguite. Per fare le somme occorre che nessun Deposito e nessun Prelievo stia eseguendo la propria sezione critica. Dopo avere stampato il necessario, il thread BancaDiItalia aspetta 1 sec. poi esce dalla mutua esclusione e attende 30 secondi prima di ricominciare il loop.

Implementare il tutto usando una sola variabile di tipo pthread_mutex_t. Non si possono usare variabili di tipo pthread_cond_t. **Iniziare scrivere il Makefile**

Suggerimento: partire dall'esempio **mutex.c** ed usare un vettore di 3 variabili T ed un vettore di 3 variabili N. Passare a ciascun thread Depositi e Prelievi l'indice della banca a cui si riferisce.

Esercizio 2: es2_banche Mutua esclusione

Come nell'esercizio precedente, ma usando una variabile `pthread_mutex` diversa per ciascuna banca.

La BancaDiItalia dovrà ottenere la mutua esclusione su tutte e tre le mutex prima di poter svolgere le proprie operazioni di somma.

Iniziate l'esercizio scrivendo il Makefile.

Confrontare il numero di operazioni bancarie totali che si riescono a effettuare in 2 minuti, nei due esercizi (questo ed i precedente).
Verificare se ci sono differenze sostanziali.
Capire perchè ci sono differenze.

Soluzione 1: es1_banche

vedere http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es1_banche.tgz

Soluzione 2: es2_banche

vedere http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es2_banche.tgz

Esercizio 3: es3_fachiri

Esistono 2 fachiri e un vettore di 10 spade.

L'accesso a ciascuna spada è protetto da **una variabile mutex per ciascuna spada**.

I 2 fachiri svolgono continuamente il loro esercizio che consiste nel: 1) prendere uno dopo l'altra 10 spade. 2) Dopo avere preso tutte le spade, trafiggersi con le spade stampando poi a video un grido di dolore. 3) Rimettere le spade al loro posto. 4) ricominciare da capo.

Entrambi i fachiri cominciano a prendere le spade cominciando dalla spada di indice minore.

Modellare il sistema mediante un thread per ciascun fachiro.

Iniziate l'esercizio scrivendo il Makefile.

soluzione in http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es3_fachiri.tgz

Esercizio 3bis: es3bis_Deadlock_fachiri

Come nell'esercizio precedente. ma stavolta un fachiro comincia a prendere le spade cominciando dalla spada di indice minore, mentre l'altro comincia a prendere le spade cominciando dalla spada di indice maggiore.

Iniziate l'esercizio scrivendo il Makefile.

Verificare CHE, con queste ipotesi, il sistema NON riesce a svolgere correttamente .

Soluzione che verifica che il sistema non può funzionare

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es3bis_Deadlock_fachiri.tgz