Corso di Sistemi Operativi e Reti, corso di Sistemi Operativi

Prova scritta di Novembre 2016

ISTRUZIONI

1. **Rinomina** la cartella chiamata "CognomeNomeMatricola" che hai trovato sul Desktop e in cui hai trovato questa traccia, sostituendo "Cognome" "Nome" e "Matricola" con i tuoi dati personali;
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, ma **senza spegnere** il PC.

5. **SALVA SPESSO il tuo lavoro**

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.

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

Bisogna progettare una libreria per la gestione delle scommesse online. La libreria gestisce delle partite di calcio sui quali vari scommettitori possono effettuare scommesse in concorrenza; i bookmaker possono aggiornare in tempo reale le corrispondenti quote, mentre gli scommettitori possono fare, sempre in tempo reale, le loro scommesse. La gestione del sistema di scommesse deve essere pertanto thread-safe.

Di seguito si fornisce una versione iniziale delle classi Partita e Scommessa (tali classi possono essere arricchite a piacere):

```
public class Partita
{
	//
	// Possibili risultati della partita
	//
	final static int RIS1 = 0;
	final static int RISX = 1;
	final static int RIS2 = 2;
	//
	// Risultato attuale. Da considerare definitivo se terminato=true;
	//
	int risultatoCorrente = 1;
	//
	// quote dei bookmaker. 0= 'uno', 1='X', 2= 'due'
	//
	double quote[] = new double[3];
	//
	// dice se la partita è in corso o meno
	//
	boolean terminata = false;

}
class Scommessa
{
	Partita p;			// Partita su cui si è scommesso
	double importoScommesso;		// Quanto si è scommesso in EUR
	int risultatoScommesso;		// Risultato su cui si è scommesso (1, X, 2)
	double quota;			// Quota sulla quale si è scommesso
}
```

Ogni partita può avere tre possibili risultati (1, X, 2), a ciascuno dei quali è associata una quota corrispondente a quanto si vincerebbe puntando 1 EUR sul risultato corrispondente (per esempio, se la quota per il risultato "X" è 2.00, si vincerebbero 4 EUR puntando 2 EUR, posto che il risultato finale della partita sia realmente "X"). Risultati e quote possono cambiare fintantoché la partita non è terminata. Non si può scommettere a evento concluso.

Una scommessa viene fatta su una specifica partita, per la quale si punta un certo `importo` su uno dei possibili risultati finali (1, X, 2), a una certa `quota` che viene congelata al momento della scommessa. Si può determinare se una scommessa è stata vinta solo quando la partita corrispondente è `terminata`. In tal caso, se il risultato dell'evento corrisponde al risultato scommesso, la vincita corrispondente sarà pari a `importoScommesso*quota`.

Fissate le specifiche di cui sopra, bisogna implementare i seguenti metodi per la classe Partita:

`double[] leggiQuote()`
Restituisce le quote correnti per la partita in oggetto.

`void setQuote(double nuoveQuote[])`
Imposta le quote per la partita ai nuovi valori impostati.

`int getRisultato() , void setRisultato(int nuovoRisultato)`
Rispettivamente, legge e imposta il risultato corrente della partita.

`Scommessa scommetti(double importo, int risultatoScommesso, double qmin)`
Registra una scommessa sulla partita, di importo pari a `importo`, scommettendo sul risultato `risultatoScommesso`, alla quota `qmin`. Viene restituita una scommessa valida solo se la partita **non è terminata** e la quota a cui si scommette **non è superiore** al valore della quota correntemente accettata dai bookmaker. Ad esempio, se si scommette 1 EUR alla quota di 2.50, sul risultato "X" per la partita P, questa scommessa non può essere accettata se, P.quote[1] == 1.2.
Per cui, se la scommessa non è valida, viene restituito il valore `null`.

`double attendiFinale(Scommessa s)`
pone il thread chiamante in attesa fintantoché la partita non termina. Restituisce un double pari all'importo vinto (zero se la scommessa risulta persa, e cioè quando il `risultatoScommesso` non corrisponde con il risultato finale). Se `s` non è una scommessa fatta su `this`, viene sollevata un'eccezione.

`void terminaPartita()`
dichiara terminata la partita, per cui non deve essere più possibile scommettere, aggiornare le quote e i risultati.

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. Non è consentito modificare il prototipo dei metodi se questo è stato fornito.*

Il linguaggio da utilizzare per l'implementazione è Java. È consentito usare qualsiasi funzione di libreria di Java 7 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.

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

Si consideri il comando 'df' disponibile nei sistemi Linux (output di esempio riportato):

```
Filesystem                                          1K-blocks      Used Available Use% Mounted on
rootfs                                               19228180  15782444   3250388  83% /
udev                                                    10240         0     10240   0% /dev
tmpfs                                                  103380      1676    101704   2% /run
/dev/disk/by-uuid/a6d2d3ec-f91c-49a9-8bb7-f62de501e09d  19228180  15782444   3250388  83% /
tmpfs                                                    5120         0      5120   0% /run/lock
tmpfs                                                  592300         0    592300   0% /run/shm
/dev/md2                                            124960732 112233824   6379232  95% /home
/dev/md1                                             96124812  58815580  32426284  65% /usr/files
```

Si implementi uno script perl chiamato `MONITOR.PL` da invocare nel formato:

`MONITOR.PL <nomefilesystem> <intervallo>`

Facendo uso di ripetute invocazioni del comando '`df`', tale comando deve monitorare ogni `<intervallo>` secondi l'occupazione su disco di `<nomefilesystem>`, stampando un rigo informativo iniziale, e un rigo informativo ogni qual volta il numero di blocchi usati dovesse cambiare su tale filesystem.

Ad esempio, invocando lo script come:

`MONITOR.PL /dev/md2 5`

Si vedrà immediatamente a video la scritta

```
/dev/md2                                            124960732 112233824   6379232  95% /home
```

Un eventuale cancellazione o aggiunta di file sul filesystem monitorato, cambierà il numero di blocchi allocati, e in tal caso dovrebbe comparire un ulteriore linea, ad es:

```
/dev/md2                                            124960732 112233830   6379226  95% /home
```

Lo script gira finché non viene interrotto con CTRL-C.

Suggerimento: in Perl è possibile sospendere l'esecuzione per X secondi usando la primitiva 'sleep X'