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 o **usleep**:

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

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 ma il processo continua a eseguire.

Ciascun altro thread deve: stampare il proprio indice, aspettare 1000 microsecondi, poi

se l'indice che gli è stato passato è minore di 100 allora crea un altro thread passandogli (**anche**) il valore dell'indice incrementato di uno, aspetta che il proprio thread padre (quello che lo ha creato termini, e infine termina esso stesso. Se indice è maggiore o uguale a 100 il thread termina senza creare un figlio.

I thread non possono usare variabili globali per passarsi informazioni.

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

NB: per attendere la terminazione del pthread padre, il pthread figlio deve conoscere il thread identifier del pthread padre. Occorre passarglielo tra gli argomenti al momento della creazione.

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().

In caso di errore stampare il codice numerico che descrive l'errore. Non occorre stampare la stringa che descrive l'errore.

Usate come base l'esempio https://www.cs.unibo.it/~ghini/didattica/sistemioperativi/PTHREAD/JOIN_DETACHED/joinable.c

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

Esercizio 0: es0_premortem.c SUGGERIMENTO

Quando un thread crea un altro thread, gli passa una struttura dati contenente:

l'indice che il thread creato deve usare,

il thread identifier del thread padre per poter fare la pthread-join.

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

identificazione dell'errore in forma testuale (1)

Esercizio 1 usare_strerror_r

Vi passo un programma, in linguaggio C, in cui il main chiama in modo sbagliato una funzione dei pthread e così genera un errore e poi invoca la funzione strerror_r per ottenere e poi stampare a video la stringa che descrive lì'errore accaduto.

codice sorgente e Makefile da completare in

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

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/STRERROR_R/Makefile.DACOMPLETARE

VOI DOVETE MODIFICARE il codice ed il Makefile PER COMPILARE E LINKARE IL PROGRAMMA. IL PROBLEMA è la funzione strerror_r che per essere usata richiede di includere un header file nel main e richiede di definire una particolare versione del simbolo _POSIX_C_SOURCE

Usare **man strerror_r** per capire che include file usare e che valore definire per quel simbolo. Oppure guardare le dispense delle lezioni della settimana scorsa.

Soluzione in

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

NB: Dettagli su come viene generato l'errore nel main:

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 (usa la funzione pthread_self() per ottenere l'identificatore del thread main.

La pthread_join così chiamata genera un errore e quindi si controlla il risultato restituito e si usa la funzione strerror_r per ottenere la stringa descrittiva dell'errore.

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

L'esecuzione dell'eseguibile dovrebbe dare il seguente output:

pthread_join failed: 35 Resource deadlock avoided

identificazione dell'errore in forma testuale (2)

DA COMPLETARE E SCRIVERE MAKEFILE

Esercizio usare_strerror_r

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <pthread.h>
/*  AGGIUNGERE  QUI INCLUDE 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 (3) Spiegazione soluzione

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 (4)
Spiegazione soluzione

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
```

identificazione dell'errore in forma testuale (5)
Spiegazione soluzione

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);
}
```

Esercizio 000: es000_infiniti_thread_senza_join.c

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

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.

es2_banche PIU' VARIABILI PER MUTUA ESCLUSIONE

Un programma su un server gestisce le operazioni di cassa su 2 diverse banche chiamate B1 e B2. 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 3 pthread Depositi e 2 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 4/10 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 e rifare le somme.

Usate una variabile di tipo pthread_mutex_t **per ciascuna banca**, in un vettore **mutex**.

Ciascun thread Deposito o Prelievo, per effettuare la modifica dovrà ottenere la mutua esclusione sulla mutex della banca su cui opera. La BancaDiItalia dovrà ottenere la mutua esclusione su tutte e tre le mutex prima di poter svolgere le proprie operazioni di somma.

Suggerimento: Passare a ciascun thread Depositi e Prelievi l'indice della banca a cui si riferisce.

Completare il codice disponibile qui:

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/BANCHE2/es2_banche.COMPLETARE.tgz

Soluzioni in:

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/BANCHE2/es2_banche.SOLUZIONE.tgz

es1_banche Mutua esclusione semplice

Come nell'esercizio precedente es2_banche, ma usando una sola unica variabile pthread_mutex_t per proteggere tutte le variabili condivise.

Completare jl codice contenuto qui:

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/BANCHE1/es1_banche.COMPLETARE.tgz

Soluzione 1: es1_banche

vedere
http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/BANCHE1/es1_banche.SOLUZIONE.tgz

Confrontare il numero di operazioni bancarie totali che si riescono a effettuare in 5 minuti, nei due esercizi (questo ed il precedente es2_banche).
Verificare se ci sono differenze sostanziali.
Capire perchè ci sono differenze.

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