NON SPEGNERE IL PC A FINE ESAME

Corso di Sistemi Operativi e Reti

Prova scritta di FEBBRAIO 2020

PRIMA PARTE

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

SALVA SPESSO

NON SPEGNERE IL PC A FINE ESAME

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

Si deve progettare una struttura dati thread-safe, detta `FairLock`. Un `FairLock` eredita il proprio comportamento da quello di un normale Lock rientrante, ma garantisce che l'ordine di acquisizione del lock corrisponda all'ordine temporale in cui ciascun thread ha invocato l'operazione di acquire. E' inoltre disponibile: *1)* l'operazione `urgentAcquire`, che invece dovrebbe consentire di acquisire il `FairLock` prima possibile, scavalcando dunque qualsiasi thread sia già in attesa di prendere il lock; e *2)* un sistema di gestione della starvation configurabile.

I metodi di cui deve essere dotato un `FairLock` sono:

`acquire(self)`
Acquisisce il `FairLock` se questo è libero. Altrimenti si pone in attesa bloccante finchè non arriva il proprio turno di acquisizione al lock.

`urgentAcquire(self)`
Acquisisce il `FairLock` se questo è libero. Altrimenti si pone in attesa bloccante finchè non arriva il proprio turno di acquisizione al lock. Ai thread che invocano questo metodo deve essere data priorità nell'acquisizione del lock secondo le regole menzionate sotto.

`release(self)`
Rilascia il proprio accesso al `FairLock`. Se ci sono dei thread in attesa di acquisire il lock, l'accesso deve essere garantito nell'ordine: ai thread che hanno invocato più recentemente una `urgentAcquire`, dal più recente al meno recente; ai thread che hanno invocato una normale acquire, partendo dal thread che aspetta da più tempo fino al thread che aspetta più di recente.

`setStarvationControl(self, n : int )`
Dal momento che le `urgentAcquire` hanno sempre priorità sulle `acquire` normali, deve essere possibile configurare la gestione della starvation usando questo metodo. In particolare il parametro `n` indica il numero di `urgentAcquire` consecutive, tra quelle in attesa, che possono essere smaltite prima di gestire una eventuale `acquire` in attesa. Se `n=0`, il lock non effettua alcun controllo della starvation e smaltisce tutte le `urgentAcquire` prioritariamente.

SALVA SPESSO

NON SPEGNERE IL PC A FINE ESAME

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.

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 **funzionante**, che è comunque necessario per testare il proprio codice prima della consegna.

SALVA SPESSO

NON SPEGNERE IL PC A FINE ESAME

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

Si scriva uno script `Perl` dal nome `subtitles.pl` in grado di effettuare modifiche su uno o più file di sottotitoli presenti ricorsivamente all'interno di directory e subdirectory.

Nota bene: Il nome di ogni file contiene un "identificativo" che ha lo scopo di evidenziare la lingua in cui il film o la serie TV è sottotitolato.
Ad esempio, la puntata:

`The.Big.Bang.Theory.S10E10.The.Property.Collision.720p.WEB-DL.DD5.1.H264-R2D2.`**`ita`**`.srt`

è sottotitolata in lingua **italiana** in quanto prima dell'estensione del file (`.srt`) è presente la stringa `.ita`.
Per semplicità, si può assumere che, per ogni file, la lingua è sempre specificata direttamente prima dell'estensione. Se non esplicitamente presente, si prendano come estensione di lingua, le **3 lettere precedenti** il `.srt` (`.ita.srt`, `.eng.srt`, `.fra.srt`, …)

Di seguito è riportata la sinossi del comando da implementare

```
./subtitles.pl OPTIONS path_to_directory [str]
```

In particolare, lo script può ricevere in input fino ad un **massimo** di **3 parametri**:

- `OPTIONS`**:** è un parametro **obbligatorio**. Le opzioni possibili sono soltanto 2 e sono di seguito elencate.
  - Se l'opzione inserita è **`-c`** lo script conterà, per ogni diversa lingua, il numero di file di sottotitoli presenti. Infine, stamperà il numero di file di sottotitoli presenti per ogni diversa lingua sia **`stdout`** che su **`file`** in ordine decrescente di numero. A parità di numero file sarà necessario ordinare anche alfabeticamente il nome della lingua del sottotitolo..
  - Se l'opzione inserita è **`-r`**, lo script dovrà ricevere tra gli argomenti un ulteriore parametro dal seguente formato: **`/str1/str2/`** . Lo script cercherà in tutte le sottocartelle i file che conterranno all'interno del loro nome la stringa **`str1`** e la sostituirà con la stringa **`str2`**.
- `path_to_directory`**:** è un parametro obbligatorio che specifica il percorso della cartella entro cui cercare i file di sottotitoli o i file da rinominare (a seconda dell'opzione inserita).

Hint: è possibile utilizzare i comandi **`find path_to_directory`** ed **`ls -R path_to_directory`** al fine di ottenere la lista di file e cartelle presenti all'interno della una directory specificata.

ATTENZIONE: ESEMPI DI ESECUZIONE A PAGINA SUCCESSIVA

SALVA SPESSO

NON SPEGNERE IL PC A FINE ESAME

Esempio di esecuzione 1: (NON È RICHIESTA LA STAMPA DELL'ALBERO DELLE CARTELLE. È SOLO A SCOPO DIMOSTRATIVO)

Esecuzione Script:

```
perl subtitles.pl -r rinominare/ /.txt/.ciao/
```

```
francesco@francesco:~/Scrivania/Esami/2020/Febbraio2020$ tree rinominare
rinominare
├── folder1
│   ├── doc1.docx
│   ├── doc1.txt
│   └── subfolder1
│       ├── doc1.docx
│       └── doc1.txt
├── folder2
│   ├── doc1.txt
│   └── doc2.txt
├── folder3
│   ├── doc1.input
│   └── doc3.docx
└── folder4
    ├── doc1.doc
    └── doc1.log

5 directories, 10 files
francesco@francesco:~/Scrivania/Esami/2020/Febbraio2020$ perl subtitles.pl -r rinominare/ /.txt/.ciao/
rinominare/
├── folder1
│   ├── doc1.ciao
│   ├── doc1.docx
│   └── subfolder1
│       ├── doc1.ciao
│       └── doc1.docx
├── folder2
│   ├── doc1.ciao
│   └── doc2.ciao
├── folder3
│   ├── doc1.input
│   └── doc3.docx
└── folder4
    ├── doc1.doc
    └── doc1.log

5 directories, 10 files
francesco@francesco:~/Scrivania/Esami/2020/Febbraio2020$
```

Esempio di esecuzione 2:

Esecuzione Script:

```
perl subtitles.pl -c serieTV/
```

```
francesco@francesco:~/Scrivania/Esami/2020/Febbraio2020$ perl subtitles.pl -c serieTV/
ita --> 78
eng --> 54
fra --> 1
```

SALVA SPESSO