

Corso di Sistemi Operativi e Reti, corso di Sistemi Operativi – 19 Feb. 2014 -

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.

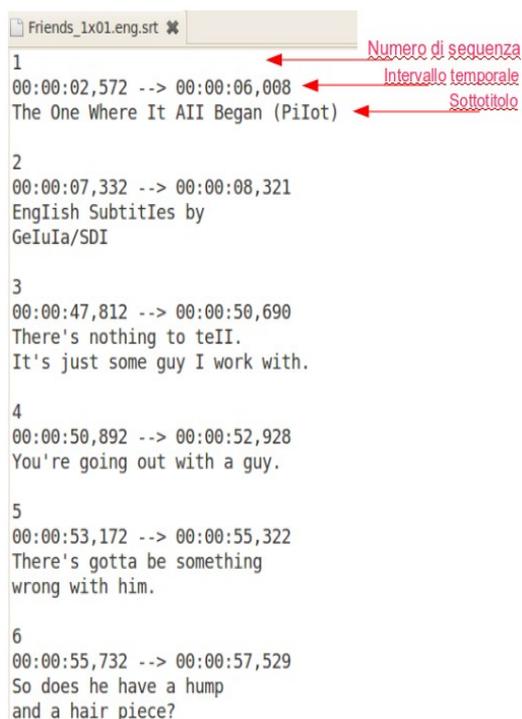
Si consiglia di salvare SPESSO il proprio lavoro.

ESERCIZIO 1 (Linguaggi di scripting)

Un file `.srt` è un file di testo contenente sottotitoli utilizzabili da diversi lettori multimediali. Il formato di un file con `.srt` è piuttosto semplice, include infatti:

- un *numero* che ne indica la sequenza
- un intervallo di *tempo* in cui il sottotitolo deve comparire
- il *sottotitolo* del testo
- una *riga vuota* per indicare l'inizio di un nuovo sottotitolo

Ad esempio, la **Figura 1** riporta un estratto del file `Friends_1x01.eng-2.srt`:



The image shows a screenshot of a text editor window titled "Friends_1x01.eng.srt". The content of the file is as follows:

```
1
00:00:02,572 --> 00:00:06,008
The One Where It All Began (PiIot)

2
00:00:07,332 --> 00:00:08,321
English Subtitles by
GeLuIa/SDI

3
00:00:47,812 --> 00:00:50,690
There's nothing to tell.
It's just some guy I work with.

4
00:00:50,892 --> 00:00:52,928
You're going out with a guy.

5
00:00:53,172 --> 00:00:55,322
There's gotta be something
wrong with him.

6
00:00:55,732 --> 00:00:57,529
So does he have a hump
and a hair piece?
```

Red arrows point from labels to specific parts of the first subtitle line:

- "Numero di sequenza" points to the number "1".
- "Intervallo temporale" points to the time range "00:00:02,572 --> 00:00:06,008".
- "Sottotitolo" points to the text "The One Where It All Began (PiIot)".

Figura 1. Estratto del file `Friends_1x01.eng-2.srt`.

Si scriva uno script perl che dati in input (da linea di comando) il nome di un *file* in formato *.srt* (ad esempio *Friends_1x01.eng.srt*) ed un *intervallo temporale* espresso in minuti (ad esempio *01-03*) sia capace di estrarre il **solo testo** dei dialoghi che si svolgono strettamente nell'intervallo temporale indicato. I dialoghi da estrarre corrispondono a quelli i cui minuti di inizio sono compresi tra il primo valore (nell'esempio, il minuto *01*) e strettamente compresi nel minuto di inizio di sequenza pari al secondo valore (nell'esempio, il minuto *03*).

Ad esempio, lo script invocato da linea di comando con gli argomenti: *Friends_1x01.eng.srt 01-03* deve estrarre il solo testo dei dialoghi con minuti di inizio pari a *01* e *02*, mentre il testo che si presenta a partire dal terzo minuto (*03*) deve essere escluso (perchè non strettamente compreso nell'intervallo indicato).

