Corso di Sistemi Operativi e Reti

Prova scritta 22 GIUGNO 2021

ISTRUZIONI PER CHI È IN PRESENZA:

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

ISTRUZIONI PER CHI SI TROVA ONLINE:

1. **Questo file contiene il testo che ti è stato dato ieri, incluso il codice;**
2. **Mantieni a tutto schermo** questo file per tutta la durata della prova; puoi scorrere liberamente tra le sue pagine, ma non puoi cambiare applicazione;
3. **Firma** preliminarmente il foglio che userai per la consegna con nome cognome e matricola;
4. **Svolgi** il compito; puoi usare solo carta, penna e il tuo cervello;
5. **Aiutati** con i numeri di linea per indicare le eventuali modifiche che vorresti fare al codice che ti è stato dato.
6. **Alla scadenza** termina *immediatamente* di scrivere, e attendi di essere chiamato, pena l'esclusione dalla prova;
7. **Quando è il tuo turno** mostra il foglio ben visibile in webcam, e poi metti una foto dello stesso foglio in una chat privata Microsoft Teams con il prof.

ESERCIZIO 1, TURNO 1 - PROGRAMMAZIONE MULTITHREADED

Punto 1:

Si osservi che l'istanza di thread `SorgenteVetture` (il campo `Traghetto.sorgente`) viene creata all'interno del metodo `CaricaVetture`, ma rimane in esecuzione anche quando il metodo `CaricaVetture` termina. Si modifichi opportunamente il codice in maniera tale che `Traghetto.sorgente` si arresti correttamente allorquando la fase di carico del traghetto si conclude.

Punto 2:

Si modifichi il codice modificando il comportamento dei thread di tipo `Parcheggiatore`, e della `SorgenteVetture`.

Devono essere introdotti i thread di tipo `ParcheggiaAutobus` e `ParcheggiaAutomobili`; I quattro parcheggiatori originari devono essere ora suddivisi in 2 `ParcheggiaAutobus` e 2 `ParcheggiaAutomobili`. La classe `SorgenteVetture` deve essere arricchita con i metodi `getAutobus` e `getAutomobile`. Il metodo `getAutobus` va in attesa bloccante fino a che non è presente un Autobus come prima vettura da estrarre nell'ordine di estrazione FIFO di `SorgenteVetture`, infine, appena questo è disponibile, si restituisce l'autobus primo nell'ordine della coda. Analogamente, il metodo `getAutomobile` va in attesa bloccante fino a che non è presente un Automobile come prima vettura da estrarre nell'ordine FIFO di `SorgenteVetture`, restituendo un'Automobile appena essa è disponibile.

N.B. Nel caso non ci si ricordi come si verifica il tipo run-time di un oggetto, nel verificare se una vettura è automobile o autobus ci si può aiutare guardando il campo `Vettura.size`.

