Laboratorio su Programmazione Concorrente in C

Problemi classici e derivati

Dalla Nona lezione di laboratorio in avanti …

NOTA BENE: E INSISTO !!!!!

Usare il comando **man** nomefunzionedilibreria per ottenere informazioni sull'uso di una specifica funzione di libreria standard del linguaggio C
Ad esempio, per ottenere informazioni sull'uso della funzione **sqrt**:

man sprintf

Lezione 8 in laboratorio: concorrenza. sinc. circolare, fornaio, prod. cons.

- non saranno tutte rose e fiori.

Perciò testa bassa e pedalare ...

Esercizio 4: ripartire_in_ordine

Utilizzare wget per scaricare l'esempio :

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/PTHREAD/MUTEX_CONDVAR/CondVarSignal.c

Modificare l'esempio in modo che:

i 5 thread si fermino tutti nella funzione SynchPoint (punto di sincronizzazione) e,

una volta che sono tutti arrivati, ripartano ed escano nell'ordine in cui si sono fermati.

Suggerimento:

prima di incrementare il contatore synch_count, che conta il numero di thread entrati nella funzione SynchPoint, ciascun thread salva quel valore in una sua variabile locale.

Quel valore salvato è il turno di uscita dei thread .....

Soluzione 4: ripartire_in_ordine

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es4_ripartire_in_ordine.tgz

Esercizio 10: Sincronizzazione circolare tra 3 pthread

Un programma e' composto da tre thread T1 T2 e T3, creati da un main. Ciascuno dei tre thread itera all'infinito eseguendo una propria sequenza di operazioni Op1 Op2 e Op3. La sincronizzazione tra i thread deve ottenere che le operazioni dei tre thread eseguano in modo alternato, prima tutta Op1, poi tutta Op2, poi tutta Op3, poi ancora tutta Op1 e poi tutta Op2 e poi tutta Op3 e cosi' all'infinito. Chiariamo il concetto:

La prima iterazione deve essere svolta dal thread T1.

Al termine di ciascuna propria iterazione il thread T1 passa il controllo al thread T2 e si mette in attesa che il thread T3 gli restituisca il controllo. Quando ottiene il controllo da T1, il thread T2 esegue una iterazione e poi passa il controllo al thread T3. Quando ottiene il controllo da T2, il thread T3 esegue una iterazione e poi passa il controllo al thread T1 Quando ottiene il controllo, il thread T1 esegue una altra iterazione e poi passa nuovamente il controllo al thread T2. E cosi' via all'infinito

Per semplicita', partire dall'esempio con la variabile turno presentato a lezione eventualmente modificando il codice dell'esempio con le variabili faiwait nell'esempio:

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/PTHREAD/SINCRO_CIRCOLARE_1a1/circolare1a1.c

Due possibili soluzioni nelle slide successive, la prima qualche volta risveglia dei thread inutilmente, la seconda risveglia solo i thread di turno.

Non usate subito le soluzioni, prima sforzatevi di trovare voi una soluzione e provatela.

Traccia Soluzione Esercizio 10 con 1 cond (Sincronizzazione circolare a 3)

Nel main vengono inizializzate 1 var mutex, 1 var cond,.
La variabile turno assegna il turno e assume valore 0, 1 o 2 e viene inizializzata a 0.

LINK SOLUZIONE IN PAGINA SEGUENTE

```
pthread_mutex_t   mutex;          pthread_cond_t   cond;          int   turno=0;

        void *ThreadFunction(  void * arg  )  {
                intptr_t   mioindice  =  (intptr_t)arg;

                while(1) {
                        pthread_mutex_lock(&mutex);
                        while ( turno != mioindice   )  {
                                pthread_cond_wait ( &cond,  &mutex );
                                /* se non e' il mio turno sveglio qualcuno e ritorno in wait */
                                if  ( turno != mioindice    )
                                        pthread_cond_signal ( &cond );
                        }
                        /* SEZIONE CRITICA : fa qualcosa */
                        turno= (turno+1) %3;              /*  passo il turno al successivo  */
                        pthread_cond_signal (  &cond  );          /* risveglio qualcuno  */

                        pthread_mutex_unlock(&mutex);
                }
        }
        int main(  void )  {      intptr_t  i;              pthread_t   tid;
                pthread_mutex_init (&mutex, NULL); pthread_cond_init (&cond, NULL);
                 for( i=0;i < 3;i++)
                        pthread_create ( &tid , NULL, ThreadFunction,  (void *) i );
                pthread_exit(NULL);
                return(0);
        }
```

