Corso di Sistemi Operativi e Reti

Prova scritta del 13 NOVEMBRE 2020

ESERCIZI 1 e 2 - MATERIALE PRELIMINARE E ISTRUZIONI

ISTRUZIONI

In questo documento trovi:

1. **La traccia di un esercizio sulla programmazione multi-threaded insieme con la sua soluzione commentata.** Fino al momento dell'esame puoi analizzare questo codice da solo, in compagnia, facendo uso di internet o di qualsiasi altro materiale. Puoi fare girare il codice, puoi modificarlo, fino a che non lo hai capito a fondo. Per comodità, a questo file è allegato anche il sorgente in file di testo separato.
2. **Alcune informazioni preliminari** sull'esercizio da scrivere in Perl **.**

COME SI SVOLGERA' L'ESAME

L'esame si svolgerà in due round separati, uno per ciascun esercizio.
Per l'esercizio sulla programmazione multi-threaded, ti verrà passato attraverso Microsoft Teams un file che contiene il testo che ti è stato dato il giorno prima, incluso il codice: all'interno di questo file, troverai l'esercizio da svolgere. L'esercizio richiederà di modificare il codice che già conosci secondo alcune specifiche date.
Per l'esercizio sulla programmazione scripting, ti verrà passato attraverso Microsoft Teams il testo di un esercizio. L'esercizio richiederà di lavorare sui comandi il cui output ti è stato mostrato in questo documento.

Per entrambi gli esercizi:

1. **Dovrai mantenere a tutto schermo** il file della traccia per tutta la durata della prova; potrai scorrere liberamente tra le pagine, ma non potrai cambiare applicazione;
2. **Il compito va svolto su carta. Firma** preliminarmente il foglio che userai per la consegna con nome cognome e matricola;
3. **Svolgi** il compito; potrai usare solo carta, penna e il tuo cervello;
4. **Aiutati** con i numeri di riga che ti sono stati forniti per indicare eventuali modifiche che vorresti fare al codice che ti abbiamo fornito;
5. **Alla scadenza** dovrai terminare *immediatamente* di scrivere, e attendere di essere chiamato, pena l'esclusione dalla prova;
6. **Non toccare la camera che ti inquadra** fino a quando non ti verrà chiesto dai prof;
7. **Quando sarà il tuo turno** dovrai mostrare il foglio con il tuo elaborato ben visibile in webcam, e in seguito inviare una foto dello stesso foglio in una chat privata Microsoft Teams con il prof.
8. **Le prove saranno videoregistrate** e il contenuto distrutto a chiusura dell'appello.

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 a turno, 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.

```
001: from threading import Thread,RLock,Condition
002: from random import randint
003:
004: debug = True
005:
006: class RoundRobinLock:
007:
008:     def __init__(self,N : int):
009:         self.nturni = N
010:         self.lock = RLock()
011:         self.conditions = [Condition(self.lock) for _ in range(0,N)]
012:         self.inAttesa = [0 for _ in range(0,N)]
013:         self.turnoCorrente = 0
014:         self.possessori = 0
015:
016: #
017: # Non c'è bisogno di particolare attenzione alla gestione del primo accesso
018: #
019:
020:     def acquire(self,id : int):
021:         with self.lock:
022:             self.inAttesa[id] += 1
023:             while( self.possessori > 0 and self.turnoCorrente != id):
024:                 self.conditions[id].wait()
025:
026:             self.inAttesa[id] -= 1
027:             self.possessori += 1
028:             if debug:
029:                 self.__print__()
030:
031:
032:     def release(self,id : int):
033:         with self.lock:
034:             self.possessori -= 1
035:             if self.possessori == 0: # and self.inAttesa[self.turnoCorrente] ==0:
036:                 for i in range(1,self.nturni):
037:                     turno = (id + i) % self.nturni
```

