ZZZ02 Esercizi Vari

Potete usare tutto il materiale, normalmente disponibile sulla pagina di sistemi operativi, che trovate tar-gzippato in questo archivio di 48 MB.

http://www.cs.unibo.it/~ghini/didattica/TREE4OS.tgz

Scaricatelo con wget e scompattatelo con tar xvzf TREE4OS.tgz

Se , ad esempio, vi trovate nella cartella /home/studente/
allora potete navigare tra i documenti per mezzo di un browser con open file:

file:///home/studente/TREE4OS/sistemioperativi/**sistemioperativi_index.html**

Rivedrete la consueta pagina web di sistemi operativi, ma tutta in locale.

Esercizi per preparazione alla prova pratica di laboratorio

1. eliminare warning
2. correggere pthread
3. programmazione concorrente "sincronizzazione alla terminazione"
4. processi padri e figli
5. script bash
6. processi con condivisione di segmenti e sincronizzazione
7. script bash con uso text utils
8. script bash con espressioni condizionali
9. dimensione struttura
10. programmazione concorrente "Incrocio nebbioso"
11. programmazione concorrente "cani e biciclette"
12. programmazione concorrente "la setta"

Esercizio ZZZ02_01 - **codice a warning zero**

Correggere i warning

Scaricare i due files,

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/PTHREAD/ESERCIZICASA/ELIMINAWARNING01/eliminawarning01.c

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/PTHREAD/ESERCIZICASA/ELIMINAWARNING01/Makefile

Correggere i warning che si producono in compilazione e linking senza variare il funzionamento di fondo del programma.

Esercizio ZZZ02_02 - **correggere**

Correggere il funzionamento

Scaricare i due files,

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/PTHREAD/ESERCIZICASA/CORREGGI_01/loopinfinito1.c

http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/PTHREAD/ESERCIZICASA/CORREGGI_01/Makefile

Il programma esegue un loop infinito in cui vengono generati continuamente dei pthread, i quali stampano un valore e terminano.

Correggere eventuali malfunzionamenti senza variare il funzionamento di fondo del programma.

Esercizio ZZZ02_03 - **attendi terminazione**

Implementare sincronizzazione

Scaricare i due files,
http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/PTHREAD/ESERCIZICASA/ATTENDIFINE/attendi.c
http://www.cs.unibo.it/~ghini/didattica/sistemioperativi/PTHREAD/ESERCIZICASA/ATTENDIFINE/Makefile

il main crea alcuni thread.
Ciascun thread genera un'attesa di lunghezza casuale e poi termina.
Il main deve attendere con delle join la terminazione di tutti i thread,
facendo per prime le join dei thread che terminano per primi le sleep
e che quindi terminano se stessi per primi.
Ciascun thread, dopo avere terminato la propria sleep,
deve comunicare al main la propria intenzione di terminare,
mettendo il proprio thread identifier in una variabile globale **threadID_globale.**
Questa variabile globale deve essere protetta opportunamente.
Il main, man mano che termina le join, deve stampare il risultato intero restituito da
ciascun thread, che e' proprio il numero di secondi che il thread ha atteso.

Modificare il codice scaricato inserendo le strutture POSIX ed il codice necessario per sincronizzare opportunamente i diversi thread e il main.

Esercizio ZZZ02_04 - **processi padri e figli**

Implementare

Un processo padre deve creare un figlio ed attendere la terminazione del figlio.
Il processo figlio deve creare un figlio ed attendere la terminazione del proprio figlio.
Ciascun processo figlio deve creare un figlio e attendere la terminazione di quel figlio.
Quando viene creato, in totale, il DECIMO processo figlio, quel processo figlio termina senza creare nessun figlio e restituisce 1.
Ciascun processo, ad eccezione del DECIMO figlio, termina restituendo come exit code il valore restituito dal suo figlio, incrementato di 1.

Implementare in C il programma padre descritto.
Scrivere da subito il Makefile per generare il programma partendo dai sorgenti.
Se manca il Makefile oppure se il Makefile non funziona correttamente, non correggo l'esercizio e lo considero sbagliato !