Soluzione Esercizio 10 con 1 sola condition variable

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es10_circolareA3_turn.tgz

Traccia Soluzione Esercizio 10 con 3 cond (Sincronizzazione circolare a 3)

Nel main vengono inizializzate 1 var mutex e 3 condition variables (una per ciascun thread) queste ultime contenute in un vettore denominato Vcond.

```
void *ThreadFunc (  void * arg  )  {
        while(1) {
                pthread_mutex_lock( &mutex );
                while ( turn != indice ) {
                        pthread_cond_wait( & (Vcond[indice] ) , &mutex );
                /* SEZIONE CRITICA : fa qualcosa */

                turn = (turn+1) % 3;
                pthread_cond_signal( & ( Vcond[turn] )   );
                pthread_mutex_unlock(&mutex);
        }
}

int main ()
{       pthread_t    th[NUMTHREADS]; int  rc; intptr_t  i;
        pthread_mutex_init(&mutex, NULL);
        for(i=0;i<NUMTHREADS;i++)   pthread_cond_init( & (Vcond[i]) , NULL);

        for(i=0;i<NUMTHREADS;i++)
                pthread_create(  &(th[i]), NULL, ThreadFunc, (void*) i );

         pthread_exit(NULL);
}
```

GUARDARE COMMENTO e LINK SOLUZIONE IN PAGINA SEGUENTE

NOTA SULLA SOLUZIONE Soluzione Esercizio 10 con 3 cond

In questa soluzione risveglio solo il thread giusto.

E' più comodo implementare le 3 condition variable
mettendole in un vettore,
invece che considerarle come tre variabili separate.

Si veda a tal proposito l'implementazione in:

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es10_circolareA3_vet3cond.tgz

Esercizio 7: fornaio con biglietto e scaffale a doppio accesso es7_fornaio2

- Ad un fornaio non piace stare al bancone a servire i clienti, perciò i clienti si servono da soli, accedendo ad uno scaffale con le ceste del pane, in cui però c'è posto **al massimo per due clienti insieme alla volta.**
- Inoltre al fornaio piace l'ordine, quindi **i clienti entrando nel negozio prendono un biglietto con un numero intero sequenziale crescente, e accedono allo scaffale nell'ordine stabilito dal numero del loro biglietto.**

- Prima di accedere al bancone del pane, perciò, ciascun cliente prende il biglietto contenente un numero che stabilisce il suo turno ed aspetta il suo turno.
- Un display indica il numero del prossimo cliente che può accedere allo scaffale.

- **Attenti a quando e come incrementare la variabile globale bigliettoSulDisplay.**

Implementare i thread che rappresentano i clienti, avvalendosi delle opportune strutture dati per la sincronizzazione. Scrivere un main che lancia 10 clienti.
Scrivere da subito il Makefile per compilare e linkare i sorgenti. La mancanza del Makefile viene considerato una mancanza grave.
Occorre inserire il controllo di errore nelle chiamate a funzione delle librerie dei pthread. In caso di errore grave, terminare il programma producendo un avviso a video.

soluzione esercizio 7

Soluzione Esercizio 7: fornaio con biglietto e scaffale a doppio accesso

es7_fornaio2

VEDERE

ttp://www.cs.unibo.it/~ghini/didattica/sistemioperativi/ESERCIZI/es7_fornaio2.tgz