Corso di Sistemi Operativi e Reti, corso di Sistemi Operativi – Novembre 2015

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".

Per il codice Java, **si consiglia di raggruppare tutto il proprio codice in un package dal nome "CognomeNomeMatricola",** secondo lo schema usato per rinominare la cartella "CognomeNomeMatricola".

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)

Si consideri un file di log *weblog.log*, che registra tutte le richieste HTTP fatte ad un server. Ciascuna riga del file *weblog.log* è del tipo:

```
site logName fullName [date:time GMToffset] "req file proto" status length referer user-agent
```

dove *site* rappresenta l'IP o il nome host che ha effettuato la richiesta HTTP; *logName* indica il login del client che ha effettuato la richiesta (oppure '-' se non specificato); *fullName* rappresenta il nome del client che ha effettuato la richiesta (oppure '-' se non specificato); *date* è la data della richiesta HTTP; *time* è l'ora della richiesta HTTP; *GMToffset* indica il fuso orario; req indica il tipo di richiesta http (ad esempio GET); ***file*** **è il path o il nome del file richiesto**; *proto* indica il tipo di protocollo usato per la richiesta; *status* è il codice di stato HTTP a 3 cifre restituito dal web server rappresentativo dello stato della richiesta; *length* è la lunghezza in byte dell'oggetto richiesto. Infine gli ultimi due campi *referer* e *user-agent* identificano la pagina che riferisce la risorsa richiesta e lo user agent (nome dell'applicazione, sistema operativo..).

Ad esempio, una riga del file *weblog.log* potrebbe essere la seguente:

```
151.74.62.177 - - [09/Nov/2011:11:13:15 +0100] "GET /ianni/PonteMareMonti.pdf HTTP/1.1"
302 578 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/534.50 (KHTML, like Gecko)
Version/5.1 Safari/534.50"
```

Parte1.

Si scriva uno script perl **analyzer.pl** capace di analizzare il file di log passato da linea di comando al fine di produrre delle statistiche. In particolare lo script deve:

1. leggere il file *weblog.log* passato da linea di comando
2. produrre un nuovo file di report che riporti per ciascuna risorsa (cioè *file* richiesto) il numero di richieste di accesso alla stessa (possibilmente ordinando i risultati in senso decrescente rispetto al numero di richieste)

Ad esempio, invocando lo script come:

```
perl analyzer.pl weblog.log
```

lo script produrrà un nuovo file con un contenuto del tipo:

```
/~ianni -> 46
/~ianni/ -> 23
/ianni/Reparto.pdf -> 21
/ianni/PonteMareMonti.pdf -> 18
/ianni/MainSedie.java -> 18
/~ianni/foafTiny.gif -> 14
/~ianni/mail.jpg -> 14
/~ianni/L_GB2.jpg -> 14
/ianni/risSO092011.pdf -> 7
/ianni/storage/ProdConsEmptyFullCondition.java -> 7
/ianni/storage/filosofiSempliciMain.java -> 6
/ianni/ReadWriteLock.java -> 6
/ianni/netkit-filesystem-i386-F5.1.patch3.tar.bz2 -> 6
…
…
```

Parte2.

Lo script dovrebbe essere in grado di gestire un secondo parametro da linea di comando, che rappresenta, sotto forma di stringa, l'estensione dei file (.zip, .pdf, .html, …) per i quali si vuole produrre la statistica. In questa modalità, infatti, lo script dovrebbe produrre la statistica, non genericamente per tutte le risorse (file) presenti in *weblog.log*, ma per i soli file con estensione indicata nel parametro da linea di comando.
Ad esempio, invocando lo script come:

```
perl analyzer.pl weblog.log "pdf"
```

lo script produrrà la statistica sul numero di richieste di accesso per i soli file di tipo pdf. L'output sarà del tipo

```
/ianni/Reparto.pdf --> 21
/ianni/PonteMareMonti.pdf --> 18
/ianni/risSO092011.pdf --> 7
/ianni/storage/Pizzeria.pdf --> 4
```

ESERCIZIO 2 (Programmazione multithread)

Si supponga di avere a disposizione una classe BlockingQueue4<T> che segue il prototipo standard dell'omonima classe Java BlockingQueue<T>, ma con il vincolo di poter essere creata con solo e soltanto 4 posti disponibili al suo interno.

Si progetti la classe BlockingQueueQualsiasi<T> (BQQ nel seguito) che consenta di creare delle BlockingQueue di dimensione qualsiasi, facendo uso al suo interno di sole BlockingQueue4<T>. Ad esempio, se si vuole creare una BQQ di 11 elementi, sarà necessario combinare almeno tre code da 4 (4*3=12, mentre due code da 4 non sarebbero sufficienti) elementi ciascuna in maniera opportuna.

I metodi della classe che devono obbligatoriamente implementati sono:

`public BlockingQueueQualsiasi<T>(int dimensione)` : crea una BlockingQueueQualsiasi di dimensione specificata.

`T take()` : estrae un elemento di tipo T dalla BQQ in oggetto. Si blocca se la BQQ è vuota, fino a che non venga inserito un altro elemento da un altro thread. L'ordine di estrazione deve rispettare l'ordine di inserimento.

`put(T elem)` : inserisce un elemento elem nella BQQ in oggetto. Si blocca se la BQQ dovesse essere piena, fino a che un altro thread non estrae un elemento liberando un posto.

E' parte integrante *di questo esercizio completare le specifiche date nei punti non esplicitamente definiti, introducendo o estendendo 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 o successivi. Non è esplicitamente richiesto di scrivere un main() o di implementare esplicitamente del codice di prova, anche se lo si suggerisce per testare il proprio codice prima della consegna.