NON SPEGNERE IL PC A FINE ESAME

Corso di Sistemi Operativi e Reti

Prova scritta di SETTEMBRE 2019

ISTRUZIONI

1. **Rinomina** la cartella chiamata "Cognome-Nome-Matricola" che hai trovato sul Desktop e in cui hai trovato questa traccia, sostituendo "Cognome" "Nome" e "Matricola" con i tuoi dati personali e **lasciando i trattini**; se hai un doppio nome oppure un doppio cognome dovrai chiamare la cartella come in questo esempio:
    a. DeLuca-MarcoGiovanni-199999
2. **Carica** tutto il materiale didattico che vorrai usare sul Desktop; puoi farlo solo nei primi 5 minuti della prova;
3. **Svolgi** il compito; lascia tutto il sorgente che hai prodotto nella cartella di cui al punto 1;
4. Quando hai finito lascia la postazione facendo logout,

senza spegnere il PC.

SALVA SPESSO il tuo lavoro

NON SPEGNERE IL PC A FINE ESAME

ESERCIZIO 1 (Programmazione multithread. Punti: 0-20)

Si progetti un insieme di thread che cooperano per costruire una torre fatta da strati alternati di mattoni e cemento. Un mattone è simboleggiato da "*", mentre il cemento è simboleggiato da "-". La torre è costituita da un array di stringhe alternate tra "***" (strato di tre mattoni) e "---" (strato di tre pezzi di cemento)", e comincia alla base con uno strato di cemento, ad esempio:

```
Torre[3] = '***';
Torre[2] = '---';
Torre[1] = '***';
Torre[0] = '---';
```

rappresenta una torre valida, dove la base della torre è rappresentata dall'elemento 0 dell'array. La torre viene costruita incrementalmente, a partire dalla sua base, da dei Thread "Mattonatori" e da dei Thread "Cementatori".

Un thread mattonatore può aggiungere un mattone per volta alla torre (e cioè un solo asterisco), mentre un thread cementatore può aggiungere un pezzetto di cemento alla volta (e cioè un trattino); l'operazione può essere effettuata solo se in quel momento la costruzione della torre necessita di mattoni o, rispettivamente, di cemento. Mettere un mattone richiede 50ms di tempo, mentre mettere un pezzetto di cemento richiede 25ms.

Si progetti dunque una classe chiamata Torre, che esporti il metodo pubblico:

```
makeTorre(H : int, M : int, C : int)
```

Tale metodo deve costruire una torre alta H strati, facendo uso di un numero M di mattonatori e C di Cementatori, e infine restituire la torre una volta questa è stata completata.

Le strutture dati devono essere implementate garantendo la necessaria thread safety; non è ammesso progettare strutture dati con potenziali situazioni di deadlock; è opportuno migliorare l'accessibilità concorrente alle strutture dati ed evitare, se presenti, situazioni di starvation.

NON SPEGNERE IL PC A FINE ESAME

CI SONO DEI PUNTI AMBIGUI NELLA TRACCIA? COMPLETA TU

È parte integrante di questo esercizio completare le specifiche date nei punti non esplicitamente definiti, introducendo o estendendo tutte le strutture dati laddove si ritenga necessario, e risolvendo eventuali ambiguità.

POSSO CAMBIARE IL PROTOTIPO DEI METODI RICHIESTI? NO

Non è consentito modificare il prototipo dei metodi se questo è stato fornito. Potete aggiungere qualsivoglia campo e metodo di servizio, e qualsivoglia classe ausiliaria, ma NON variare l'interfaccia dei metodi pubblici già specificati.

CHE LINGUAGGIO POSSO USARE? PYTHON 3.X; oppure JAVA 7 o successivo

Il linguaggio da utilizzare per l'implementazione è Python 3, o in alternativa, Java. È consentito usare qualsiasi funzione di libreria di Python 3.X o di Java 7 e versioni successive.

MA IL MAIN() LO DEVO SCRIVERE? E I THREAD DI PROVA? SI

Sebbene non saranno oggetto di valutazione, è obbligatorio scrivere un `main()` e implementare esplicitamente del codice di prova, che è comunque necessario per testare il proprio codice prima della consegna.

ESERCIZIO 2 (Linguaggi di scripting. Punti 0-10)

Si scriva uno script `PERL` dal nome `auth.pl` in grado di fornire analisi dettagliate riguardanti eventuali accessi non autorizzati registrati nel file `auth.log` di Linux. Un esempio di file di log è fornito, a titolo di esempio, in allegato alla traccia. Il file si compone di più righe del seguente tipo:

Sep 2 06:25:12 server sshd[15334]: Failed password for invalid user lyssa from 103.228.55.79 port 45462 ssh2
Sep 2 06:25:12 server sshd[15334]: Received disconnect from 103.228.55.79 port 45462:11: Bye Bye [preauth]
Sep 2 06:25:12 server sshd[15334]: Disconnected from invalid user lyssa 103.228.55.79 port 45462 [preauth]
Sep 2 06:25:16 server sshd[15336]: Invalid user test from 45.55.35.40 port 33438
Sep 2 06:25:16 server sshd[15336]: pam_unix(sshd:auth): check pass; user unknown
Sep 2 06:25:16 server sshd[15336]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=45.55.35.40

Come è possibile notare, la prima riga è composta dalla data e dall'ora in cui si è verificato il tentativo di connessione (**Sep 2 06:25:12**) succeduta da altri parametri e dalla stringa **"Failed password for invalid user"** seguita dal nome utente che ha tentato la connessione (nell'esempio **lyssa**), dall'indirizzo IP dal quale è provenuta la connessione (nell'esempio **103.228.55.79**) ed infine dalla porta sorgente (nel caso in esame **45462**).

Di seguito è riportata la sinossi del comando da implementare

```
./auth.pl [-ip|-user username] nome_file
```

Lo script, a seconda dell'opzione inserita in fase di esecuzione, eseguirà una tra seguenti operazioni:

1. **`./auth.pl -ip nome_file`** : lo script legge il file di accesso il cui nome viene passato su linea di comando tramite il parametro `nome_file`; per ogni riga contenente la stringa **"Failed password for invalid user",** salva l'indirizzo IP sorgente in una apposita struttura dati e, per ogni IP riscontrato, conta i tentativi di connessione non autorizzati provenienti dallo stesso indirizzo (ovvero le occorrenze dello stesso indirizzo IP nelle righe contenenti la stringa **"Failed password for invalid user"**). Lo script termina stampando su STDOUT la coppia `IP - NUMERO_OCCORRENZE` in ordine **decrescente**.
2. **`./auth.pl -user utente nome_file`** : lo script legge il file di accessi il cui nome viene passato da linea di comando tramite il parametro `nome_file`; per ogni riga contenente la stringa **"Failed password for invalid user"** conta gli accessi effettuati dallo user dal nome `utente` e memorizza tutte le date in cui tale utente ha effettuato i tentativi di accesso. Lo script termina stampando su file la coppia `NOME_UTENTE - NUMERO_ACCESSI` nella prima riga del file, seguita da **N** righe contenenti le date in cui si è tentato l'accesso.

NON SPEGNERE IL PC A FINE ESAME