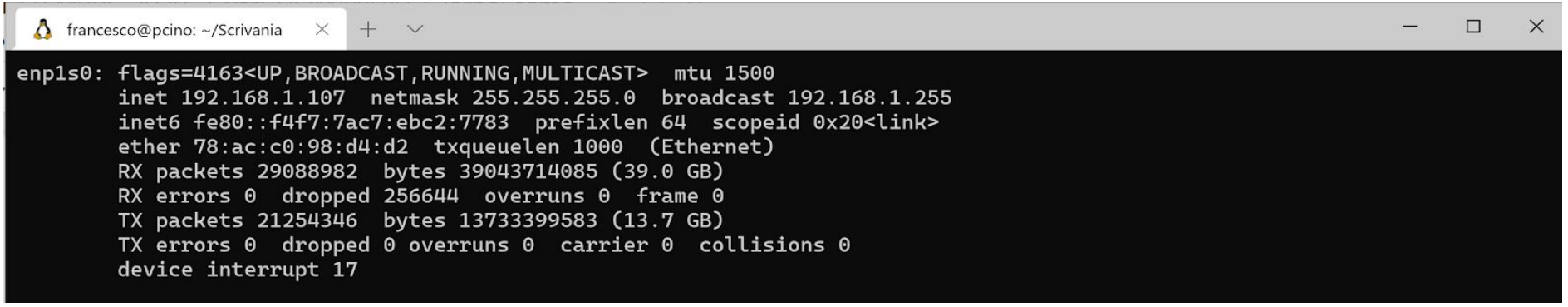
```
038:                     if self.inAttesa[turno] > 0:
039:                         self.turnoCorrente = turno
040:                         self.conditions[turno].notifyAll()
041:                         break
042:             if debug:
043:                 self.__print__()
044: 
045:     def __print__(self):
046:         with self.lock:
047:             print("=" * self.turnoCorrente + "|@@|" + "=" * (self.nturni - self.turnoCorrente -1) )
048:             for l in range(0,max(max(self.inAttesa),self.possessori)):
049:                 o = ''
050:                 for t in range(0,self.nturni):
051:                     if self.turnoCorrente == t:
052:                         if self.possessori > l:
053:                             o = o + "|o"
054:                         else:
055:                             o = o + "|-"
056:                     if self.inAttesa[t] > l:
057:                         o = o + "*"
058:                     else:
059:                         o = o + "-"
060:                     if self.turnoCorrente == t:
061:                         o = o + "|"
062:                 print (o)
063:             print("")
064: 
065: class RoundRobinLockStarvationMitigation(RoundRobinLock):
066: 
067:     SOGLIASTARVATION = 5
068: 
069:     def __init__(self,N : int):
070:         super().__init__(N)
071:         self.consecutiveOwners = 0
072: 
073: 
074:     def acquire(self,id : int):
```

```
075:         with self.lock:
076:             self.inAttesa[id] += 1
077:             while( self.possessori > 0 and
078:                    self.turnoCorrente != id or
079:                    self.turnoCorrente == id and
080:                    self.consecutiveOwners > self.SOGLIASTARVATION and
081:                    max(self.inAttesa) > 0
082:                  ):
083:                 self.conditions[id].wait()
084: 
085:             self.inAttesa[id] -= 1
086:             self.possessori += 1
087:             self.consecutiveOwners += 1
088:             if debug:
089:                 self.__print__()
090: 
091: 
092:     def release(self,id : int):
093:         with self.lock:
094:             self.possessori -= 1
095:             if self.possessori == 0:
096:                 for i in range(1,self.nturni):
097:                     turno = (id + i) % self.nturni
098:                     if self.inAttesa[turno] > 0:
099:                         self.turnoCorrente = turno
100:                         self.consecutiveOwners = 0
101:                         self.conditions[turno].notifyAll()
102:                         break
103: 
104: 
105: class RoundRobinLockConCoda(RoundRobinLock):
106: 
107:     def __init__(self,N : int):
108:         pass
109: 
110:     def acquire(self,id : int):
111:         pass
```

