

## Corso di Sistemi Operativi e Reti, corso di Sistemi Operativi - 12 Set 2013 -

**1. PER GLI STUDENTI DI SISTEMI OPERATIVI E RETI:** è necessario sostenere e consegnare entrambi gli esercizi. Sarà attribuito un unico voto su tutta la prova.

**2. PER GLI STUDENTI DI SISTEMI OPERATIVI:** si può sostenere solo uno dei due esercizi se si è già superata in un appello precedente la corrispondente prova. Il tempo a disposizione in questo caso è di 2 ORE.

Troverete sul vostro Desktop una cartella chiamata "CognomeNomeMatricola" che contiene la traccia dell'elaborato ed eventuali altri file utili per lo svolgimento della prova. Ai fini del superamento della prova è indispensabile rinominare tale cartella sostituendo "Cognome" "Nome" e "Matricola" con i vostri dati personali. Ad esempio, uno studente che si chiama Alex Britti ed ha matricola 66052 dovrà rinominare la cartella "CognomeNomeMatricola" in "BrittiAlex66052".

*Non saranno presi in considerazione file non chiaramente riconducibili al proprio autore. E' possibile caricare qualsiasi tipo di materiale didattico sul desktop nei primi 5 minuti della prova.*

**Si consiglia di salvare SPESSO il proprio lavoro.**

### ESERCIZIO 1 (Linguaggi di scripting)

Il file *posti.txt* definisce la disposizione dei posti in un teatro, la cui configurazione è quella riportata nella **Fig.1**. Ciascun posto è rappresentato mediante un identificativo alfanumerico costituito da un numero ed una tra le lettere A,B,C. Il numero corrisponde alla numerazione dei posti, la lettera identifica la categoria del posto (C sta per platea, B per loggione, A per palchi laterali) in base alla quale varia il prezzo del biglietto. La stringa "OCC" (evidenziata in **Fig. 1**) indica che quel posto è stato precedentemente riservato per un'altra richiesta (quindi non è più disponibile).

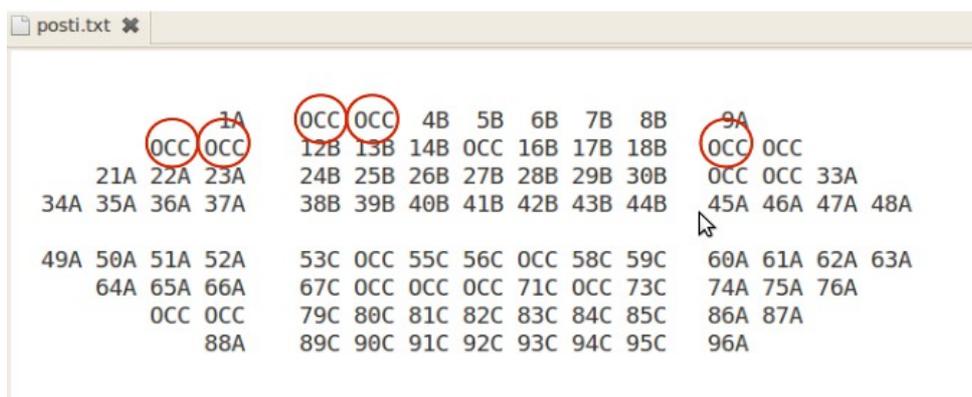


Fig.1 Il contenuto del file *posti.txt*

Il file *richiesta.txt* riporta su ciascuna linea il numero di biglietti richiesti per ciascuna tipologia secondo il formato rappresentato nella **Fig. 2**.

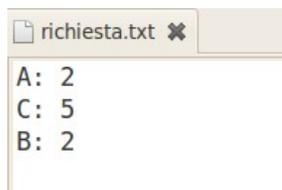


Fig.2 Il contenuto del file *richiesta.txt*

Si vuole implementare uno script perl **biglietteria.pl** che crei una prenotazione (riservando **tutti** i posti) sulla base della richiesta. In particolare, un posto può essere assegnato se:

- non è occupato (non è contrassegnato dalla stringa "OCC") ed
- è del tipo richiesto (A,B,C)

Ad esempio, rispetto alla richiesta mostrata in **Fig. 2**, ovvero 2 posti di tipo A, 5 posti di tipo C e 2 posti di tipo B, lo script deve assegnare i posti 1 e 9 (ti tipo A), 53, 55, 56 (di tipo C), 4 e 5 (di tipo B). I posti assegnati sono i primi disponibili della tipologia corrispondente alla richiesta.

