Corso di Sistemi Operativi e Reti

Corso di Sistemi Operativi

Prova scritta di Gennaio 2017

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, **senza spegnere il PC**.

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)

Si deve progettare una *BlockingQueue con ritardo*, che chiameremo `DelayedBlockingQueue`. Una `DelayedBlockingQueue` accumula gli elementi al suo interno nel momento in cui si esegue l'operazione di inserimento, ma questi elementi non sono resi disponibili prima di un tempo fissato *d*.

Ad esempio, supponiamo di avere una `DelayedBlockingQueue` che può contenere al massimo 10 elementi, inizialmente vuota, e con ritardo (*d*) di 5000 millisecondi (5 secondi). Se si inserisce un elemento E con l'operazione `put(E)`, l'operazione termina immediatamente, ma una successiva `take()` si bloccherà in attesa dell'elemento fino allo scadere dei cinque secondi. I 5 secondi sono conteggiati a partire dal momento dell'inserimento dell'elemento.

La classe da progettare deve supportare elementi qualsiasi usando i costrutti generics; I metodi di cui deve essere dotata la `DelayedBlockingQueue` devono essere i seguenti:

`put(T e)` : inserisce l'elemento e nella coda, con il ritardo di default. Il metodo si blocca se la coda dovesse essere piena.

`put(T e, int d)` : inserisce l'elemento `e` nella coda, con il ritardo `d`, espresso in millisecondi. Il metodo si blocca in attesa di uno slot libero se la coda dovesse essere piena.

`T take()` : preleva e restituisce dalla coda un elemento. Si blocca in attesa se non ci sono disponibili elementi nella coda il cui ritardo è scaduto.

`long poll()` : restituisce 0 se ci sono elementi già prelevabili, -1 se la coda è vuota, o altrimenti il tempo che resta (in millisecondi) prima che almeno un elemento sia disponibile al prelievo.

`setDelay(long d)`: imposta il ritardo di default a *d* millisecondi.

`long getDelay()`: restituisce il ritardo di default.

Si noti che, secondo le specifiche date, la `DelayedBlockingQueue` non segue necessariamente la politica FIFO. Ad esempio, se si eseguono in immediata sequenza gli inserimenti `put(a,300)` e `put(b,100)`, l'elemento `b` sarà disponibile con 200ms di anticipo rispetto all'elemento `a`, nonostante l'elemento `a` sia stato inserito prima.

L'accesso a una `DelayedBlockingQueue` deve essere, ovviamente, thread safe.

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)

Il comando `env`, se eseguito senza nessun parametro, stampa (su Standard Output) la lista di tutti i valori delle variabili di ambiente (environment variables) presenti nel sistema.
Ad esempio:

```
TERM=xterm-256color
PATH=/usr/local/bin:/usr/bin:/bin
PWD=/home/utente
HOME=/home/utente
_=/usr/bin/env
```

Tra queste variabili un ruolo particolare lo ha la variabile `PATH` perché specifica le directory (separate da '`:`') dove sono localizzati i programmi eseguibili immediatamente a disposizione.

L'esercizio consiste nel trovare le dimensioni in bytes di tutti i file e le cartelle presenti "direttamente" all'interno di ciascuna delle directory specificate nella variabile `PATH`, di sommarli tra loro e di visualizzare su Standard Output, nel formato riportato nell'Esempio, le seguenti informazioni:

I. il valore di tale somma per ogni directory del `PATH` (con la relativa unità di misura)
II. la somma totale di tutte le directory del `PATH` (con la relativa unità di misura)

Per reperire la dimensione di un file si può usare il comando shell che si preferisce.
Non è necessario visitare ricorsivamente le sottodirectory
È possibile usare qualsiasi comando shell disponibile.

Esempio:

Supponendo di avere questi file (con le relative dimensioni in byte) nelle directory del `PATH`**:**

`/usr/local/bin`

dir1	4096
dir2	0
file1	2475
file2	742

`/usr/bin`

dir7	48
file79	216
file15	31

`/bin`

file10	12
file11	913
file80	13159
file15	31

Si dovrà visualizzare su Standard Output:

```
/usr/local/bin : 7313 bytes
/usr/bin : 295 bytes
/usr/local/bin : 14115 bytes
TOTALE : 21723 bytes
```

Si noti che bisogna sommare tutti gli elementi; se un elemento è presente in più di una directory (in questo caso l'elemento "file15" è presente in `/usr/bin` e in `/bin`), bisogna sommarlo in ognuna di esse.

Nota bene: si possono "saltare" tutte le directory del `PATH` per cui si ottengono problemi di accesso ("Access denied")