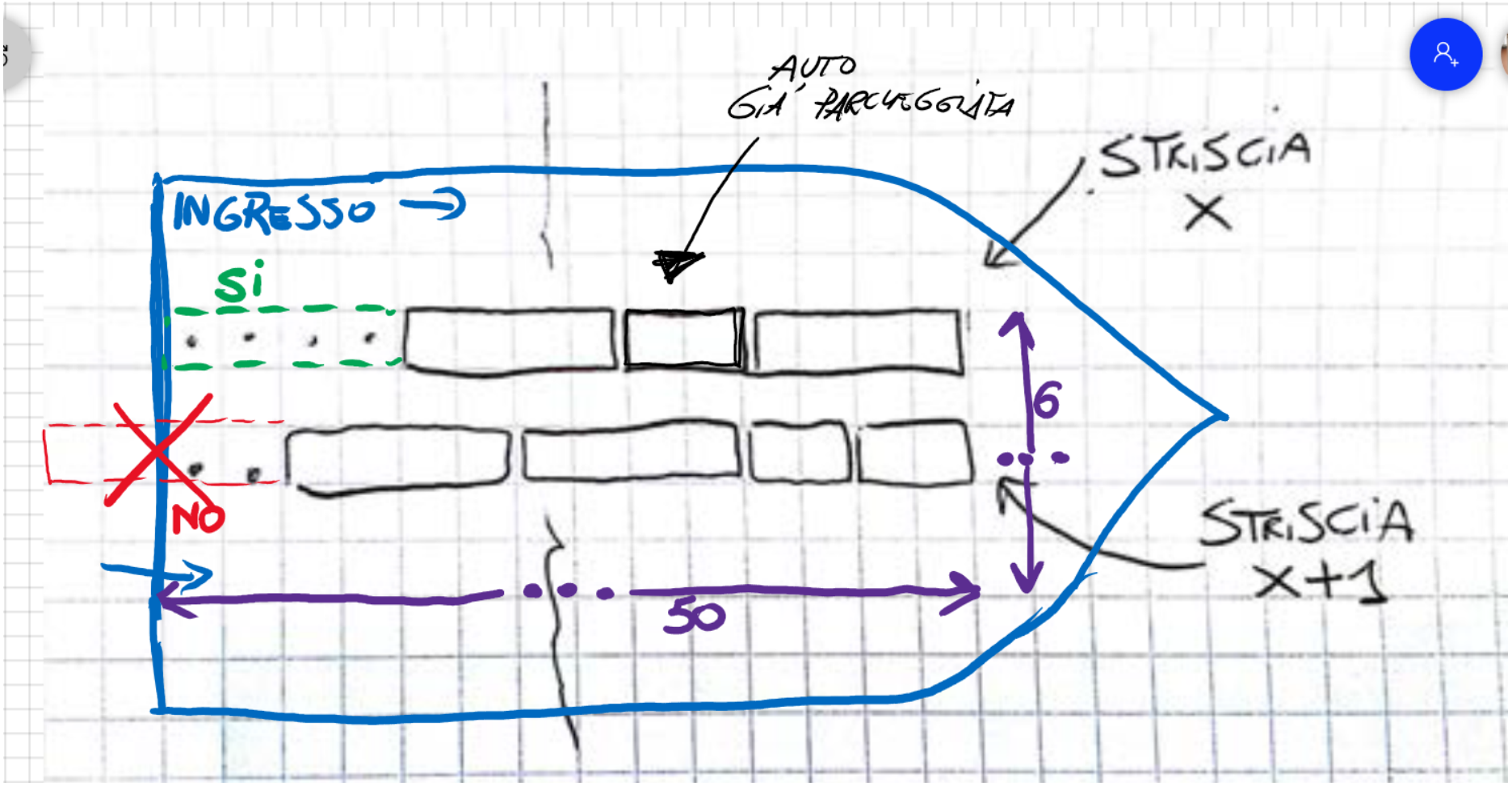
MATERIALE PER LA PROVA SULLA PROGRAMMAZIONE MULTI-THREADED

Il codice fornito simula la procedura di carico di un `Traghetto`. Un traghetto è composto da 6 strisce di 50 “celle” ciascuna. Le celle possono essere occupate da delle `Vetture`. Un Traghetto è gestito da 4 `Parcheggiatori` (implementati usando 4 thread dello stesso tipo), i quali sistemano in contemporanea le Vetture che via via si presentano all’imbarco. Una `Vettura` può essere di due tipi: `Automobile`, di dimensione 1x2 celle, o `Autobus`, di dimensione 1x4 celle. Le vetture possono essere posizionate nel parcheggio solo in posizione orizzontale all’interno di una striscia, e solo se in tale striscia è disponibile un numero sufficiente di celle, e nell’ordine di posizionamento nella striscia che viene scelta.

Ad esempio, supponiamo che le auto entrino da sinistra e che vengano parcheggiate riempiendo il traghetto partendo dal fondo di quest’ultimo (sulla destra).