Esercizio ZZZ02_05 - **script bash**

Implementare

Implementare uno script bash che prende dallo standard input i percorsi assoluti di alcuni file e trova il piu' recente di quei file, stampandone a video il nome ed il numero di righe.

Esercizio ZZZ02_06 - **processi con memoria condivisa**

Implementare

Un processo crea un segmento di memoria condiviso contenente tutto il necessario alla sincronizzazione dei suoi figli ed anche una variabile intera **Count** inizializzata a ZERO.

Il processo genera 10 figli e attende la terminazione dei primi 5 figli che terminano.

Ciascun processo figlio aggiunge UNO alla variabile intera.

Anche il processo padre aggiunge UNO alla variabile intera dopo avere visto la terminazione dei primi 5 figli.

L'ultimo processo che modifica la variabile Count deve anche eliminare il segmento di memoria condivisa stampando prima a video il valore di Count.

Scrivere da subito il Makefile per generare il programma partendo dai sorgenti.
Se manca il Makefile oppure se il Makefile non funziona correttamente, non correggo l'esercizio e lo considero sbagliato !

Esercizio ZZZ02_07 - **script bash**

Implementare

Implementare uno script bash che lancia in background 10 volte il comando

sleep 10

Dopo avere lanciato i comandi, e senza memorizzare i pid dei processi mentre li lancia, lo script deve

- usare il comando ps per ottenere i **pid** di tutti i processi **sleep** in esecuzione,
- e usare quei pid per killare i processi sleep.

Esercizio ZZZ02_08 - **script bash**

Implementare

Implementare uno (o piu') script bash che,

tra tutti i file contenuti nella directory (e sotto-directory) /usr/include/

e che hanno il nome che termina con .h

seleziona il file piu' recente.

Di quel file, deve essere stampato a video il nome ed il numero di righe.

Esercizio ZZZ02_09 - **dimensione struttura**

Implementare

Implementare un programma in C che stampa a video la dimensione in memoria della struttura seguente:

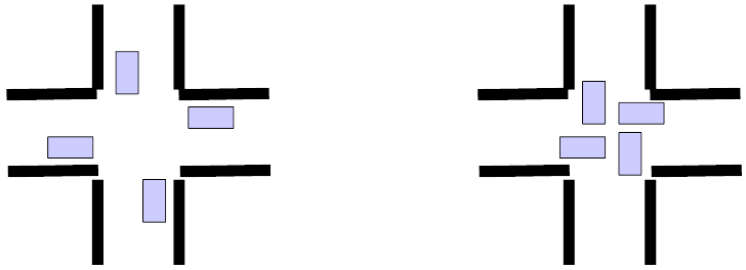
```
typedef struct struttura { uint32_t i;   uint8_t  c; uint32_t i2; } STRUTTURA;
```

Ovviamente, implementare anche un Makefile per generare l'eseguibile partendo dal sorgente C.

Esercizio ZZZ02_10 - **Incrocio nebbioso**

L'ameno paesino di Gattolino è collocato nella nebbiosa pianura a nord di Cesena.
A Gattolino esiste una sola strada, a due corsie, a forma di 8, che perciò contiene un incrocio a 4 vie, senza semaforo. Ormai da anni, 5 auto percorrono quella strada senza sosta in un verso, ed altre 10 nel verso opposto, compiendo mezzo otto in 3 secs, cercando un uscita che non esiste.
L'incrocio può essere attraversato contemporaneamente nei due sensi di marcia, ma non può essere attraversato contemporaneamente da auto che procedono in direzioni perpendicolari.
Poiché non c'è semaforo, la precedenza per l'attraversamento segue le regole della viabilità generale: ha la precedenza chi viene da destra. Inventarsi un modo per evitare deadlock.
Purtroppo, quando c'è nebbia le auto non possono attraversare l'incrocio e devono aspettare che la nebbia si diradi. Una volta diradata la nebbia, le auto controllano le precedenze e procedono all'attraversamento. Mentre un'auto attraversa l'incrocio, se cala la nebbia l'auto prosegue. Ogni volta che un auto attraversa l'incrocio, il guidatore urla di gioia il proprio nome per lo scampato pericolo.
La nebbia cala improvvisamente ogni 3 secondi circa, e dopo 1 secondo se ne va permettendo nuovamente l'attraversamento.

