Corso di Sistemi Operativi e Reti

Prova scritta dell'1 SETTEMBRE 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. **Alla scadenza** termina *immediatamente* di scrivere, e attendi di essere chiamato, pena l'esclusione dalla prova;
6. **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

Aggiungi alla classe `DelayedBlockingQueue` un metodo `takeLast(self)`. Tale metodo estrae l'elemento che in questo momento risulta essere l'ultimo in coda, considerando sia gli elementi già inseriti che gli elementi in scadenza. Se il buffer FIFO è completamente privo di elementi (sia già inseriti che in scadenza), il metodo si pone in attesa dell'arrivo di un elemento. Se l'elemento che viene scelto per l'estrazione è ancora in scadenza, viene estratto l'elemento destinato a scadere per ultimo. La scadenza stessa non viene rispettata e l'elemento viene estratto immediatamente.

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.

```
001: #!/usr/bin/python3
002:
003: from threading import RLock,Thread,Condition
004: import time
005: import random
006:
007: '''
008:     Classe Thread che aiuta a inserire elementi in ritardo.
009: '''
010:
011: class Putter(Thread):
012:
013:     def __init__(self,dcoda,e,d):
014:         super().__init__()
015:         self.dcoda = dcoda
016:         self.e = e
017:         self.d = d
018:
019:     def run(self):
020:         self.dcoda.putRitardato(self.e,self.d)
021:
022:
023: class DelayedBlockingQueue:
024:
025:     #
026:     #       * Costruisce una DelayedBlockingQueue
027:     #       *
028:     #       * @param d
029:     #       *            Ritardo iniziale di default
030:     #       * @param size
031:     #       *            Dimensione massima della blocking queue
032:     #
033:     def __init__(self, d, size : int):
034:         #
035:         #   Lock per la gestione dei campi interni alla coda
```

```
036:         #
037:         self.wlock = RLock()
038:         #
039:         #   Condizioni di attesa
040:         #
041:         self.full = Condition(self.wlock)
042:         self.empty = Condition(self.wlock)
043:         self.sleepCondition = Condition(self.wlock)
044:
045:         #
046:         #   Coda degli elementi con ritardo scaduto e pronti al prelievo
047:         #
048:         self.coda = []
049:
050:         #
051:         #   Taglia massima di 'coda'
052:         #
053:         self.maxSize = size
054:         #
055:         #   Collezione degli elementi non ancora scaduti e abbinati insieme al
056:         #   tempo in cui scadranno
057:         #
058:         self.scadenze = {}
059:         #
060:         #   Ritardo di default
061:         #
062:         self.delay = d
063:
064:
065:     #
066:     #       * Restituisce il ritardo di default
067:     #       *
068:     #       * @return il ritardo di default in millisecondi
069:     #
070:     def getDelay(self):
```

```
071:         with self.wlock:
072:             return self.delay
073:
074:     #
075:     #       * Imposta il ritardo di default
076:     #       *
077:     #       * @param d
078:     #       *            il nuovo valore del ritardo di default, in millisecondi
079:     #
080:     def setDelay(self, d):
081:         with self.wlock:
082:             self.delay = d
083:
084:     #
085:     #       * Inserisce un elemento nella DelayedBlockingQueue, con ritardo di default.
086:     #       *
087:     #       * @param e
088:     #       *            elemento di tipo T da inserire
089:     #
090:     def put(self, e, d = None):
091:         with self.wlock:
092:             if d == None:
093:                 d = self.getDelay()
094:             Putter(self,e,d).start()
095:
096:     #
097:     #       * Inserimento di un elemento nella DelayedBlockingQueue, con ritardo a
098:     #       * piacere. L'elemento viene inserito in differita sfruttando un thread che
099:     #       * dorme per d millisecondi e inserisce in coda allo scadere del tempo. Nel
100:     #       * frattempo si tiene traccia del momento della scadenza per poter
101:     #       * implementare il metodo poll()
102:     #       *
103:     #       * @param e
104:     #       *            elemento di tipo T da inserire
105:     #       * @param d
```

```
106:     #       *            ritardo passato il quale l'elemento sara' disponibile, in
107:     #       *            millisecondi
