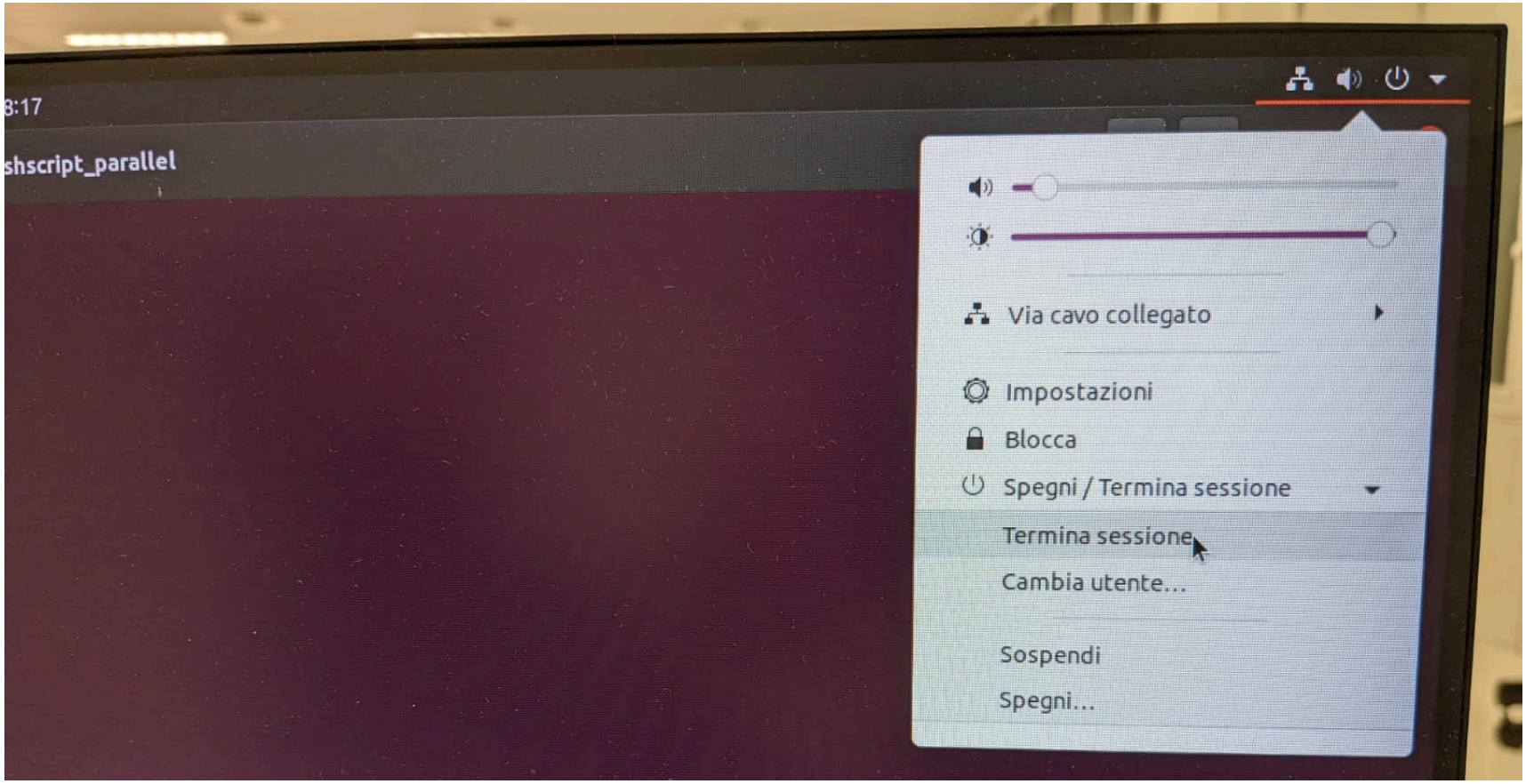
Corso di Sistemi Operativi

Corso estinto di Sistemi Operativi e Reti - Modulo Sistemi Operativi

Prova scritta - Febbraio 2025

LEGGI ATTENTAMENTE LE ISTRUZIONI:

1. **Rinomina** subito la cartella chiamata "Cognome-Nome-Matricola" che hai trovato sul Desktop e in cui hai trovato questa traccia, sostituendo "Cognome" "Nome" e "Matricola" con i tuoi dati personali e **lasciando i trattini senza spazi**; se hai un doppio nome oppure un doppio cognome dovrai chiamare la cartella come in questo esempio:
    a. DeLuca-MarcoGiovanni-199999
2. **Carica** tutto il materiale didattico che vorrai usare sul Desktop; puoi farlo solo nei primi 5 minuti della prova;
3. **Svolgi** il compito; lascia tutto il sorgente che hai prodotto nella cartella di cui al punto 1;
4. Quando hai finito fai "**Termina Sessione/Logout**", ma lascia la postazione mantenendo il PC acceso.
5. **E' tua diretta responsabilità** garantire l'integrità del tuo elaborato, anche in caso di assenza di corrente. **Salva spesso il tuo lavoro**



e NON spegnere il PC.

SALVA SPESSO

CI SONO DEI PUNTI AMBIGUI NELLA TRACCIA? DECIDI TU COSA FARE

È parte integrante di questo esercizio completare le specifiche date nei punti non esplicitamente definiti, introducendo nuove strutture dati, o estendendo quelle preesistenti laddove si ritenga necessario, risolvendo eventuali ambiguità. Si può cambiare il codice dei metodi esistenti dove serve.