Se la situazione attuale nel traghetto è la seguente:

SALVA SPESSO



E ci si trova a dover posizionare un Autobus, lo si potrà sistemare nella Striscia X immediatamente a sinistra dell'ultima vettura posizionata in questa striscia, poiché ci sono 4 celle libere. Invece, se si deve posizionare una Automobile, sarà possibile utilizzare sia la Striscia X che la Striscia X+1, poiché entrambe hanno 2 celle libere.

Una `SorgenteMacchine` produce delle vetture di tipo casuale da posizionare all'interno del Traghetto, simulando così una possibile sequenza di vetture da caricare sul Traghetto.

Il Traghetto e i Parcheggiatori sono invece implementati in maniera tale che questi ultimi sistemino le vetture prodotte dalla SorgenteMacchine secondo le regole prescritte e con la massima contemporaneità possibile. In particolare il metodo `CaricaVetture` della classe Traghetto si occupa di monitorare l'attività dei quattro parcheggiatori e termina nel momento in cui non è più possibile caricare ulteriori Vetture di alcun tipo.

Segue il codice sorgente.

SALVA SPESSO

```python
#!/usr/bin/env python

from threading import Thread,RLock,Barrier
from queue import Queue
from time import sleep
from random import random,randint

class Vettura(object):

    def __init__(self):
        self.size = 0

    def printSize(self):
        print(self.size)

class Automobile(Vettura):
    def __init__(self):
        super(Automobile, self).__init__()
        self.size = 2

class Autobus(Vettura):
    def __init__(self):
        super(Autobus, self).__init__()
        self.size = 4

class SorgenteVetture(Thread):
    vetture = Queue(10)

    #
    #  Questo metodo viene chiamato quando si vuol prendere una Vettura prodotta dalla SorgenteVettura
    #
    def getVettura(self):
        return self.vetture.get()

    def run(self):
            while True:
```

```
                sleep(random()*2)
                #  Genera una vettura a intervalli casuali
                v = Automobile() if randint(0,1) == 0 else Autobus()
                self.vetture.put(v)


class Striscia(object):


    def __init__(self):
        self.size = 50
        self.l = RLock()

    def put(self, v):
        self.size -= v.size

    def getPostiLiberi(self):
        return self.size

    #
    #  Data una vettura v, prova a posizionarla in questa striscia.
    #  Restituisce true se operazione avvenuta con successo.
    #
    def provaAInserire(self, v):
        with self.l:
            if self.getPostiLiberi() >= v.size:
                self.put(v)
                return True
            else:
                return False


class Parcheggiatore(Thread):

    def __init__(self, t, id):
        super(Parcheggiatore, self).__init__()
        self.traghetto = t
```

```
        self.id = id
        print(f"{self.id}")

    def run(self):
        possoParcheggiare = True
        while possoParcheggiare:
            v = self.traghetto.sorgente.getVettura()
            trovato = False
            for i in range(6):
                #
                # Ogni parcheggiatore ha una sua striscia preferita che dipende da self.id
                # La striscia preferita viene provata prima di tutte le altre
                #
                if self.traghetto.strisce[(i + self.id) % 6].provaAInserire(v):
                    trovato = True
                    print(f"S:{(i+self.id)%6}-{str(self.id)*v.size}")
                    break
            #
            # Un parcheggiatore che non trova posto non parcheggia la vettura corrente
            # e cessa tutte le attività
            #
            if not trovato:
                possoParcheggiare = False
        self.traghetto.b.wait()

class Traghetto:

    def __init__(self):
        self.sorgente = SorgenteVetture()
        self.b = Barrier(5)
        self.strisce = [Striscia() for _ in range(6)]

    def caricaTraghetto(self):
        self.sorgente.start()
        for i in range(4):
            Parcheggiatore(self, i).start()
        self.b.wait()
```

```
if __name__ == '__main__':
    siremarOne = Traghetto()
    siremarOne.caricaTraghetto()
```

