Esercizio 1

Scrivere un programma C in cui:

- Il master thread:
  - Inizializza una variabile globale a=0
  - crea un thread produttore ed un thread consumatore.
  - In un ciclo infinito visualizza il valore di a.
- Il Thread produttore:
  - incrementa, ad ogni passola variabile a di due unita' e dorme per un secondo.
- Il Thread consumatore:
  - decrementa la variabile a di una unita' e dorme per un secondo.
- Il processo termina quando a>10

I thread utilizzano un mutex per la sincronizzazione.

Esercizio 2

Scrivere un programma C che crea 10 thread produttori e 5 thread cosumatori:

- I thread condividono una variabile intera.
- Ogni thread produttore incrementa di una unita' la variabile condivisa se e solo se il valore della stessa e' inferiore a 20.
- Ogni thread consumatore decrementa di una unita' la variabile condivisa se e solo se il valore della stessa e' superiore a 5.
- Tutti i thread, dopo l'operazione sulla variabile condivisa dormono per un secondo.

Utilizzare una condition variable per la sincronizzazione.

Esercizio 3

Scrivere un programma C che crea 5 thread produttori e 5 thread cosumatori:

- I thread condividono una coda circolare implementata tramite un array di 100 interi.
- Ogni thread produttore
  - genera cinque numeri casuali (funzione random())
  - Inserisce gli interi in coda
- Ogni thread consumatore
  - Estrae un elemento dalla coda e lo visualizza.
- Tutti i thread, dopo l'operazione sulla variabile condivisa dormono per un secondo.

Utilizzare una condition variable per la sincronizzazione.

Esercizio 4

Scrivere un applicazione multi-thread in linguaggio C in cui il master thread:

- Crea un thread che:
  - esamina ricorsivamente le directory a partire dalla directory corrente
  - inserisce i nomi di tutti i file regolari in una lista dinamica “elenco” condivisa tra tutti i thread.
  - prima di terminare, pone a 0 la variabile “termina”
- Crea 5 thread.
  - Ogni thread riceve come parametro una stringa (ad. Esempio: “thread1”, “thread2”, “thread3”, “thread4”, “thread5”)
  - Se esistono elementi nella lista:
    - Estrae un elemento dalla lista.
    - Apre il file
    - Legge dal file una riga (si assuma che tutte le righe dei file consistano di esattamente 100 byte).
    - Se la riga contiene la stringa ricevuta come parametro, inserisce il nome del file in una lista “match” condivisa tra tutti i thread.
  - Termina quando la variabile “termina” e’ pari a 0.

Esercizio 5

La dimensione di un file e' pari al numero di byte che occupa su disco.

Definiamo ricorsivamente la "dimensione" di una directory come segue:

- La dimensione di una directory vuota e' pari a zero.
- La dimensione di una directory che contiene solo file regolari e' pari alla somma delle dimensioni dei file regolari.
- La dimensione di una directory che contiene file e sottodirectory e' pari alla somma delle dimensioni dei file regolari e delle dimensioni delle sottodirectory in essa contenute.

Scrivere un'applicazione multi-thread che calcola ricorsivamente la dimensione della directory corrente, con le seguenti modalita':

- Per ogni sottodirectory viene creato un nuovo thread che calcola la dimensione della sottodirectory ad esso assegnato.

Esercizio 6

Scrivere una applicazione multi-thread in liguaggio C riceve su riga di comando i nomi di due file f1 ed f2:

- Sia assuma che la dimensione N del file sia un multiplo di 100 byte (N=100 k).
- Il master thread
  - Apre il file f1 in lettura.
  - Apre il file f2 in scrittura. Se f2 esiste, il suo contenuto viene eliminato.
  - Crea k thread $T_0, T_1, \ldots, T_{k-1}$.
  - Il Thread T_i analizza i byte nel range compreso tra 100i e 100(i+1)-1
    - T_0 analizza i byte da 0 a 99;
    - T_1 analizza i byte da 100 a 199;
    - …
  - Il Thread T_i
    - Legge il byte x in posizione j del proprio blocco da f1
    - Converte x nell'intero y
    - Scrive y nella posizione j in f2.

Esercizio 7

- Scrivere un programma in cui :
  - Il master thread
    - Blocca i segnali SIGUSR1 e SIGUSR2
    - Apre un file text.txt (utilizzare una variabile globale per il file descriptor)
    - Crea due thread
    - Invia al proprio PID il segnale SIGUSR1.
    - Attende la terminazione di entrambi I thread.
  - Il thread 0
    - Blocca il segnale SIGUSR1
    - Si mette in attesa di un segnale.
  - Il thread 1
    - Blocca il segnale SIGURS2
    - Si mette in attesa di un segnale.
  - L'handler di SIGUSR1 (risp., SIGUSR2):
    - Visualizza Thread_id ed una riga del file.
    - Attende 1 secondo
    - Invia SIGUSR2 (risp. SIGUSR1) al proprio PID.