Corso di Sistemi Operativi e Reti

Prova scritta dell'1 SETTEMBRE 2020

ESERCIZI 1 e 2 - MATERIALE PRELIMINARE E ISTRUZIONI

ISTRUZIONI

1. **Questo documento è diviso in tre parti.**
2. **Nella prima parte trovi la traccia di un esercizio 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.
3. **Nella seconda parte di questo documento** trovi una domanda tipo, molto simile a quella che effettivamente ti verrà posta il giorno dell'esame. Puoi provare a svolgere questa domanda tipo per capire se sei in grado di sostenere la prova reale, ricordando che avrai a disposizione mezz'ora di tempo.
4. **Nella terza parte del documento** troverai una possibile soluzione alla domanda tipo.
5. **Allegato a questo file** trovi lo stesso codice Python dell'esercizio sulla programmazione multi-threaded presentato nella parte 1, e un esempio di testo che verrà usato come formato di input per l'esercizio da scrivere in Perl **.**

PRIMA PARTE - MATERIALE DIDATTICO

Il codice fornito implementa una `BlockingQueue` con ritardo, che chiameremo `DelayedBlockingQueue`. Una `DelayedBlockingQueue` accumula gli elementi al suo interno nel momento in cui si esegue l'operazione di inserimento, ma questi elementi non sono resi disponibili prima di un tempo fissato `d`. Ad esempio, supponiamo di avere una `DelayedBlockingQueue` che chiameremo `DQ`. `DQ` può contenere al massimo 10 elementi, è inizialmente vuota, ed è preimpostata con un ritardo d di 5000 millisecondi (5 secondi). Se si inserisce un elemento `E` con l'operazione `put(E)`, l'operazione termina immediatamente, ma una successiva `take()` si bloccherà in attesa dell'elemento fino allo scadere dei cinque secondi, poichè `E` non viene realmente inserito in DQ prima di 5 secondi.
I 5 secondi sono conteggiati a partire dal momento dell'operazione `put(E)`. I metodi di cui è dotata una `DelayedBlockingQueue` sono i seguenti:

