Corso di Sistemi Operativi e Reti

Prova scritta telematica 3 LUGLIO 2020

ESERCIZIO 1 - PROGRAMMAZIONE MULTITHREAD

ISTRUZIONI

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.

Corso di Sistemi Operativi e Reti

Prova scritta telematica 3 LUGLIO 2020

ESERCIZIO 1, TURNO 1 - QUESITO DA RISOLVERE

Aggiungi alla classe `StampaPrioritaria` il metodo `stop()`. Se invocato, il metodo `stop()` mette la classe in stato di arresto: si inibisce la richiesta di ulteriori stampe attraverso il metodo `stampa`, mentre il thread stampatore deve terminare di effettuare le stampe già prenotate, e infine terminare la sua attività. Le stampe prenotate mentre si è in stato di arresto devono essere ignorate.

Ad esempio, il seguente frammento di codice:

```
sp = StampaPrioritaria(10)
sp.stampa("Come va",0)
sp.stampa("Ciao",2)
sp.stop()
sp.stampa("Salve",0)
```

Provocherebbe la stampa a video di “Come va” e “Ciao”, mentre la terza stampa non verrà effettuata.

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.

MATERIALE DIDATTICO

Il codice fornito implementa una classe per la stampa a schermo multithreaded e con priorità tra le stampe. La libreria/classe fornisce un unico metodo `stampa(s :str, prio :int)` dove `s` è una stringa da stampare a video e `prio` un valore di priorità che può valere 0 (altà priorità), 1 (media priorità), 2 (bassa priorità).

L'effetto dell'invocazione di `stampa` per il thread chiamante è quello di "prenotare" la visualizzazione a video della stringa `s`., e di uscire immediatamente, senza attendere la stampa su video.

L'esecuzione della reale stampa a video è infatti asincrona: il codice di `stampa()` non è bloccante per il thread chiamante, ma serve a bufferizzare le richieste di stampa e uscire immediatamente, salvo il caso in cui le strutture dati che memorizzano le richieste di stampa siano piene e non possano accogliere la richiesta corrente.

Un thread `Stampatore` si occupa di eseguire materialmente le stampe a video, prelevando continuamente le richieste di stampa dalle strutture dati che conservano le richieste di stampa prenotate. Tale thread stampa secondo le seguenti regole:

1. Una richiesta di stampa a bassa priorità può essere eseguita solo se non ci sono richieste ad alta e media priorità pendenti;
2. Una richiesta di stampa a media priorità può essere eseguita solo se non ci sono richieste ad alta priorità pendenti;
3. Come eccezione della regola 1, le stampe a bassa priorità vengono comunque eseguite al più presto se sono state eseguite più di 10 stampe consecutive a priorità 0 e 1 senza che avvenga alcuna stampa a priorità 2.
4. Come eccezione della regola 2, le stampe a media priorità vengono comunque eseguite al più presto se sono state eseguite più di 5 stampe consecutive a priorità 0 senza che avvenga alcuna stampa a priorità 1.
5. In assenza di richieste di stampa pendenti, il thread stampatore si pone in stato di attesa e si risveglia quando necessario.

Indicazioni sul codice fornito

La classe è stata implementata prevedendo il metodo pubblico ‘stampa’ (con prototipo definito in accordo alle specifiche) e il metodo ‘prelevaStampa’, il quale sceglie e ritorna una stringa da stampare. La classe è dotata di tre array gestiti con politica FIFO (non sincronizzati) C[0], C[1] e C[2], e un lock L. La funzione stampa(s,p) è implementata:

1. Acquisendo il lock L;

2. Inserendo s sul rispettivo buffer C[p]. Se questa operazione non si può concludere poiché il corrispondente buffer è pieno, ci si pone in wait() su una Condition opportuna;

3. “notify”-cando il thread stampatore, per avvisarlo della presenza di una nuova richiesta di stampa;

4. Rilasciando il lock/monitor L.

Il thread Stampatore si comporta nel seguente modo:

1. S = prelevaStampa()
2. Stampa S.
3. Ritorna al punto 1.

Il corpo della funzione prelevaStampa() contiene le operazioni:

1. Acquisizione del lock L;
2. Estrazione di una stringa S da uno dei tre buffer. La scelta della stringa dipende dalle regole specificate; in assenza di valori da stampare, lo Stampatore si pone in wait();
3. Rilascio del lock L;

