Corso di Sistemi Operativi e Reti, corso di Sistemi Operativi – 6 Dic 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 *parcheggi.txt* definisce la disposizione dei posti macchina condominiali per uno stabile di *20* appartamenti. Ciascun posto auto è delimitato dal simbolo "|".
Ciascun condomino è identificato mediante un valore alfanumerico che lo associa al numero interno della sua abitazione, ad esempio, *int01*, *int13*, *int04* indicano rispettivamente i condomini che abitano nell'appartamento con numero di interno *01*, *14* e *04*.
Come si evince dalla *Figura1*, nel file *parcheggi.txt* i 20 posti auto disponibili sono disposti su 4 file di 5 posti l'una, ciascuna fila è separata dall'altra da un corridoio, e ciascun posto può essere occupato correntemente da un condomino (in tal caso è indicato l'identificativo alfanumerico del un condomino) oppure libero (in tal caso è presente la stringa "_____" di cinque caratteri).

```
int01|int03|int02|_____|int07|

_____|int18|_____|_____|int06|

int10|_____|_____|_____|_____|

int05|_____|_____|_____|int13|
```

Figura 1. Il file *parcheggi.txt*.

```
int01:2
int02:3
int03:4
int04:13
int05:1
int06:5
int07:7
int08:8
....
```

Figura 2. Il file *assegnazioni.txt*.

Il file *assegnazioni.txt* riporta invece per ciascun condomino (identificato dal su valore alfanumerico) il corrispettivo posto auto pre-assegnato (si veda la *Figura2*).

Si vuole realizzare uno script perl **`amministratore.pl`** che verifica la congruenza dei posti auto effettivamente occupati dai condomini (e cioè quelli che appaiono nel file *parcheggi.txt*) con l'assegnazione dei posti predefinita per il condomini, e che è riportata nel file *assegnazioni.txt*. I nomi dei due file devono essere specificati come parametri sulla linea di comando.

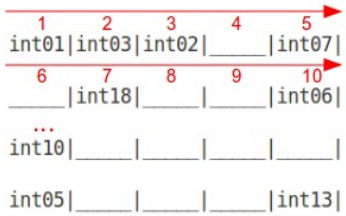
Lo script deve verificare che ciascun posto auto sia legittimamente occupato. In particolare, deve:

1. aprire in base ai nomi specificati da linea di comando i file dei *parcheggi* e delle *assegnazioni;*
2. verificare se ciascun posto occupato nel file *parcheggi.txt* sia effettivamente occupato dal

condomino associato a quel posto nel file *assegnazioni.txt;*

3. scrivere su standard output l'elenco dei condomini che stanno violando le norme, ovvero occupano abusivamente un posto auto;

N.B: La numerazione dei posti nel file *parcheggi.txt* è implicitamente assunta essere la seguente:



ESERCIZIO 2 (Programmazione multithread)

Si progetti una struttura dati per la gestione di una matrice in maniera tale che le operazioni di seguito specificate, nello scheletro di classe fornito, siano **thread-safe**. Il meccanismo di locking sulle celle della matrice deve essere organizzato in maniera tale da massimizzare la concorrenza, comunque evitando problemi di interferenza tra thread multipli.
Ad esempio, dovrebbe essere possibile che due thread distinti possano invocare rispettivamente `incrementaRiga(1)` e `incrementaRiga(2)` senza rallentamenti dovuti a problemi di mutua esclusione, mentre operazioni che interferiscono sulla stessa parte della matrice devono essere soggette ad opportune politiche di locking.
Qualsiasi altra struttura dati ritenuta necessaria per lo svolgimento dell'esercizio è a cura dello studente. 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 non thread-safe del software. Il lavoro può essere sviluppato o in Java o in C++.
ATTENZIONE: Non stiamo richiedendo di velocizzare l'esecuzione delle singole operazioni su matrice tramite parallelizzazione, ma di rendere queste operazioni THREAD-SAFE.

```
public class Matrice {

        int matrice[][];

    //...

        /**
         * Costruisce una matrice di dimensione N per M
         */
        Matrice (int N, int M)
        {
                //..
        }
        /**
         * Incrementa la riga i di 1 (elemento per elemento)
         */
        void incrementaRiga(int i)
        {

        }
        /**
         * Incrementa la colonna i di 1 (elemento per elemento)
         */
        void incrementaColonna(int i)
        {

        }
        /**
         * Restituisce la somma dei valori della colonna i
         */
   int sommaDiColonna(int i)
   {

   }
   /**
    * Restituisce la somma dei valori della riga i
    */
   int sommaDiRiga(int i)
   {

   }

}
```