`put(self,e)` : inserisce l'elemento `e` nella coda, con il ritardo di default. Il metodo si blocca se la coda dovesse essere piena.
`put(self,e,r : int)` : inserisce l'elemento `e` nella coda, con il ritardo `r`, espresso in millisecondi. Il metodo si blocca in attesa di uno slot libero se la coda dovesse essere piena.
`take(self)` : preleva e restituisce dalla coda un elemento. Il metodo si blocca in attesa se non ci sono disponibili elementi nella coda il cui ritardo sia già scaduto.
`poll(self) -> int`: restituisce 0 se ci sono elementi già prelevabili, -1 se la coda è vuota, o altrimenti il tempo che resta (in millisecondi) prima che almeno un elemento sia disponibile al prelievo.
`setDelay(self,d : int)`: imposta il ritardo di default a `d` millisecondi.
`getDelay(self) -> int`: restituisce il ritardo di default.

```python
#!/usr/bin/python3

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

'''
    Classe Thread che aiuta a inserire elementi in ritardo.
'''

class Putter(Thread):

    def __init__(self,dcoda,e,d):
        super().__init__()
        self.dcoda = dcoda
        self.e = e
        self.d = d

    def run(self):
        self.dcoda.putRitardato(self.e,self.d)


class DelayedBlockingQueue:

    #
    #    * Costruisce una DelayedBlockingQueue
    #    *
    #    * @param d
    #    *            Ritardo iniziale di default
    #    * @param size
    #    *            Dimensione massima della blocking queue
    #
    def __init__(self, d, size : int):
        #
        #   Lock per la gestione dei campi interni alla coda
        #
        self.wlock = RLock()
```

```
        #
        #   Condizioni di attesa
        #
        self.full = Condition(self.wlock)
        self.empty = Condition(self.wlock)
        self.sleepCondition = Condition(self.wlock)

        #
        #   Coda degli elementi con ritardo scaduto e pronti al prelievo
        #
        self.coda = []

        #
        #   Taglia massima di 'coda'
        #
        self.maxSize = size
        #
        #   Collezione degli elementi non ancora scaduti e abbinati insieme al
        #   tempo in cui scadranno
        #
        self.scadenze = {}
        #
        #   Ritardo di default
        #
        self.delay = d


    #
    #    * Restituisce il ritardo di default
    #    *
    #    * @return il ritardo di default in millisecondi
    #
    def getDelay(self):
        with self.wlock:
            return self.delay

    #
```

```
#    * Imposta il ritardo di default
#    *
#    * @param d
#    *            il nuovo valore del ritardo di default, in millisecondi
#
def setDelay(self, d):
    with self.wlock:
        self.delay = d

#
#    * Inserisce un elemento nella DelayedBlockingQueue, con ritardo di default.
#    *
#    * @param e
#    *            elemento di tipo T da inserire
#
def put(self, e, d = None):
    with self.wlock:
        if d == None:
            d = self.getDelay()
        Putter(self,e,d).start()

#
#    * Inserimento di un elemento nella DelayedBlockingQueue, con ritardo a
#    * piacere. L'elemento viene inserito in differita sfruttando un thread che
#    * dorme per d millisecondi e inserisce in coda solo allo scadere del tempo. Nel
#    * frattempo si tiene traccia del momento della scadenza per poter
#    * implementare il metodo poll()
#    *
#    * @param e
#    *            elemento di tipo T da inserire
#    * @param d
#    *            ritardo passato il quale l'elemento sara' disponibile, in
#    *            millisecondi
#
def putRitardato(self,e,d):

    with self.wlock:
```

```
        while len(self.coda)+len(self.scadenze) == self.maxSize:
            self.full.wait()
        quandoScade = time.time() + d
        self.scadenze[e] = quandoScade

        #
        #  Per evitare eventuali risvegli spuri, usiamo un ciclo di controllo
        #
        while time.time() < quandoScade:
            self.sleepCondition.wait(d)
            d = time.time() - quandoScade


        del self.scadenze[e]
        self.veraPut(e)



#
#    * inserisce effettivamente un elemento in coda. Usato quando scade il tempo
#    * di attesa
#    *
#    * @param e elemento da inserire
#
def veraPut(self, e):
    self.coda.insert(0,e)
    self.empty.notify_all()

#
#    * Prelievo da coda. Coincide sostanzialmente con il normalissimo codice di
#    * prelievo da una blockingqueue standard
#    *
 #
def take(self):
    with self.wlock:
        while len(self.coda) == 0:
            self.empty.wait()
```

```
        self.full.notify_all()
        return self.coda.pop()

#
#    * Calcola il tempo mancante come la piu' imminente delle scadenze meno il
#    * tempo corrente
#    *
#
def poll(self):
    with self.wlock:
        if len(self.coda) > 0:
            return 0
        elif len(self.scadenze) > 0:
            tempoMancante = self.scadenze[min(self.scadenze, key=lambda x: self.scadenze[x] )] - time.time()

            #
            #  Per eliminare le situazioni in cui c'e' un elemento in
            #  scadenza imminente (o gia' avvenuta) tale per cui
            #  tempoMancante <= 0, ma ancora il thread che fa l'inserimento
            #  non ha potuto inserire.
            #
            return tempoMancante if tempoMancante > 0 else 0
        else:
            return -1


def minScadenza(self):
    #
    # Metodo che illustra a cosa equivale:
    # self.scadenze[min(self.scadenze, key=lambda x: self.scadenze[x] )]
    # non realmente usato.
    #
    min = None
    for key in self.scadenze:
        if min == None or self.scadenze[key] < min:
            min = self.scadenze[key]
    return min
```

```
    def show(self):

        with self.wlock:

            print ("MIN:{0:.2f} ".format(self.poll()),end='')

            for e in self.coda:
                print ("{}:0 ".format(e), end='')

            for e in self.scadenze:
                print("{0:}:{1:.3f} ".format(e,self.scadenze[e]-time.time()),end='')

            print()
'''
    Classi Consumer e Producer di test
'''
class Consumer(Thread):


    def __init__(self,buffer):
        self.queue = buffer
        Thread.__init__(self)

    def run(self):
        while True:
            time.sleep(random.random()*2)
            self.queue.take()
            self.queue.show()


class Producer(Thread):

    counter = 0

    def __init__(self,buffer):
        self.queue = buffer
```

```
        Thread.__init__(self)

    def run(self):
        while True:
            time.sleep(random.random() * 2)
            Producer.counter +=1
            self.queue.put("E-{}.{}".format(self.name,Producer.counter),random.random()*5)
            self.queue.show()
#
#  Main
#
buffer = DelayedBlockingQueue(1,10)

producers = [Producer(buffer) for x in range(3)]
consumers = [Consumer(buffer) for x in range(10)]

for p in producers:
    p.start()

for c in consumers:
    c.start()

print ("Started")
```

SECONDA PARTE - ESEMPIO DI DOMANDA

Aggiungi alla classe `DelayedBlockingQueue` un metodo `rallenta(d : int)`. Tale metodo prende in considerazione tutti gli elementi attualmente presenti nella coda (in attesa di scadenza o già scaduti) e vi aggiunge un ulteriore ritardo `d`.

