Corso di Sistemi Operativi e Reti

Prova scritta del 13 NOVEMBRE 2020

ESERCIZIO 1 - PROGRAMMAZIONE MULTITHREAD

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.

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 riga che ti sono stati forniti per indicare eventuali modifiche che vorresti fare al codice che ti abbiamo fornito;
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 - QUESITO DA RISOLVERE

Modificare la classe RoundRobinLock in maniera tale che sia possibile marcare uno tra gli ID possibili come l'ID "Presidente". Il Presidente può acquisire il RoundRobinLock con priorità rispetto agli altri ID. I metodi da aggiungere sono:

`SetPresident(self, id : int)`
Imposta l'ID del "Presidente" al valore specificato. Il "Presidente" di default è assegnato all'ID 0.

`UrgentAcquire(self)`

Tale metodo deve essere implementato in accordo ai seguenti due casi:

1. Se il RoundRobinLock è libero, lo assegna immediatamente all'id "Presidente";
2. Se il RoundRobinLock è occupato, bisogna attendere che si liberi. Nessun thread di ID diverso da quello del presidente, incluso l'ID del turno corrente, può nel frattempo acquisire il lock. Una volta libero, il RoundRobinLock viene assegnato all'ID presidente, *senza* rispettare le regole sui turni a rotazione.

In entrambi i casi di cui sopra, quando il presidente rilascia definitivamente il lock, bisogna riprendere la turnazione normale dal punto in cui era stata sospesa.

Esempio. Supponiamo N=10, con ID presidente = 0.

Supponiamo che il RoundRobinLock sia nello stato:

`self.inAttesa[4] = 2, self.inAttesa[5] = 4,  self.turnoCorrente=3, self.possessori = 1`

Una eventuale chiamata di UrgentAcquire() si porrebbe in attesa bloccante, ed eventuali nuovi thread con ID=3 verrebbero messi in attesa nel caso cercassero di acquisire il lock. Nel momento in cui il microfono viene liberato dai thread con ID=3, bisogna cedere immediatamente il RoundRobinLock ai thread in attesa con ID=0 (per i quali è stata invocata una UrgentAcquire()). Quando il "Presidente" rilascerà il lock, bisognerà riprendere la turnazione precedentemente sospesa, passando il microfono ai thread di ID=4.

MATERIALE PER LA PROVA SULLA PROGRAMMAZIONE MULTI-THREADED

Si deve progettare una struttura dati thread-safe, detta `RoundRobinLock`. Un `RoundRobinLock` può essere usato per gestire l'accesso a un unico microfono in una tavola rotonda di N partecipanti. Ogni partecipante può prendere il microfono se questo è libero, mentre si pone in attesa se il microfono è occupato; l'accesso al microfono è soggetto a turnificazione. Quando un microfono viene rilasciato questo viene ceduto alla propria destra al primo dei partecipanti che ha chiesto la parola.

Un altro possibile uso del RoundRobinLock è quando ci sono N gruppi distinti di fruitori di una risorsa, che però non possono usare la risorsa contemporaneamente. Si immagini una ciotola di cibo per animali, dalla quale possono mangiare contemporaneamente tutti i gatti, tutti i cani, tutte le galline, ma non gatti, cani e galline contemporaneamente. Sono fornite due implementazioni del RoundRobinLock, una che evita i problemi di possibile starvation, e una versione più semplice senza controllo della starvation.

I metodi di cui deve essere è dotata la struttura dati sono:

`__init__(self, N : int)`

Costruisce un RoundRobinLock da N categorie distinte di partecipanti. Ad esempio, se N=5, ci potrebbero essere 5 speaker seduti al tavolo, che condividono un microfono unico il cui accesso è disciplinato attraverso un RoundRobinLock; oppure potrebbero esserci 5 categorie di fruitori di una ciotola unica, il cui accesso è disciplinato attraverso il RoundRobinLock: cani (N=0) , gatti (N=1), galline (N=2), maialini (N=3), capre (N=4).

`acquire(self,id : int)`

Prova ad acquisire il RoundRobinLock, dichiarando un id partecipante che può andare da 0 a N-1. Il lock può essere acquisito subito se questo è libero, o anche se è occupato da altri partecipanti che hanno dichiarato lo stesso id del chiamante. In tutti gli altri casi è necessario porsi in attesa bloccante finchè non arriva il proprio turno di acquisire il lock.

