Laboratorio su Programmazione Concorrente in C Posix Thread

Settima lezione di laboratorio

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, mutua esclusione

- pthread, il disastro è dietro l'angolo

Pthread facile

Esercizio 00000: es00000_facile

Un main lancia in esecuzione 10 thread passando a ciascuno di essi un valore di tipo double inizializzato con un valore intero casuale.

Fatto questo il main termina lasciando i suoi thread in esecuzione.

Ciascun thread aspetta 5 secondi e stampa a video il valore ricevuto.

Infine il thread termina senza necessità di restituire un particolare risultato.

soluzioni in http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es00000_facile.tgz

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 RIMASTO.

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.

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.

Iniziate l'esercizio scrivendo il Makefile.

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

Esercizio 5: es5_dealloca_albero puntatori e alberi binari.

Una struttura ad albero binario e' implementata con la struttura dati NODO seguente:

typedef struct s_NODO{ int key; double x; s_NODO *destra; s_NODO *sinistra; } NODO;

Ogni nodo dell'albero può avere due figli puntati dai puntatori destra e sinistra, che assumono valore NULL se il figlio corrispondente non esiste. L'albero viene costruito allocando dinamicamente ciascun nodo chiamando la funzione malloc. L'albero può essere vuoto.

Sia root la radice dell'albero (NODO *root), root e' una variabile locale del main, root assume valore NULL quando l'albero è vuoto.

Nel main l'albero viene allocato e costruito da una funzione costruisci_albero().

Quando l'albero non serve più, la memoria allocata per ciascun nodo deve essere rilasciata mediante la funzione di libreria free(void*) passandole come parametro l'indirizzo dell'area di memoria da deallocare.

Implementare la funzione **void dealloca_albero(NODO* *ppnodo);** che verrà chiamata ad. es. nel modo seguente:

```
void main(){            NODO *root;
        costruisci_albero( & root );
        usa_albero( root );
        dealloca_albero( & root );
}
```

La funzione dealloca_albero deve deallocare la memoria allocata per ciascuno dei nodi dell'albero, e infine deve porre a NULL il puntatore root di cui viene passato l'indirizzo come parametro della funzione, affinche' l'albero appaia vuoto.

Soluzione 5: es5_dealloca_albero

soluzione in
http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es5_dealloca_albero.tgz