Modellare ed implementare il sistema descritto, utilizzando dei thread POSIX per ciascuna figura (le auto, la nebbia) ed avvalendosi delle opportune strutture dati per la sincronizzazione. Scrivere il Makefile per generare l'eseguibile. Realizzare **il controllo di errore nelle chiamate a funzione delle librerie dei pthread. In caso di errore grave, terminare il programma.**



Occhio al deadlock !

Esercizio ZZZ02_11 - **Cani e Biciclette**

Uno sconosciuto ciclista, vic, pedala sulle strade sterrate che si inerpicano verso Ciola Araldi, nelle colline tra Cesena e Sogliano.
Nelle peggiori salite, ogni cane che incontra cerca di morderlo alle caviglie, e vic si difende ringhiando e scalciando mentre continua a pedalare.
I cani sono talmente tanti che non riescono ad avvicinarsi tutti alle caviglie di vic, così si organizzano formando una fila che rincorre vic. Al massimo due cani possono mordere contemporaneamente la stessa caviglia. Dopo un primo tentativo di morso, il cane si becca un calcio nei denti e torna in fondo alla fila, aspettando il proprio turno. Quando una caviglia è occupata da meno di due cani, il primo cane della fila va a mordere quella caviglia.

Modellare ed implementare il sistema descritto, utilizzando dei thread ed avvalendosi delle opportune strutture dati per la sincronizzazione. Scrivere il Makefile per generare l'eseguibile. Realizzare **il controllo di errore nelle chiamate a funzione delle librerie dei pthread. In caso di errore grave, terminare il programma.**

Esercizio ZZZ02_12 - **La Setta**

Una setta religiosa, con unica sede a Gattolino di Cesena, venera un Dio bizzarro che mira a sincronizzare in maniera egualitaria i suoi adepti.
Nel luogo di culto ci sono N=16 sacerdoti, di cui M=10 sono novizi e L=6 sono anziani.
I sacerdoti novizi devono svolgere ripetutamente le seguenti azioni:

- prendere un bicchiere di vino (santo) da un primo distributore inesauribile,
- inginocchiarsi di fronte alla statua del Dio e versare il vino ai suoi piedi.

I sacerdoti anziani, invece, devono svolgere ripetutamente le seguenti azioni:

- prendere un bicchiere di vino (buono) da un secondo distributore inesauribile,
- allontanarsi dal distributore e bere il bicchiere di vino buono,
- prendere un altro bicchiere di vino (santo) dal primo distributore inesauribile,
- inginocchiarsi di fronte alla statua del Dio e versare il vino ai suoi piedi salmodiando frasi sconnesse .

Vanno rispettati alcuni dettami divini:

- un solo sacerdote per volta può attingere al distributore di vino santo.
- un solo sacerdote per volta può attingere al distributore di vino buono.
- un solo sacerdote per volta può inginocchiarsi di fronte alla statua.
- ciascun sacerdote può tornare ad inginocchiarsi nuovamente per la n+1 esima volta solo dopo che anche tutti gli altri sacerdoti si sono inginocchiati per n volte.

Inoltre, tutti conoscono il numero totale di sacerdoti.
Modellare ed implementare il sistema descritto, utilizzando dei thread POSIX per ciascuna figura (novizi, anziani) ed avvalendosi delle opportune strutture dati per la sincronizzazione. Scrivere il Makefile per generare l'eseguibile. Realizzare **il controllo di errore nelle chiamate a funzione delle librerie dei pthread. In caso di errore grave, terminare il programma.**