Soluzione tipo all’esercizio StampaPrioritaria. Ricorda che **questo stesso codice è disponibile sotto forma di file *.py** utilizzabile per fare qualche prova di esecuzione.

```
1   from threading import Thread,RLock, Condition, get_ident
2   from random import randint
3
4   """
5   #
6   # Questo è il codice del Thread Stampatore. E' prevista una unica istanza che effettua le stampe
7   #
8   """
9   class Stampatore(Thread):
10
11      def __init__(self,SP):
12          super().__init__()
13          self.SP = SP
14
15      def run(self):
16          while(True):
17              s = self.SP.prelevaStampa()
18              print(s)
19
20
21  """
22  #   Ci sono in tutto tre code di attesa per le stampe: 0 = alta pr., 1 = media pr., 2 = bassa pr.
23  #    NUMERO BASSO = MAGGIORE PRIORITA
24  #
25  """
26  class StampaPrioritaria:
27
28
29      NCODE = 3
30      """
31         La traccia prescrive che per le stampe a media priorità, sia possibile effettuare comunque una stampa ogni 5 a priorità più alta,
32         Mentre per le stampe a bassa priorità, questa analoga soglia è 10.
33         Queste soglie sono codificate nell'array SOGLIE
34      """
35      SOGLIE = [0,5,10]
36
37      def __init__(self,n):
38          """
39              Dimensione massima di ogni coda
40          """
41          self.size = n
42          """
43              Le tre code saranno rispettivamente C[0], C[1] e C[2].
44          """
45          self.C = []
46          """
47              Le attese ci dicono quante stampe sono state effettuate consecutivamente
48              a priorita minore di i senza che una stampa a priorità i sia stata fatta.
49              Esempio: attese[2] = 12 indica che sono state fatte 12 stampe consecutive
50              a priorità 0 e 1 senza che sia stata mai fatta una stampa a priorità 2
51          """
52          self.attese = []
```

```
53              """
54                  Useremo L come unico lock per garantire la thread safety
55              """
56              self.L = RLock()
57              """
58                  Se non ci sono stampe da fare, il thread Stampatore aspetterà su questa condition
59              """
60              self.condEmpty = Condition(self.L)
61              """
62                  Ci sarà invece una condizione di attesa per ciascuna coda nel caso in cui questa sia piena.
63                  Esempio, se C[1] è piena, aspetto su condFull[1].
64              """
65              self.condFull = []
66              """
67                  Qui riempio opportunamente C, condFull e attese
68              """
69              for i in range(0,self.NCODE):
70                  self.C.append([])
71                  self.condFull.append(Condition(self.L))
72                  self.attese.append(0)
73              """
74                  Creo e avvio l'unico thread stampatore
75              """
76              self.printer = Stampatore(self)
77              self.printer.start()
78
79          """
80              Metodo privato che mi restituisce len(C[0]) + len(C[1]) + len(C[2])
81          """
82          def __totLen(self) -> int:
83              retVal = 0
84              for i in range(0,self.NCODE):
85                  retVal += len(self.C[i])
86              return retVal
87
88          """
89              Metodo privato che mi dice se NON ci sono stampe a priorità più bassa di una certa priorità p che sono in attesa da troppo tempo
90              Esempio, supponiamo che p = 0, len(C[1]) = 1, len(C[2]) = 0, attese [0,6,0]
91              siccome c'è una stampa a priorità 1 che aspetta da 6 "giri", allora restituisco False, e cioè non posso eseguire una stampa a livello p=0
92              poichè ci sono stampe a priorità più bassa, ma che aspettano da troppo tempo.
93          """
94          def __noAltreSoglieSuperate(self,p : int) -> bool:
95              for q in range(p+1,self.NCODE):
96                  if len(self.C[q]) > 0 and self.attese[q] >= self.SOGLIE[q]:
97                      return False
98              return True
106
107         """
108             Questo metodo pone la stringa s a priorità prio nel buffer C[prio].
109             Si mette in attesa bloccante se in C[prio] non c'è posto.
110         """
111         def stampa(self, s: str, prio: int):
```

```
112             with self.L:
113                 while len(self.C[prio]) == self.size:
114                     self.condFull[prio].wait()
115                 self.C[prio].append(s)
116                 #print(f"{self.C[prio]}")
117                 self.condEmpty.notify()
118 
119         """
120             Il thread stampatore sceglie la prossima stampa da effettuare grazie a questo metodo
121         """
122         def prelevaStampa(self) -> str:
123             with self.L:
124                 """
125                     Attendo se non ci sono stampe in nessuna coda
126                 """
127                 while self.__totLen() == 0:
128                     self.condEmpty.wait()
129 
130                 """
131                     Ciclo sulle tre code partendo da quella a priorità più alta.
132                 """
133                 for p in range(0,self.NCODE):
134                     """
135                         Posso stampare a priorità p ?
136                         Per poter stampare a priorità p:
137                           -devo avere qualche stampa in attesa su questa priorità (len(C[p]) > 0) e inoltre:
138                           -NON ci devono essere in attesa da troppo tempo delle stampe a livello p+1 a salire
139 
140                     """
141                     if len(self.C[p]) > 0 and self.__noAltreSoglieSuperate(p):
142                         """
143                         OK, se sono qui, posso stampare a priorità p. Procedo ad aggiornare le attese sulle priorità da p+1 a salire
144                         """
145                         for q in range(p+1,self.NCODE):
146                             if len(self.C[q]) > 0:
147                                 self.attese[q] += 1
148                         """
149                             Finalmente ho stampato a livello p, quindi azzero attese[p]
150                         """
151                         self.attese[p] = 0
152                         """
153                             Se in questo momento C[p] è piena, vuol dire che sto per fare un pop() che potrebbe
154                             sbloccare un thread in attesa di trovare posto su C[p]. Quindi faccio notify()
155                         """
156                         if len(self.C[p]) == self.size:
157                             self.condFull[p].notify()
158                         """
159                             Infine, estraggo un elemento da C[p] e lo restituisco
160                         """
161                         return self.C[p].pop(0)
162 
163 
```

```
    """
        ClientThread è giusto una tipologia di thread di esempio che sorteggia una priorità casuale e produce stampe a quella priorità
    """
    class clientThread(Thread):

        def __init__(self,SP):
            super().__init__()
            self.SP = SP
            self.p = randint(0,StampaPrioritaria.NCODE-1)

        def run(self):
            print(f"Stampa n.1 del Thread {get_ident()} con priorità {self.p}")
            self.SP.stampa(f"Stampa n.1 del Thread {get_ident()} con priorità {self.p}",self.p)
            count = 1
            while(True):
                count += 1
                self.SP.stampa(f"Stampa n.{count} del Thread {get_ident()} con priorità {self.p}", self.p)

    """
        Questo è un main di esempio che crea una istanza di StampaPrioritaria e dei ClientThread che ne fanno uso
    """

    if __name__ == '__main__':
        stampa = StampaPrioritaria(10)
        for t in range(0,5):
            clientThread(stampa).start()

```