A tal scopo, lo script deve:

- leggere da linea di comando il file *richiesta.txt* al fine di elaborare la richiesta in esso contenuta
- leggere il file *posti.txt* per individuare i posti da assegnare per soddisfare interamente la richiesta ricevuta e, in particolare
  - se è possibile trovare tutti i posti necessari per soddisfare la richiesta, deve emettere un biglietto
  - altrimenti, deve restituire su standard output la stringa "Non è stato possibile completare la prenotazione con successo."

Il biglietto eventualmnte restituito dallo script è memorizzato su un nuovo file (da creare) di nome *biglietto.txt* con il seguente contenuto (vedi **Fig. 3**):

- i posti assegnati per la richiesta, esplicitando SOLO il numero del posto che è stato riservato (rispetto al tipo richiesto)
- la configurazione dei posti, utile per capire dove sono posizionati i posti appena prenotati

```

biglietto.txt ✖
Hai prenotato i seguenti posti:
posto 1 di tipo A.
posto 9 di tipo A.
posto 53 di tipo C.
posto 55 di tipo C.
posto 56 di tipo C.
posto 58 di tipo C.
posto 59 di tipo C.
posto 4 di tipo B.
posto 5 di tipo B.

      1A   OCC OCC  4B  5B  6B  7B  8B   9A
      OCC OCC  12B 13B 14B OCC 16B 17B 18B  OCC OCC
      21A 22A 23A  24B 25B 26B 27B 28B 29B 30B  OCC OCC 33A
34A 35A 36A 37A  38B 39B 40B 41B 42B 43B 44B  45A 46A 47A 48A

49A 50A 51A 52A  53C OCC 55C 56C OCC 58C 59C  60A 61A 62A 63A
  64A 65A 66A  67C OCC OCC  71C OCC 73C  74A 75A 76A
      OCC OCC  79C 80C 81C 82C 83C 84C 85C  86A 87A
      88A  89C 90C 91C 92C 93C 94C 95C  96A
  
```

**Fig.3** Il contenuto del file *biglietto.txt*

## ESERCIZIO 2 (Programmazione multithread)

Si vuole implementare un sistema automatico per l'esecuzione di task (o attività) eventualmente correlati tra loro da vincoli di precedenza, al fine di completare tutte le attività nel rispetto dei vincoli imposti con il massimo grado di parallelismo. In particolare, il sistema ha a disposizione **3 thread** di tipo **Worker** in grado di gestire un generico **Task**.

Strutture dati da progettare:

**Task:** un task ha un *nome* ed una eventuale *lista di altri task* da cui dipende. Dispone, inoltre, di un metodo *esegui()* che può essere invocato per l'esecuzione dell'attività ad esso associata. In particolare, un task può essere eseguito se:

- (\*\*\*) non è già stato completato, ovvero, se valgono congiuntamente le due seguenti condizioni:
  1. il metodo *esegui()* non è stato ancora invocato (quindi non è stata ancora svolta la sua attività) e
  2. tutti gli eventuali altri task da cui dipende sono stati già completati

Ad esempio, se il task di nome *task1* ha una lista di dipendenze pari a  $\{task5, task2\}$  esso può essere completato solo se tutti i task da cui dipende sono stati già eseguiti (e, cioè, se sia il *task5* che il *task2* sono stati completati).

**NOTA:** Per semplicità, si assuma che non ci siano situazioni di reciproca dipendenza (diretta o indiretta) tra i task, ovvero, che non sia possibile che l'esecuzione di un task sia vincolata mediante una catena di dipendenze alla sua stessa attività.

```
class Task {
    //
    //
    // ... da estendere a cura dello studente

    public void esegui() {
        sleep(...)
    }
}
```

**Worker:** un *Worker* è un thread destinato all'esecuzione di un generico task. In particolare, esso preleva il primo dei task disponibili per l'esecuzione e lo esegue invocando su di esso il metodo *esegui()* se le condizioni sopra esplicitate (\*\*\*) sono soddisfatte. Diversamente, prova ad eseguire un altro task tra quelli disponibili.

Qualsiasi altra struttura dati ritenuta necessaria per lo svolgimento dell'esercizio è a cura dello studente (come, ad esempio, l'utilizzo di strutture dati ausiliarie o variabili necessarie a segnalare/aggiornare/memorizzare lo stato dell'attività associata ad un task -per indicare se è stata completata o meno-).

E' consentito (ma non obbligatorio), fare uso delle funzioni disponibili nel JDK di Java 6, e di qualsiasi altra funzione predefinita a disposizione. Non saranno valutate versioni sequenziali del programma. Il lavoro può essere sviluppato o in Java o in C++ .