Il testo estratto deve essere scritto su un **nuovo file** il cui nome è ottenuto concatenando mediante il carattere “_” il nome del file passato da linea di comando con la stringa relativa all'intervallo temporale.

Ad esempio, lo script invocato da linea di comando con gli argomenti: *Friends_1x01.eng.srt 01-03* deve stampare il contenuto su un nuovo file di nome *Friends_1x01.eng_01-03.srt*.

Nota: Si assuma per semplicità che la durata in termini di minuti dei dialoghi presenti all'interno di ciascun file *.srt* non sia superiore ai 59 minuti. Pertanto, l'intervallo temporale indicato da linea di comando è univocamente rintracciabile all'interno del file dei sottotitoli.

ESERCIZIO 2 (Programmazione multithread; network programming)

Si deve implementare una struttura dati per la gestione di transazioni tra più conti bancari, dove le transazioni possono essere *simultanee*.

A tal proposito un **ContoBancario** è composto da:

1. Una variabile *Saldo*, indicante un importo in Euro;
2. Un elenco, in ordine temporale, delle ultime 50 transazioni effettuate sul conto. Una Transazione è composta da
 - a. Un ContoBancario sorgente;
 - b. Un ContoBancario di destinazione;
 - c. Un Valore in Euro;

Ad esempio, la *Transazione* $\langle A,B,10 \rangle$ indica il trasferimento di 10 Euro dal conto A al conto B.

La struttura dati **Banca** deve contenere al suo interno:

1. Una collezione di *ContiBancari* (si supponga di dover gestire circa un milione di conti bancari);
2. Il metodo *int getSaldo(C)*: restituisce l'attuale disponibilità sul conto C;
3. Il metodo *bool trasferisci(A,B,N)*: trasferisce N Euro dal conto A al conto B (dove A e B sono conti presenti nella collezione di Conti Bancari della Banca. Il metodo fallisce restituendo *false* se la disponibilità sul conto A è insufficiente ad effettuare l'operazione. Se la transazione viene effettuata, le disponibilità sui conti A e B vengono opportunamente aggiornate, nonché l'elenco delle ultime operazioni registrate in A e B.

I metodi *getSaldo* e *trasferisci* devono essere Thread-safe. Si assuma di prevedere un volume di accessi alla struttura dati di circa 1000 thread in contemporanea.

In ordine di priorità, il codice deve essere implementato:

1. garantendo **la mutua esclusione** nell'accesso ai dati condivisi **solo ove necessario**;
2. garantendo l'assenza di situazioni di stallo permanente (deadlock);
3. garantendo il massimo grado di parallelismo ed efficienza; Si assuma di prevedere un volume di accessi alla struttura dati di circa 1000 thread in contemporanea.
4. garantendo l'assenza di situazioni di stallo temporaneo (starvation).

Per i soli studenti del corso di Sistemi operativi e Reti:

Si supponga che una istanza della classe Banca sia disponibile su un server remoto, e si progetti il codice client-server necessario a effettuare le operazioni di *getSaldo* e *trasferisci* da una postazione remota.

E' parte integrante di questo esercizio completare le specifiche date nei punti non esplicitamente definiti, introducendo tutte le strutture dati che si ritengano necessarie, e risolvendo eventuali ambiguità. Non è consentito modificare il prototipo dei metodi se questo è stato fornito.

Si può svolgere questo esercizio in un qualsiasi linguaggio di programmazione a scelta dotato di costrutti di supporto alla programmazione multi-threaded (esempio, C++ con libreria JTC, Java).

E' consentito usare qualsiasi funzione di libreria di Java 6.

*Non è esplicitamente richiesto di scrivere un main() o di implementare esplicitamente dei thread di prova, anche se lo si suggerisce per testare il proprio codice prima della consegna (per evitare di bloccare il PC **non** lanciate realmente 1000 thread).*