ISTRUZIONI PRATICHE

Esame del modulo di laboratorio di “Sistemi Operativi”

Durata: 120’ (2 ore)

● Creare una cartella principale denominata con il proprio numero di matricola e dentro tutti i contenuti richiesti dal compito.

● 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 50’ ed entro 60’ 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. Nominare l’archivio parziale con “**<matricola>_parziale.zip**”

  ○ Dopo 90’ ed entro 120’ 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.

  ○ Per consegnare, effettuare il login con il proprio account universitario e selezionare il file “zip” da allegare.

  ○ Tutti i punteggi [x] e [y] 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. I punteggi [y] sono indipendenti, mentre i punteggi [x] devono essere svolti in ordine e testati accuratamente!

NOTA: le denominazioni e gli output devono essere rigorosamente aderenti alle indicazioni.

Eventuali irregolarità comportano l’esclusione dalla prova oltre a possibili sanzioni disciplinari.

Consegna

NB:

- **La mancata presenza di opportuni commenti nel codice può portare a delle penalizzazioni.**
- **La generazione di warnings nella compilazione porterà alla perdita di punti.**
- **Errori di compilazione porteranno alla completa invalidazione della prova.**
- **Fate il flush dopo la scrittura su stdout e stderr!**

Deve essere creato un programma che simuli la gestione di un torneo di ping pong.

Preparazione:

1. [2] Il programma deve accettare esattamente 2 parametri: il nome di un file `<arbitro>` esistente, e un intero compreso tra 2 e 32 `<numeroGiocatori>`. In caso di parametri errati il programma dovrà terminare con codice **50**.
   ***ES**: verrà invocato con `./main.out` `/percorso/arbitro` `32`*

2. [3] Il programma dovrà creare **esattamente** `<numeroGiocatori>` giocatori. Ogni giocatore è un processo figlio del programma principale.
   ***Es**: se `<numeroGiocatori>` è 32, il programma dovrà creare 32 processi figli (i giocatori).*

Il programma principale deve aspettare senza terminare!
Hint: testate che il programma abbia creato tutti e solo i giocatori richiesti!

Registrazione torneo:

Ogni giocatore dovrà iscriversi al torneo creando un file di registrazione. Più precisamente

3. [4] Ogni giocatore dovrà creare un nuovo file **`.txt`** nella cartella `/tmp/` avente come nome **`<mioPID>.txt`**, ossia il PID del giocatore che si sta registrando. Il file dovrà contenere una stringa con l'`<ELO>` del giocatore "*`<ELO>\n`*". L'`<ELO>` rappresenta il livello del giocatore, ed è un numero multiplo di 100 nel range di 100-4000 (compresi). Ogni giocatore dovrà avere un ELO diverso.
   ***Es**: il giocatore 24 creerà il file `/tmp/24.txt` con contenuto "`2300\n`", il giocatore 25 creerà il file `/tmp/25.txt` con contenuto "`1200\n`".*

4. [3] Per evitare distrazioni, i giocatori dovranno bloccare il segnale SIGUSR1 inviato dagli spettatori.

A questo punto del programma, tutti i giocatori (ed il processo principale) dovrebbero essere in attesa di giocare!
Hint: controllate che tutti i file siano stati creati con il contenuto giusto e che tutti i giocatori siano ancora attivi!

Turno di gioco:

Una volta che tutti i giocatori si sono registrati, l'arbitro (in fase di valuazione) manderà segnali ai vari giocatori con un **payload** indicante l'avversario `<PID_avversario>` con cui devono giocare. I segnali saranno di tipo `SIGUSR2` ed il payload `<PID_avversario>` sarà un intero.

5. [3] Una volta che i giocatori ricevono il segnale con `<PID_avversario>` dovranno scrivere su **stdout** "`Gioco contro <PID_avversario>\n`".
   Es: *il giocatore 24 riceve un segnale con payload 25. Dovrà quindi scrivere su stdout "`Gioco contro 25\n`".*

6. [5] Una volta che i giocatori ricevono il segnale con `<PID_avversario>`, si decreterà il vincitore in base all'ELO. Vincerà sempre il giocatore con ELO maggiore, il quale dovrà scrivere su **stdout** "`<mioPID> vinco\n`", mentre il giocatore che perde dovrà scrivere "`<mioPID> perdo\n`", con `<mioPID>` corrispondende al PID del giocatore.
   ES*: Se si sfidano il giocatore 24 con ELO 2300 ed il giocatore 25 con ELO 1200, il giocatore 24 scriverà su stdout "`24 Vinco\n`", mentre il 25 scriverà "`25 Perdo\n`".*

7. [6] Entrambi i giocatori (i processi figli) dovranno **accordarsi sul punteggio** usando una tecnica di IPC a scelta tra quelle viste nel corso.
   A ping pong, le partite si concludono a 11 punti, senza vantaggi per semplificare la vita dei giocatori. Il punteggio deve essere **random** (potete usare **`srand()`** e **`rand()`**)***** ed essere coerente con il risultato finale (vince sempre chi ha l'ELO maggiore, come da punto precedente).
   Una volta deciso il punteggio, entrambi i giocatori devono inviare un messaggio sulla **coda preesistente** identificata dalla coppia (`<arbitro>`,`<numeroGiocatori>`), con tipo uguale a `<PID_vincitore>`, ed un payload stringa (dimensione 100) avente come testo "`<mioPID>-<mioPunteggio>-<punteggioAvversario>`".
   ***** Vien da sé che il punteggio del vincitore è sempre 11, mentre il del perdente è random.

   ES*: se i giocatori 24 e 25 si affrontano e concordano il punteggio di 11 a 6 per il giocatore 24 (che vince per l'ELO maggiore), dovranno inviare un messaggio sulla coda con tipo 24 avente come payload "`24-11-6`" e "`25-6-11`", inviati rispettivamente dal processo 24 e 25.*

Hint: createvi la coda per testare e controllate che i messaggi arrivino e che abbiano il tipo e il payload corretto!

Dopo partita

Una volta inviato un punteggio, il giocatore vincente dovrà tornare ad aspettare indicazioni dall'arbitro per la nuova partita. Il giocatore perdente dovrà terminare. Inoltre

8. [4] Il giocatore vincente riceverà un incremento di ELO dopo ogni partita. Questo incremento è pari all'ELO dell'avversario battuto, diviso 4 (usate divisioni tra interi, per esempio 1350/4 = 337). Il vincente dovrà dunque stampare su stdout "`<mioPID> nuovo ELO: <ELO>\n`". L'incremento dovrà influenzare il punto 6!
   NB: gli ELO sono dinamici, due vincitori che si sfidano avranno un ELO diverso da quello iniziale! Attenzione al punto 6!

   ***ES**: Se si sfidano il giocatore 24 con ELO 2300 ed il giocatore 25 con ELO 1200, il giocatore 24 vincerà ed arriverà ad ELO 2600 (2300 + 300), e scriverà su stdout "`24 nuovo ELO: 2600\n`".*

9. [2] Il programma principale dovrà aspettare che tutti i figli abbiano terminato prima di poter terminare.