Università degli Studi di Napoli Federico II
Corso di Laurea in Ingegneria Informatica
Esame di Sistemi Operativi
Proff. Cinque, Cotroneo, Natella

Prova pratica del 21/12/2020 - TURNO 10
Durata della prova: 75 minuti

Lo studente completi il programma a corredo di questo documento, seguendo le seguenti indicazioni.

La prova sarà valutata come segue:

- **A**: Prova svolta correttamente.
- **B**: Il programma non esegue correttamente, con errori minori di programmazione o di concorrenza.
- **C**: Il programma non esegue correttamente, con errori significativi (voto max: 22).
- **INSUFFICIENTE**: Il programma non compila o non esegue, con errori gravi di sincronizzazione.

Istruzioni per la consegna dell'elaborato

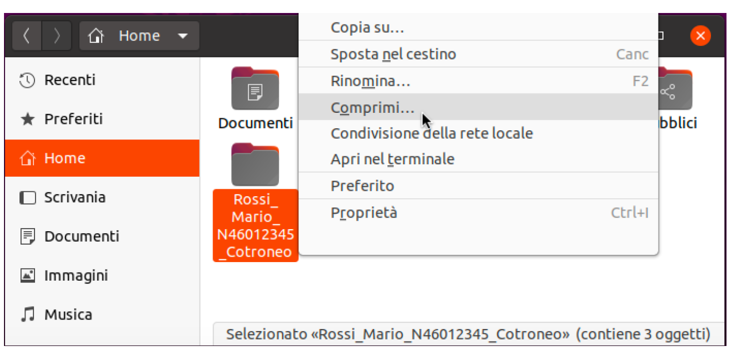
L'elaborato dovrà essere svolto in una cartella dal nome: `Cognome_Nome_Matricola_Docente`

Esempio:

mkdir Rossi_Mario_N46012345_Cotroneo
cd Rossi_Mario_N46012345_Cotroneo
…. copiare nella cartella i file forniti per l'esame ….
…. sviluppare il programma ….
…. per la consegna, dalla shell (assumendo di essere ancora nella cartella di lavoro), creare un file compresso ("tar") con i seguenti comandi:

```
cd ..
tar -czvf ./Rossi_Mario_N46012345_Cotroneo.tar.gz  ./Rossi_Mario_N46012345_Cotroneo
```

In alternativa, è consentito creare il file compresso "tar" tramite l'interfaccia grafica.



All'interno della macchina virtuale, aprire il browser all'indirizzo: **https://tinyurl.com/y97qfmx9**

Nel form, cliccare su "**aggiungi file**", selezionare il file compresso contenente il proprio svolgimento, e indicare il proprio nome ed email @studenti.unina.it.

Attendere una notifica del docente, e quindi scollegarsi dalla piattaforma di VirtualClassroom.

Testo della prova

Si realizzi in linguaggio C/C++ un processo servente **multiprocesso** basato su **code di messaggi UNIX**. Il processo servente, denominato **Store**, genera due tipologie di processi, per gestire le richieste in *load balancing*: processi **Scrittori** che ricevono richieste di tipo *WRITE* da una coda di messaggi, e processi **Lettori**, che ricevono richieste di tipo *READ* dalla stessa coda. Entrambe le tipologie di processi hanno accesso ad un **Magazzino** condiviso, contenente un array di **3** elementi, ognuno rappresentante il livello di *scorte* (intero) di una certa tipologia di prodotto. La sincronizzazione tra i processi Lettori e Scrittori sui singoli elementi del Magazzino deve avvenire attraverso lo schema **lettori-scrittori con starvation degli scrittori**, implementato attraverso **semafori**, con la particolarità che l'accesso (anche in scrittura) ad un elemento del Magazzino non deve ritardare l'accesso di un altro processo ad un elemento diverso. I messaggi di tipo WRITE contengono la quantità di cui decrementare il livello di scorta e l'id numerico (tra 0 e 2) del prodotto da aggiornare. Alla ricezione di un messaggio WRITE, lo scrittore decrementa il livello di scorta del prodotto indicato nella richiesta di una quantità pari a quella indicata nella richiesta. I messaggi di tipo READ contengono il pid del processo mittente e l'id numerico (tra 0 e 2) del prodotto di cui si vuole conoscere il livello di scorta. I processi lettori rispondono al mittente con il livello di scorta correntemente disponibile per il prodotto indicato nella richiesta.

Il processo Store istanzia 2 processi Scrittori, ognuno dei quali gestisce 2 richieste, e 3 processi Lettori, ognuno dei quali gestisce 4 richieste. Le richieste sono inviate da due tipologie di processi: 1) i **Viewer**, ognuno dei quali invia 4 richieste di tipo READ sulla coda di richiesta indicando un id prodotto scelto casualmente, con la funzione *rand()*, tra 0 e 2; prima di inviare la successiva richiesta, il Viewer attende la risposta, con il livello di scorta, sulla coda di risposta; 2) i processi **Updater**, ognuno dei quali invia 2 richieste di tipo WRITE sulla coda di richiesta, indicando un id prodotto, scelto casualmente tra 0 e 2, e la quantità di cui decrementare, scelta casualmente tra 1 e 4.

I processi Viewer, Updater e Store (che a sua volta genera i processi Lettori e Scrittori) devono risiedere in **tre eseguibili distinti**, che sono invocati da un **programma principale** attraverso una delle varianti della primitiva **exec**. Il programma principale genera 3 Viewer, 2 Updater e un processo Store. Quando tutti i processi terminano, il programma principale rimuove le code e termina a sua volta.