ISTRUZIONI PRATICHE

Esame del modulo di laboratorio di “Sistemi Operativi”

Durata: 150’ (2:30 ore)

● Creare una cartella principale denominata con il proprio numero di matricola e dentro esclusivamente un file main.c ed eventuali altri header files.

● Questa cartella andrà consegnata “zippandola” (compressione formato “zip”) in modo da creare un file avente per nome il proprio numero di matricola più l’estensione “.zip”. Deve essere compressa l’intera cartella e non solo il suo contenuto. Se il proprio numero di matricola fosse 123456 questo deve essere anche il nome della cartella e l’archivio compresso da consegnare deve chiamarsi 123456.zip. All’estrazione dovrà essere presente la cartella 123456.

● Consegna:
  ○ Dopo 70’ ed entro 80’ dall’inizio della prova si deve fare una prima consegna (lavoro parziale) con il lavoro compiuto complessivo fino a tale momento ANCHE SE NON FUNZIONANTE utilizzando il modulo nell’homepage del browser. Il file consegnato dovrà chiamarsi **<matricola>_parziale.zip**
  ○ Dopo 120’ ed entro 150’ dall’inizio della prova si deve fare una seconda consegna (lavoro finale) utilizzando il modulo nell’homepage del browser: questa consegna è l’unica considerata per la valutazione finale .
  ○ I punteggi [x] sono indicativi dato che la valutazione tiene conto anche di dettagli “trasversali” che non sono riferibili a singoli punti. Un punteggio complessivo maggiore o uguale a 31 porterà alla lode.

NOTA: parte delle verifiche può avvenire con procedure completamente o parzialmente automatizzate per cui le denominazioni e gli output devono essere rigorosamente aderenti alle indicazioni. Eventuali irregolarità comportano l’esclusione dalla prova oltre a possibili sanzioni disciplinari.
Il codice va opportunamento commentato.

Prima di iniziare, ricordatevi che:

- Codice che non compila non viene valutato.
- La presenza di warnings durante la compilazione porta a delle penalità di punteggio.
- La consegna in ritardo porta a delle penalità di punteggio.
- Il codice deve funzionare! Testate opportunamente ogni sua parte. Se modifiche al codice potrebbero influenzare il lavoro già fatto, testate nuovamente!

Makefile

[3] Deve essere creato un makefile di nome "`Makefile`" contenente un target "`prepara`" ed un target "`compila`". Il target "`prepara`" deve creare un file avente come nome il contenuto della variabile `FILE_DA_CREARE` ricevuta in input da terminale. Il target "`compila`" deve compilare un file `main.c` in un eseguibile chiamato "**`student.out`**" solo dopo aver svolto il target "`prepara`".

Programma in C

Deve essere creato un programma in C che gestisca una partita di carte tra processi. Il gioco è molto semplice: ogni giocatore ha delle carte numerate e ad ogni mano deve giocare la sua carta più bassa. Vince la mano il giocatore con la carta più alta, vince il gioco chi ha vinto più mani.

1. [3] Il processo principale, "*tavolo*", gestirà i vari giocatori ed i vari eventi. Il programma deve essere lanciato con 2 soli parametri `<gameSettings>` e `<checkerPID>`, che corrispondono ad un file ed un PID. Un numero diverso di parametri, un file non esistente o un PID non esistente devono far terminare il programma con codice 40. Il tavolo dovrà rimanere attivo fino alla fine del programma.

2. [4] Il tavolo deve leggere il file `<gameSettings>` contenente un numero di giocatori `<n>` (compreso tra 2 e 10) e deve quindi creare esattamente `<n>` processi figli (del processo principale), uno per ogni giocatore. Una volta creati, i processi devono rimanere attivi fino alla fine del programma. Un file **di esempio** vi viene fornito.

3. [3] Il tavolo deve creare una pipe anonima con ognuno dei figli, che verrà poi usata nei punti successivi.

4. [4] Ogni giocatore (processo al punto 2) deve quindi leggere le proprie carte. Queste saranno contenute in un messaggio sulla coda identificata dal file `<gameSettings>` e `<n>`. La coda (già esistente al momento della valutazione) verrà riempita (in fase di valutazione) con messaggi aventi come tipo il PID del giocatore a cui il messaggio è indirizzato, e come payload un array di 10 interi, rappresentanti le carte. Le carte sono interi (`int`), non stringhe, compresi tra 1 e 9. Ogni giocatore dovrà aspettare il messaggio a lui indirizzato.
   Es: verrà inviato un messaggio di tipo 34 con le seguenti carte: 1, 2, 4, 6, 8, 8, 8, 8, 8, 8 che dovrà essere letto dal giocatore con PID 34, ed un messaggio di tipo 35 con payload 3, 4, 4, 4, 4, 4, 4, 5, 5, 5 per il PID 35.

5. [3] Verranno inviate 10 carte, ordinate dalla più bassa alla più alta. Una volta letto il proprio messaggio, ogni giocatore dovrà inviare un segnale `SIGRTMIN` al processo `<checkerPID>`, avente come payload la propria carta più alta (un intero).
   Es: il giocatore 34 dovrà inviare un segnale con payload 8, il giocatore 35 un segnale con payload 5.

Turno di gioco

6. [4] Ogni giocatore, in ordine di PID (es prima al PID 34 e poi al PID 35), riceverà un segnale `SIGUSR1` all'inizio del turno di gioco. Alla ricezione di questo segnale `SIGUSR1`, il giocatore dovrà iniziare a giocare scegliendo la propria carta più bassa e inviandola al processo tavolo usando la pipe creata al punto 3. Una volta ricevuta, il tavolo dovrà stampare su **stdout** il valore della carta.
   Es: il giocatore 34 invierà al tavolo la carta 1 ed il tavolo stamperà "1\n", il giocatore 35 la carta 3 ed il tavolo stamperà "3\n".

7. [2] Ricevute le carte da tutti i giocatori, il tavolo calcolerà il vincitore della mano basandosi sul valore della carta (vince la più alta, in caso di carta uguale, vince quella del giocatore con il PID maggiore). Il tavolo stampa in **stderr** "`<PIDVincitore> <carta>\n`" con `<PIDVincitore>` uguale al PID del giocatore vincente, `<carta>` uguale alla carta vincente.
   *Es: se ci sono 2 soli giocatori, 34 e 35, il primo round lo vincerà il giocatore 35 con la carta 3, ed il tavolo scriverà "35 3\n" su **stderr**. Il terzo round, un pareggio, lo vincerà il giocatore 35 con la carta 4 per PID maggiore, ed il tavolo scriverà "35 4\n" su **stderr**,*

Alla fine del turno di gioco, verrà effettuato un nuovo invio di `SIGUSR1` per proseguire il gioco. NB: una volta usata una carta, questa deve essere scartata!

8. [2] Implementare il punto 6 affinchè funzioni con i segnali `SIGUSR1` che vengono inviati con ordine casuale ai vari giocatori (quindi non più in ordine di PID).

9. [4] Alla fine gioco, quindi quando tutte le carte sono state giocate, il giocatore che ha vinto di più scriverà il proprio PID ed il numero di vittorie, con una stringa "`<PID>-<vittorie>`" sul file `<gameSettings>`, sovrascrivendone il contenuto.
   Es: il giocatore 34 scriverà "34-7" sul file.