108:     #
109:     def putRitardato(self,e,d):
110:
111:         with self.wlock:
112:             while len(self.coda)+len(self.scadenze) == self.maxSize:
113:                 self.full.wait()
114:             quandoScade = time.time() + d
115:             self.scadenze[e] = quandoScade
116:
117:             #
118:             #  Per evitare eventuali risvegli spuri, usiamo un ciclo di controllo
119:             #
120:             while time.time() < quandoScade:
121:                 self.sleepCondition.wait(d)
122:                 d = time.time() - quandoScade
123:
124:
125:             del self.scadenze[e]
126:             self.veraPut(e)
127:
128:
129:
130:     #
131:     #       * inserisce effettivamente un elemento in coda. Usato quando scade il tempo
132:     #       * di attesa
133:     #       *
134:     #       * @param e elemento da inserire
135:     #
136:     def veraPut(self, e):
137:         self.coda.insert(0,e)
138:         self.empty.notify_all()
139:
140:     #
```

```
141:     #       * Prelievo da coda. Coincide sostanzialmente con il normalissimo codice di
142:     #       * prelievo da una blockingqueue standard
143:     #       *
144:      #
145:     def take(self):
146:         with self.wlock:
147:             while len(self.coda) == 0:
148:                 self.empty.wait()
149:             self.full.notify_all()
150:             return self.coda.pop()
151:
152:     #
153:     #       * Calcola il tempo mancante come la piu' imminente delle scadenze meno il
154:     #       * tempo corrente
155:     #       *
156:     #
157:     def poll(self):
158:         with self.wlock:
159:             if len(self.coda) > 0:
160:                 return 0
161:             elif len(self.scadenze) > 0:
162:                 tempoMancante = self.scadenze[min(self.scadenze, key=lambda x: self.scadenze[x] )]
- time.time()
163:
164:                 #
165:                 #  Per eliminare le situazioni in cui c'e' un elemento in
166:                 #  scadenza imminente (o gia' avvenuta) tale per cui
167:                 #  tempoMancante <= 0, ma ancora il thread che fa l'inserimento
168:                 #  non ha potuto inserire.
169:                 #
170:                 return tempoMancante if tempoMancante > 0 else 0
171:             else:
172:                 return -1
173:
174:
```

```
175:     def minScadenza(self):
176:         #
177:         # Metodo che illustra a cosa equivale:
178:         # self.scadenze[min(self.scadenze, key=lambda x: self.scadenze[x] )]
179:         # non realmente usato.
180:         #
181:         min = None
182:         for key in self.scadenze:
183:             if min == None or self.scadenze[key] < min:
184:                 min = self.scadenze[key]
185:         return min
186: 
187:     def show(self):
188: 
189:         with self.wlock:
190: 
191:             print ("MIN:{0:.2f} ".format(self.poll()),end='')
192: 
193:             for e in self.coda:
194:                 print ("{}:0 ".format(e), end='')
195: 
196:             for e in self.scadenze:
197:                 print("{0:}:{1:.3f} ".format(e,self.scadenze[e]-time.time()),end='')
198: 
199:             print()
200: '''
201:     Classi Consumer e Producer di test
202: '''
203: class Consumer(Thread):
204: 
205: 
206:     def __init__(self,buffer):
207:         self.queue = buffer
208:         Thread.__init__(self)
209: 
```

```
210:     def run(self):
211:         while True:
212:             time.sleep(random.random()*2)
213:             self.queue.take()
214:             self.queue.show()
215:
216:
217: class Producer(Thread):
218:
219:     counter = 0
220:
221:     def __init__(self,buffer):
222:         self.queue = buffer
223:         Thread.__init__(self)
224:
225:     def run(self):
226:         while True:
227:             time.sleep(random.random() * 2)
228:             Producer.counter +=1
229:             self.queue.put("E-{}.{}".format(self.name,Producer.counter),random.random()*5)
230:             self.queue.show()
231: #
232: #  Main
233: #
234: buffer = DelayedBlockingQueue(1,10)
235:
236: producers = [Producer(buffer) for x in range(3)]
237: consumers = [Consumer(buffer) for x in range(10)]
238:
239: for p in producers:
240:     p.start()
241:
242: for c in consumers:
243:     c.start()
244:
```

```
245: print ("Started")
```