

Corso di Sistemi Operativi e Reti, corso di Sistemi Operativi – Gennaio 2016

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.

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

Non saranno valutate prove d'esame il cui codice è stato messo nel package di default.

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. Punti 0-10)

Il file *auth.log* registra una serie di messaggi relativi alla sicurezza del sistema, come ad esempio tentativi di login. Si scriva uno script perl che analizzi il file *auth.log* al fine di produrre una statistica sui tentativi di login falliti relativi all'accesso SSH. Per far ciò, lo script deve individuare nel file, le righe del tipo:

..... Failed password for invalid user username from *ip_address*

dove *username* è una stringa usata come username per l'accesso e *ip_address* è l'indirizzo IP utilizzato. Lo script deve individuare, per tutti gli indirizzi IP (*ip_address*) che compaiono nelle righe individuate, il numero di tentativi di login fallimentari effettuati usando quello stesso indirizzo IP.

Ad esempio, l'indirizzo 185.130.5.234 compare 5 volte in righe della tipologia indicata. Lo script deve stampare a video i risultati di questa statistica, cioè deve stampare per ogni indirizzo IP coinvolto in tentativi di attacco brute-force, il numero dei tentativi condotti. Inoltre, lo script dovrebbe collezionare e scrivere su un nuovo file l'insieme delle *username* utilizzate in corrispondenza dei tentativi fallimentari, ovvero le *username* che compaiono in corrispondenza delle righe della tipologia indicata.

ESERCIZIO 2 (Programmazione multithread. Punti: 0-20)

Un `Job` rappresenta un compito da eseguire a più riprese in base al suo stato; un job può essere ATTIVO (Job da eseguire ripetutamente) o INATTIVO (Job quiescente, da non eseguire, salvo cambi di stato). Un job rappresenta una porzione di codice del tutto passiva, che per essere eseguita necessita di un thread che invochi il metodo `esegui` del job stesso.

Bisogna progettare uno `Scheduler` che, data una collezione di Jobs, ne programmi l'esecuzione.

Un `Job` include le seguenti caratteristiche:

- Il metodo `esegui()` che deve essere chiamato per eseguire il Job corrispondente, e che restituisce il nuovo stato del Job stesso (0=MANTIENI ATTIVO QUESTO JOB, 1=DISATTIVA QUESTO JOB);
- Un valore di priorità iniziale e un valore di priorità attuale, rappresentati da un numero intero positivo (0=massima priorità) e inizialmente impostati allo stesso valore;

Lo `Scheduler` è costituito da un thread con il seguente ciclo di vita:

1. Scelta di un job J da eseguire tra quelli attivi;
2. Esecuzione di J, ottenuta invocando il metodo `J.esegui()`;
3. Ripianificazione di J in base al valore restituito da `J.esegui()`; (0 = RISCHEDULA POICHE' IL JOB E' ANCORA ATTIVO, 1 = NON RISCHEDULARE POICHE' IL JOB E' INATTIVO)
4. Ricalcolo delle priorità dei Job;
5. Ritorno al punto 1;

Ogni ripetizione dei punti da 1 a 5 è detta **TURNO**.

In particolare:

- La scelta del prossimo Job (Punto 1 di cui sopra) deve essere fatta partendo dai job di priorità attuale più elevata e con politica Round Robin (=a turni ripetuti periodicamente) tra i job di pari priorità; E' possibile delegare l'esecuzione di più job in parallelo facendo uso di un massimo di 4 Thread creati e gestiti opportunamente;
- Ricalcolo delle priorità (Punto 4 di cui sopra): sia `RECALC` una variabile interna al thread scheduler. Se passano più di `RECALC` turni senza che un certo job J venga eseguito, allora la priorità attuale di J deve essere decrementata di 1. Quando infine J viene eseguito, la sua priorità attuale deve essere reimpostata alla priorità iniziale.
- Lo scheduler deve possedere metodi pubblici per aggiungere, rimuovere, cambiare lo stato di un job tra ATTIVO e INATTIVO, e aggiornarne il valore di priorità iniziale e attuale; deve essere inoltre disponibile un metodo per alterare `RECALC`. Tali metodi devono essere invocabili da thread diversi dal thread scheduler, e devono pertanto essere sincronizzati in maniera opportuna, rendendoli così thread-safe.

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.