POSSO CAMBIARE IL PROTOTIPO DEI METODI RICHIESTI O DI QUELLI ESISTENTI? NO

Il codice che scriverai è pensato per essere usato come un modulo di libreria. Un modulo di libreria potrebbe essere usato da altri programmatori, i quali si aspettano di trovare una specifica interfaccia.

Non è quindi consentito modificare il prototipo dei metodi pubblici di una classe se questi sono stati forniti. Puoi aggiungere qualsivoglia campo e metodo privato, e qualsivoglia classe ausiliaria, ma NON variare l'interfaccia dei metodi pubblici già specificati. Analogamente, i metodi esistenti possono essere modificati nel loro codice, ma non se ne deve cambiare l'interfaccia, il risultato finale o il significato.

CHE LINGUAGGIO POSSO USARE? PYTHON 3.X

POSSO CONSENTIRE SITUAZIONI DI RACE CONDITION NEL MIO CODICE? NO
POSSO CONSENTIRE SITUAZIONI DI DEADLOCK NEL MIO CODICE? NO
POSSO CONSENTIRE ALTRE SITUAZIONI DI BLOCCO TOTALE NEL MIO CODICE, TIPO NESTED LOCKOUT, LIVELOCK O ALTRO? NO
POSSO CONSENTIRE SITUAZIONI DI STARVATION NEL MIO CODICE? SI, tranne quando ti viene chiesto esplicitamente di rimuoverle
MA IL MAIN() LO DEVO AGGIORNARE? E I THREAD DI PROVA? SI

E' obbligatorio implementare esplicitamente il codice di prova oppure modificare il codice di prova pre-esistente, e accertarsi che giri senza errori prima della consegna.

ESERCIZIO 1 - PROGRAMMAZIONE MULTITHREADED

(Punteggio minimo richiesto 18/30. Pesa per ⅔ del voto finale)

Svolgi tutti i punti della traccia in un unico file che chiamerai `SOLUZIONE.py`

-> Qualsiasi altro nome di file verrà IGNORATO <-

Punto 1.
Avrai notato che ogni `Player` incrementa periodicamente il valore della variabile `self.posizione` nell'istanza di `TiroAllaFune` corrente. L'incremento è di `1` o di `-1`, a seconda della squadra di cui si fa parte. Fai in modo che ogni `Player` disponga di un valore di energia. L'energia di partenza per ogni giocatore è fissata a `10`. Ogni volta che un Player svolge una operazione `tira()`, l'energia del Player deve diminuire di 1. Quando l'energia di un `Player` scende a 0, l'operazione `tira` non deve avere più effetto. Dopo aver fatto questo, introduci nel gioco i Thread di tipo `Caricatore`. I Thread di tipo `Caricatore` aumentano, a intervalli di tempo casuali, l'energia di un `Player` scelto a caso, di una quantità scelta a caso compresa tra 1 e 2. Dopo averne progettato il codice, istanzia un certo numero, a tua scelta, di thread Caricatori nel metodo `start` che avvia il gioco.

Punto 2.
Revisiona il metodo `stampa_situazione` in maniera tale da visualizzare anche i valori di energia di tutti i giocatori. Dopodiché sposta l'invocazione delle operazioni di stampa della situazione nel codice della `run()` di un thread di tipo `Visualizzatore` separato. Il Visualizzatore aggiorna lo schermo mostrando la situazione corrente solo quando gli viene notificata la presenza di una modifica alla situazione di gioco corrente. La situazione di gioco include la posizione della fune, ma anche i valori di energia di tutti i player.

Punto 3.
Prevedi che la partita finisca in pareggio nel caso in cui non sia stato determinato un vincitore dopo `MAX_OPERAZIONI` operazioni di tipo `tira`.

Punto 4.
Prevedi che un `Player` possa andare in modalità riposo se la sua energia scende sotto il valore di 3. Un Player in modalità "riposo" deve restare in stato di wait su una condition opportunamente introdotta, fino a che la propria energia non risale.

Punto 5.
Introduci un Thread `Killer`. Il thread Killer elimina a intervalli regolari il `Player` che si è rivelato più "debole" fino a quel momento. Il Player più "debole" è considerato essere quello che ha effettuato il minor numero di operazioni `tira()` fino a quel momento. Decidi tu cosa fare quando c'è più di un `Player` in queste condizioni.

SALVA SPESSO

ESERCIZIO 2 - LINGUAGGI DI SCRIPTING

(Punteggio minimo richiesto 18/30. Pesa per ⅓ del voto finale)

Scrivi uno script perl chiamato `elabora_file.pl` che si comporti come segue:

1. Stampa su STDOUT gli ultimi comandi eseguiti dall'utente corrente che abbiano come primo comando `ls`, fino a un massimo di 10;
2. Se invocato come
   `./elabora_file.pl str`
   stampa su un file chiamato `trovati.log` tutte le linee di comando eseguite nel corso del tempo dall'utente corrente e che contengano il testo `str` al loro interno;
3. Crea un file di backup chiamato `backup.tar.gz` nella cartella corrente che contenga tutto lo storico dei comandi eseguiti dall'utente corrente. Si consiglia di utilizzare il comando `tar`.