Corso di Sistemi Operativi e Reti

Prova scritta del 16 SETTEMBRE 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 **.**

MATERIALE PER LA PROVA SULLA PROGRAMMAZIONE MULTI-THREADED

Il codice fornito implementa una classe che incapsula un dato accessibile con la tradizionale modalità Read/Write lock. Un meccanismo di controllo della starvation impone che uno scrittore non attenda più di 5 operazioni di locking in lettura prima di ottenere accesso in scrittura. I metodi offerti dalla classe DatoCondiviso sono:

`acquireReadLock(self):`
Acquisisce il lock in lettura sul dato condiviso, secondo le normali politiche di accesso in lettura, ma con controllo della starvation.
`releaseReadLock(self):`
Rilascia il lock in lettura sul dato condiviso, secondo le normali politiche di accesso in lettura.
`acquireWriteLock(self):`
Acquisisce il lock in scrittura sul dato condiviso, secondo le normali politiche di accesso in scrittura, ma con controllo della starvation.
`releaseWriteLock(self):`
Rilascia il lock in scrittura sul dato condiviso, secondo le normali politiche di accesso in scrittura.

`getDato(self):` Restituisce il valore del dato condiviso.
`setDato(self, v):` Imposta a v il valore del dato condiviso.

```
from threading import Thread,RLock,Condition
from random import random
from time import sleep

#
# Funzione di stampa sincronizzata
#
plock = RLock()
def prints(s):
    plock.acquire()
    print(s)
    plock.release()

class DatoCondiviso:
    SOGLIAGIRI = 5

    def __init__(self,v):
        super().__init__(v)
        self.numScrittoriInAttesa = 0
        self.numGiriSenzaScrittori = 0

    def acquireReadLock(self):
        self.lock.acquire()
        while self.ceUnoScrittore or \
              (self.numScrittoriInAttesa > 0 and self.numGiriSenzaScrittori > self.SOGLIAGIRI):
            self.condition.wait()
        self.numLettori += 1
        #
        #        * Il contatore viene incrementato solo se effettivamente ci sono
        #        * scrittori in attesa.
        #
        if self.numScrittoriInAttesa > 0:
            self.numGiriSenzaScrittori += 1
        self.lock.release()
```

```
def releaseReadLock(self):
        self.lock.acquire()
        self.numLettori -= 1
        #
        #   Nella versione senza starvation, possono esserci anche dei lettori in attesa.
        #   E' necessario
        #   dunque svegliare tutti.
        #
        if self.numLettori == 0:
            self.condition.notify_all()
        self.lock.release()

    def acquireWriteLock(self):
        self.lock.acquire()
        self.numScrittoriInAttesa += 1
        while self.numLettori > 0 or self.ceUnoScrittore:
            self.condition.wait()
        self.ceUnoScrittore = True
        self.numScrittoriInAttesa -= 1
        self.numGiriSenzaScrittori = 0
        self.lock.release()

    def releaseWriteLock(self):
        self.lock.acquire()
        self.ceUnoScrittore = False
        self.condition.notify_all()
        self.lock.release()


class Scrittore(Thread):

    maxIterations = 1000

    def __init__(self, i, dc):
        super().__init__()
        self.id = i
        self.dc = dc
```

```
        self.iterations = 0

    def run(self):
        while self.iterations < self.maxIterations:
            prints("Lo scrittore %d chiede di scrivere." % self.id)
            self.dc.acquireWriteLock()
            prints("Lo scrittore %d comincia a scrivere." % self.id )
            sleep(random())
            self.dc.setDato(self.id)
            prints("Lo scrittore %d ha scritto." % self.id)
            self.dc.releaseWriteLock()
            prints("Lo scrittore %d termina di scrivere." % self.id)
            sleep(random() * 5)
            self.iterations += 1


class Lettore(Thread):
    maxIterations = 100

    def __init__(self, i, dc):
        super().__init__()
        self.id = i
        self.dc = dc
        self.iterations = 0

    def run(self):
        while self.iterations < self.maxIterations:
            prints("Il lettore %d Chiede di leggere." % self.id)
            self.dc.acquireReadLock()
            prints("Il lettore %d Comincia a leggere." % self.id)
            sleep(random())
            prints("Il lettore %d legge." % self.dc.getDato())
            self.dc.releaseReadLock()
            prints("Il lettore %d termina di leggere." % self.id)
            sleep(random() * 5)
            self.iterations += 1
```

```
if __name__ == '__main__':
        dc = DatoCondiviso(999)

        NUMS = 5
        NUML = 5
        scrittori = [Scrittore(i,dc) for i in range(NUMS)]
        lettori = [Lettore(i,dc) for i in range(NUML)]
        for s in scrittori:
            s.start()
        for l in lettori:
            l.start()
```

PROGRAMMAZIONE IN PERL - MATERIALE PRELIMINARE

All’interno dell’esercizio verrà utilizzato il comando shell `du`, normalmente utilizzato per controllare le informazioni sull'utilizzo del disco di file e directory su una macchina.