PER CHI SVOLGE LA PROVA TELEMATICA: Si scriva su carta il codice del metodo, e si indichi, aiutandosi con i numeri di riga, quali modifiche andrebbero apportate al codice pre-esistente.

PER CHI SVOLGE LA PROVA IN LABORATORIO: Si modifichi il codice fornito e lo si salvi sul desktop all'interno di una cartella denominata *COGNOME-NOME-MATRICOLA* (sostituire il proprio Cognome, Nome e Matricola nel nome della cartella)

TERZA PARTE - ESEMPIO DI SOLUZIONE

Alla classe `DelayedBlockingQueue` aggiungo il metodo:

```
def rallenta(self,d : int):
        with self.wlock:
            while len(self.coda) > 0:
                self.putRitardato(self.take(),d)
            for e in self.scadenze:
                self.scadenze[e] += d
```

E' inoltre necessario agire all'interno del metodo `putRitardato`, e modificare il blocco:

```
            while time.time() < quandoScade:
                  self.sleepCondition.wait(d)
                  d = time.time() - quandoScade
```

In:

```
             while time.time() < self.scadenze[e]:
                self.sleepCondition.wait(d)
                d = time.time() - self.scadenze[e]
```

(Il motivo della modifica al metodo putRitardato non viene volutamente fornito)

PERL - MATERIALE PRELIMINARE

OUTPUT COMPLETO DEL COMANDO SHELL DA ESEGUIRE

```
Tasks: 146 total,   1 running, 105 sleeping,   0 stopped,   0 zombie
%Cpu(s):  3,7 us,  1,7 sy,  0,0 ni, 93,0 id,  1,3 wa,  0,0 hi,  0,2 si,  0,0 st
KiB Mem :  5830408 total,   137388 free,  1162844 used,  4530176 buff/cache
KiB Swap:  2097148 total,  2081520 free,    15628 used.  4389976 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
13292 frances+  20   0   41780   3528   3020 R 12,5  0,1   0:00.02 top
    1 root      20   0  225672   6628   4336 S  0,0  0,1   2:00.93 systemd
    2 root      20   0       0      0      0 S  0,0  0,0   0:00.22 kthreadd
    3 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 rcu_gp
    4 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 rcu_par_gp
    6 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 kworker/0:0H-kb
    8 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 mm_percpu_wq
    9 root      20   0       0      0      0 S  0,0  0,0   1:02.55 ksoftirqd/0
   10 root      20   0       0      0      0 I  0,0  0,0   2:45.43 rcu_sched
   11 root      rt   0       0      0      0 S  0,0  0,0   0:02.44 migration/0
   12 root     -51   0       0      0      0 S  0,0  0,0   0:00.00 idle_inject/0
   14 root      20   0       0      0      0 S  0,0  0,0   0:00.00 cpuhp/0
   15 root      20   0       0      0      0 S  0,0  0,0   0:00.00 kdevtmpfs
   16 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 netns
   17 root      20   0       0      0      0 S  0,0  0,0   0:00.00 rcu_tasks_kthre
   18 root      20   0       0      0      0 S  0,0  0,0   0:00.00 kauditd
   19 root      20   0       0      0      0 S  0,0  0,0   0:00.49 khungtaskd
   20 root      20   0       0      0      0 S  0,0  0,0   0:00.00 oom_reaper
   21 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 writeback
   22 root      20   0       0      0      0 S  0,0  0,0   0:01.46 kcompactd0
   23 root      25   5       0      0      0 S  0,0  0,0   0:00.00 ksmd
```

```
 24 root      39  19       0      0      0 S  0,0  0,0   0:00.00 khugepaged
116 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 kintegrityd
117 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 kblockd
118 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 blkcg_punt_bio
119 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 tpm_dev_wq
120 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 ata_sff
121 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 md
122 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 edac-poller
123 root       0 -20       0      0      0 I  0,0  0,0   0:00.00 devfreq_wq
124 root      rt   0       0      0      0 S  0,0  0,0   0:00.00 watchdogd
125 root      20   0       0      0      0 S  0,0  0,0   1:01.35 kswapd0
```