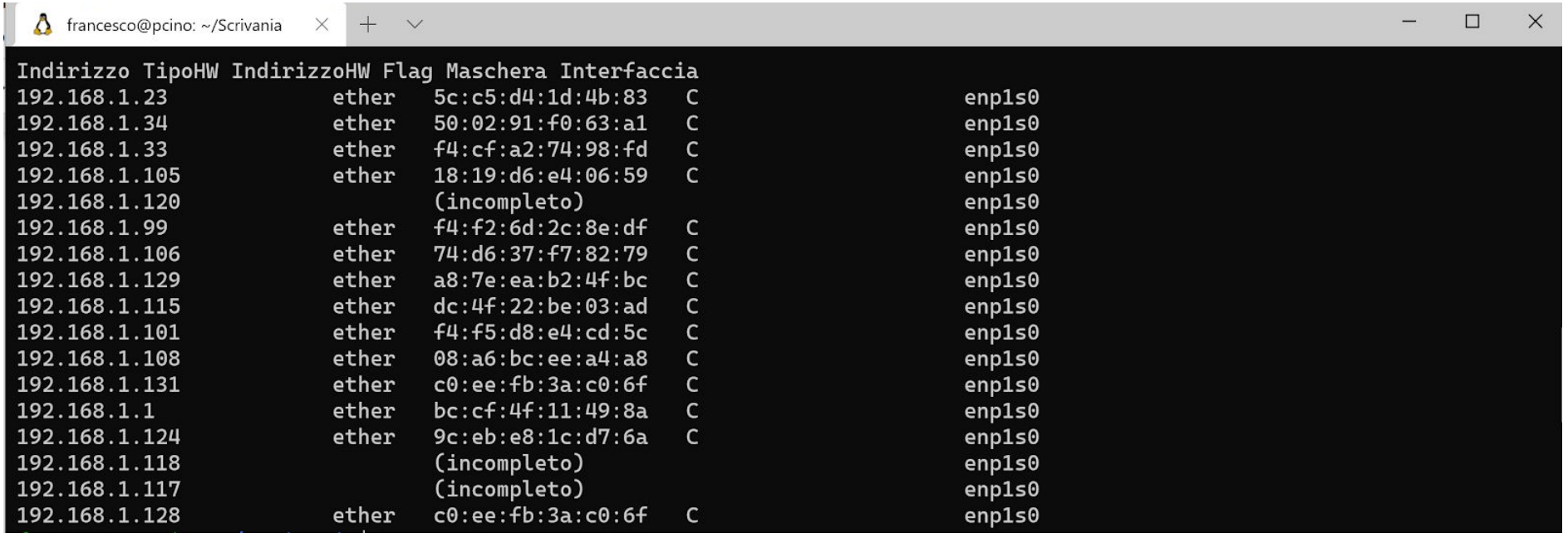
```
112:
113:     def release(self,id : int):
114:         pass
115:
116:
117: class Animale(Thread):
118:
119:     def __init__(self,id: int, idTurno : int, R : RoundRobinLock):
120:         super().__init__()
121:         self.idTurno = idTurno
122:         self.iterazioni = 1000
123:         self.lock = R
124:
125:     def run(self):
126:         while(self.iterazioni > 0):
127:                 self.iterazioni -= 1
128:                 self.lock.acquire(self.idTurno)
129:                 #self.lock.__print__()
130:                 self.lock.release(self.idTurno)
131:
132:
133: NGruppi = 5
134: R = RoundRobinLockStarvationMitigation(NGruppi)
135: #R = RoundRobinLock(NGruppi)
136: for i in range(0,60):
137:     Animale(i,randint(0,NGruppi-1),R).start()
138:
139:
140:
```

MATERIALE PRELIMINARE PER LA PROVA DI PROGRAMMAZIONE IN PERL

1. Output comando 1



2. Output comando 2



3. Output comando 3