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

COMPITO SVOLTO

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`

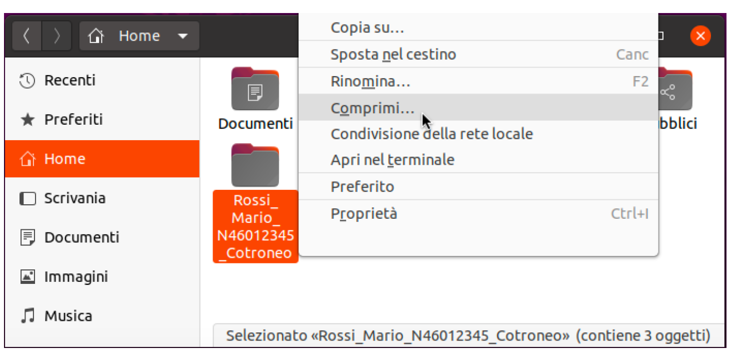
Esempio:

```
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 -cvfz ./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: **TBD**

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'applicazione **multiprocesso**, basata sul costrutto **Monitor di Hoare**, che realizzi lo schema **produttore/consumatore con vettore di stato**.

In aggiunta allo schema di sincronizzazione classica, si richiede la seguente **variante**. Si supponga di voler impedire che un numero eccessivo di processi produttori si accumuli in attesa all'interno del monitor, per evitare una condizione di sovraccarico. I processi produttori, nel caso in cui il vettore di buffer sia pieno, dovranno quindi verificare se il numero di processi già in attesa è minore di 4 (usando la variabile `produttori_in_attesa`). **Se i processi in attesa sono meno di 4**, **il processo produttore dovrà sospendersi** come al solito per l'attesa che un buffer si renda disponibile. Invece, **se vi sono almeno 4 processi in attesa**, **il monitor dovrà forzare l'uscita del processo produttore**, facendo `return` di un valore diverso da 0.

Il processo chiamante del metodo `produzione()`, nel caso sia ritornato un valore diverso da 0, dovrà attendere 3 secondi e riprovare la produzione, fin quando la produzione non abbia successo. Rifarsi allo snippet di codice seguente.

```
for (5 produzioni) {
    while(1) {
        ret  =  produzione(monitor,
valore);
        if(ret == 0) { break; }
        sleep(3);
    }
    sleep(1);
}
```

Per i processi consumatori, non è richiesto alcun controllo sul numero di processi già in attesa. Il programma principale dovrà creare 6 produttori e 2 consumatori. Ogni produttore dovrà produrre 5 valori, e il consumatore consumarne 15. I processi produttori dovranno attendere 1 secondo tra le produzioni, e i consumatori dovranno attendere 2 secondi tra le consumazioni.