ESERCIZIO 2, TURNO 1 - PERL

Si scriva uno script Perl dal nome `process_monitor.pl` che monitora in tempo reale lo stato di occupazione di risorse da parte dei vari utenti/processi del sistema. Per ottenere uno snapshot delle risorse utilizzate in tempo reale è necessario eseguire il comando `top -b -n1` il quale restituisce in output alcune informazioni relative all'utilizzo della CPU e della memoria da parte di utenti e processi.
L'output fornito è simile al seguente:

```
top - 16:05:28 up 4 days, 22:15,  1 user,  load average: 0.00, 0.05, 0.45
Tasks: 232 total,   1 running, 231 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.7 us,  0.7 sy,  0.0 ni, 97.8 id,  0.0 wa,  0.0 hi,  0.7 si,  0.0 st
MiB Mem :  11962.8 total,    617.4 free,   1125.0 used,  10220.4 buff/cache
MiB Swap:   2048.0 total,   1549.2 free,    498.8 used.  10511.2 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
      1 frances+  20   0  170104   8428   5684 S   7.8   3.5   1:51.19 java
      2 root      20   0       0      0      0 S   0.1   0.4   0:00.14 kthreadd
      4 frances+   0 -20       0      0      0 I   9.5   4.2   0:00.00 java
      6 root       0 -20       0      0      0 I   0.5   0.0   0:00.00 kworker/0:0H-kblockd
      0 systemd+  20   0   24220  10252   6060 S   0.0   0.1   0:42.63 systemd-resolve
```

Lo script deve essere eseguito obbligatoriamente con le seguenti 2 opzioni da linea di comando:

./process_monitor.pl [-u | -p] [-c | -m]

La prima opzione può essere `-u` oppure `-p` e serve ad indicare allo script il criterio di raggruppamento da eseguire.

- Se la prima opzione è `-u` bisognerà raggruppare le linee per nome **utente**
- Se la prima opzione è `-p` bisognerà raggruppare le linee per nome del **processo**

La seconda opzione può essere `-c` oppure `-m` e serve a definire il criterio di ordinamento che lo script dovrà eseguire.

- Se la seconda opzione è `-c` bisognerà **ordinare in ordine decrescente** le linee per valore totale di CPU occupata da un determinato **utente o processo** (in base a come specificato dalla prima opzione)
- Se la seconda opzione è `-m` bisognerà **ordinare in ordine decrescente** le linee per valore totale di memoria ram occupata da un determinato **utente o processo** (in base a come specificato dalla prima opzione)

Supponiamo che lo script venga eseguito con le opzioni `-p` e `-m`. Lo script dovrà calcolare la somma della memoria occupata raggruppando per il nome del processo. Nel caso dell'esempio di sopra, il primo output sarebbe il seguente:

`java → 7.7% Mem`
`kthreadd → 0.4% Mem`
`systemd-resolve → 0.1% Mem`
`kthreadd → 0.0% Mem`
`kworker/0:0H-kblockd → 0.0% Mem`

Se invece le opzioni inserite fossero state `-u` e `-c`, il primo output sarebbe stato il seguente:

`frances+ → 17.3% CPU`
`root → 0.6% CPU`
`systemd+ → 0.0% CPU`

`N.B.: Le stampe sono ordinate sempre in ordine decrescente di valore. A parità di valore, bisogna ordinare anche anche per nome utente o processo.`

`ATTENZIONE:`
Lo script continuerà la sua esecuzione controllando e aggiornando l'output ogni **2 secondi**[1] finché l'utente non lo terminerà tramite la sequenza di comandi **CTRL+C.** Inoltre, tutte le stampe, oltre ad essere riportate su standard output (STDOUT), **dovranno essere anche salvate su un file** dal nome `output.log`.

è possibile usare la funzione `sleep($time_in_seconds)` di Perl per gestire l'intervallo tra le iterazioni dello script.