`release(self,id : int)`

Rilascia il proprio accesso al RoundRobinLock dichiarando il proprio id partecipante. Se, grazie all'operazione di rilascio appena effettuata, non ci sono più partecipanti con lo stesso id del chiamante ad occupare il RoundRobinLock, è necessario garantire che il turno di accesso sia garantito, in ordine di priorità, ai partecipanti con id consecutivi, cominciando dai partecipanti in attesa con identificativo (id+1)%N, e passando eventualmente ai successivi id in attesa.

```
 1  from threading import Thread,RLock,Condition
 2  from random import randint
 3
 4  debug = True
 5  deepDebug = False
 6  def dprint(s):
 7      if deepDebug:
 8          print(s)
 9
10  class RoundRobinLock:
11
12      def __init__(self,N : int):
13          self.nturni = N
14          self.lock = RLock()
15          self.conditions = [Condition(self.lock) for _ in range(0,N)]
16          self.inAttesa = [0 for _ in range(0,N)]
17          self.turnoCorrente = 0
18          self.possessori = 0
19
20
21      #
22          #  Verifica che tra self.turnoCorrente e id non ci sia nessun thread in attesa con idTurno  tale che
self.turnoCorrente < idTurno < id
23      #
24      def _possoPrendereMicrofono(self,id):
25
26          if id == self.turnoCorrente:
27              return True
28          for i in range(1,self.nturni-1):
29                      idTurno = (id - i) % self.nturni
30                      if idTurno == self.turnoCorrente:
31                          return True
32                      if self.inAttesa[idTurno] > 0:
33                          return False
34
35
36      def acquire(self,id : int):
```

```
    37          with self.lock:
    38              self.inAttesa[id] += 1
    39              while( self.possessori > 0 and self.turnoCorrente != id or
    40                     self.possessori == 0 and not self._possoPrendereMicrofono(id)):
    41                  dprint(f"Waiting {id}")
    42                  self.conditions[id].wait()
    43                  dprint(f"Waking {id} - Turno {self.turnoCorrente}")
    44
    45              #
    46               # Se mi trovo in questo punto del codice, ho la garanzia che self.possessori == 0 e nessun thread di
altro id ha bisogno del turno
    47              # dunque prendo il turno
    48              #
    49              dprint(f"Getting Lock {id}")
    50
    51              self.turnoCorrente = id
    52              self.inAttesa[id] -= 1
    53              self.possessori += 1
    54              if debug:
    55                  self.__print__()
    56
    57
    58      def release(self, id : int):
    59          with self.lock:
    60              self.possessori -= 1
    61              if self.possessori == 0:
    62                  for i in range(1,self.nturni):
    63                      turno = (id + i) % self.nturni
    64                      if self.inAttesa[turno] > 0:
    65                          self.turnoCorrente = turno
    66                          self.conditions[turno].notifyAll()
    67                          dprint(f"Notifying {turno}")
    68                          break
    69              if debug:
    70                  self.__print__()
    71
    72      def __print__(self):
```

```
        with self.lock:
            print("=" * self.turnoCorrente + "|@@|" + "=" * (self.nturni - self.turnoCorrente -1) )
            for l in range(0,max(max(self.inAttesa),self.possessori)):
                o = ''
                for t in range(0,self.nturni):
                    if self.turnoCorrente == t:
                        if self.possessori > l:
                            o = o + "|o"
                        else:
                            o = o + "|-"
                    if self.inAttesa[t] > l:
                        o = o + "*"
                    else:
                        o = o + "-"
                    if self.turnoCorrente == t:
                        o = o + "|"
                print (o)
            print("")

class RoundRobinLockStarvationMitigation(RoundRobinLock):

    SOGLIASTARVATION = 5

    def __init__(self,N : int):
        super().__init__(N)
        self.consecutiveOwners = 0


    def acquire(self,id : int):
        with self.lock:
            self.inAttesa[id] += 1
            while( self.possessori > 0 and self.turnoCorrente != id
                   or
                   self.possessori == 0 and not self._possoPrendereMicrofono(id)
                   or
                               self.turnoCorrente == id and self.consecutiveOwners > self.SOGLIASTARVATION and
max(self.inAttesa) > 0
```

```
                 ):
                self.conditions[id].wait()

            self.turnoCorrente = id
            self.inAttesa[id] -= 1
            self.possessori += 1
            self.consecutiveOwners += 1
            if debug:
                self.__print__()


    def release(self,id : int):
        with self.lock:
            self.possessori -= 1
            if self.possessori == 0:
                for i in range(1,self.nturni):
                    turno = (id + i) % self.nturni
                    if self.inAttesa[turno] > 0:
                        self.turnoCorrente = turno
                        self.consecutiveOwners = 0
                        self.conditions[turno].notifyAll()
                        break


class RoundRobinLockConCoda(RoundRobinLock):

    def __init__(self,N : int):
        pass

    def acquire(self,id : int):
        pass

    def release(self,id : int):
        pass


class Animale(Thread):
```

```
146
147     def __init__(self,id: int, idTurno : int, R : RoundRobinLock):
148         super().__init__()
149         self.idTurno = idTurno
150         self.iterazioni = 10
151         self.lock = R
152
153     def run(self):
154         while(self.iterazioni > 0):
155                 self.iterazioni -= 1
156                 self.lock.acquire(self.idTurno)
157                 self.lock.__print__()
158                 self.lock.release(self.idTurno)
159
160
161 NGruppi = 10
162 R = RoundRobinLockStarvationMitigation(NGruppi)
163 #R = RoundRobinLock(NGruppi)
164 for i in range(0,60):
165     Animale(i,randint(0,NGruppi-1),R).start()
166
```