Corso di Sistemi Operativi e Reti

Prova scritta del 9 FEBBRAIO 2021

ESERCIZI 1 e 2 - MATERIALE PRELIMINARE E ISTRUZIONI

ISTRUZIONI

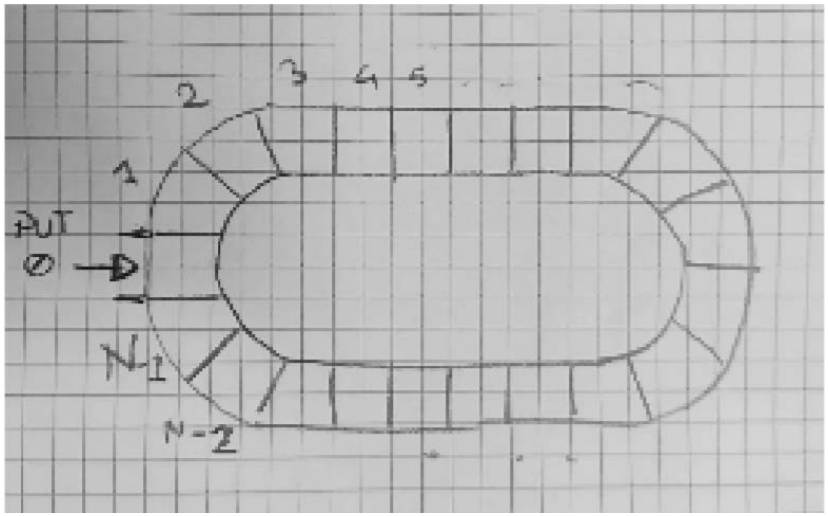
In questo documento trovi:

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 eseguire 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 un `RunningSushiBuffer`. Tale struttura dati è costituita da un particolare tipo di buffer circolare thread-safe con `N` slot in cui è possibile inserire degli elementi solo in posizione `0`. L'operazione di estrazione può essere invece eseguita da tutte le altre posizioni (da 1 a N-1). Inoltre, gli elementi del `RunningSushiBuffer` possono essere arbitrariamente ruotati di `X` posizioni in senso orario (Si veda la figura 1).
I metodi che sono stati implementati sono i seguenti:

`put(self,t)`: Si pone in attesa bloccante fintantoché la posizione `0` è occupata. Una volta usciti dall'attesa bloccante, inserisce l'elemento `t` in posizione 0.
`get(self,i)`. Si pone in attesa bloccante se la posizione `i` è libera. Rimuove e restituisce l'elemento in posizione `i` quando questo si rende disponibile.
`shift(self,j)`. Ruota di `j` posizioni il buffer circolare in senso antiorario. La rotazione è da intendersi come una rinumerazione degli indici dove ogni elemento accessibile in generica posizione `X` viene passato in posizione `(X+j) mod N`, dove `N` è la dimensione del buffer. Ad esempio se `j=1`, l'elemento in posizione `0` deve diventare quello che prima era in posizione `1`, l'elemento `1` diventa quello che prima era in posizione 2, e così via. Essendo il buffer circolare, l'elemento che prima era in posizione `0` deve trovarsi in posizione `N-1`.
Si noti che, per evitare inutili operazioni di copia, lo spostamento è stato ottenuto con un meccanismo di rimappatura dinamica degli indici.



Nel codice vengono forniti a corredo anche dei Thread di esempio delle seguenti tipologie:

1. `Cuoco.` Un cuoco produce periodicamente degli elementi che vengono depositati in posizione 0;
2. `Cliente.` Un cliente viene posizionato su una specifica posizione p che non cambia mai, ed estrae periodicamente degli elementi da questa posizione
3. `NastroRotante.` Il nastro rotante muove periodicamente le posizioni del buffer stesso.

```python
#!/usr/bin/env python

from threading import Lock,RLock, Condition, Thread
from time import sleep
from random import random, randint

debug = True

#
# Stampa sincronizzata
#
plock = Lock()
def sprint(s):
    with plock:
        print(s)
#
# Stampa solo in debug mode
#
def dprint(s):
    with plock:
        if debug:
            print(s)

class RunningSushiBuffer:

    theBuffer : list
    dim : int
    lock : RLock
    condition : Condition

    def __init__(self, dim):
        self.theBuffer = [None] * dim
        self.zeroPosition = 0
        self.dim = dim
        self.lock = RLock()
        self.condition = Condition(self.lock)

    def _getRealPosition(self,i : int):
```

```
        return (i + self.zeroPosition) % self.dim

    def get(self, pos : int):
        with self.lock:
            while self.theBuffer[self._getRealPosition(pos)] == None:
                self.condition.wait()
            palluzza = self.theBuffer[self._getRealPosition(pos)]
            self.theBuffer[self._getRealPosition(pos)] = None
            return palluzza

    def put(self, t):
        with self.lock:
            while self.theBuffer[self._getRealPosition(0)] != None:
                self.condition.wait()
            self.theBuffer[self._getRealPosition(0)] = t

    def shift(self, j = 1):
        with self.lock:
            #
            #  uso zeroPosition per spostare la posizione 0 solo virtualmente,
            #  anziche' dover ricopiare degli elementi
            #
            self.zeroPosition = (self.zeroPosition + j) % self.dim
            #
            #    E' solo grazie a uno shift che puo' crearsi la condizione per svegliare un thread
            #    in attesa, rispettivamente su put() o su get()
            #
            self.condition.notifyAll()

class NastroRotante(Thread):

    def __init__(self, d : RunningSushiBuffer):
        super().__init__()
        self.iterazioni = 10000
        self.d = d

    def run(self):
        while(self.iterazioni > 0):
```

```
            sleep(0.1)
            self.iterazioni -= 1
            self.d.shift()

class Cuoco(Thread):

    piatti = [ "*", ";", "^", "%"]

    def __init__(self, d : RunningSushiBuffer):
        super().__init__()
        self.iterazioni = 1000
        self.d = d

    def run(self):
        while(self.iterazioni > 0):
            sleep(0.5 * random())
            self.iterazioni -= 1
            randPiatto = randint(0,len(self.piatti)-1)
            self.d.put(self.piatti[randPiatto])
            print ( f"Il cuoco {self.ident} ha cucinato <{self.piatti[randPiatto]}>")
        print ( f"Il cuoco {self.ident} ha finito il suo turno e va via")

class Cliente(Thread):

    def __init__(self, d : RunningSushiBuffer, pos : int):
        super().__init__()
        self.coseCheVoglioMangiare = randint(1,20)
        self.d = d
        self.pos = pos

    def run(self):
        while(self.coseCheVoglioMangiare > 0):
            sleep(5 * random())
            self.coseCheVoglioMangiare -= 1
            print ( f"Il cliente {self.ident} aspetta cibo")
            print ( f"Il cliente {self.ident} mangia <{self.d.get(self.pos)}>")
        print ( f"Il cliente {self.ident} ha la pancia piena e va via")
```

```
size = 20
D = RunningSushiBuffer(size)
NastroRotante(D).start()
for i in range(0,2):
    Cuoco(D).start()
for i in range(1,size):
    Cliente(D,i).start()
```

PROGRAMMAZIONE IN PERL - MATERIALE PRELIMINARE

Il file `dump.log` contiene alcune informazioni riguardanti le connessioni di rete in entrata e uscita.

Una riga di esempio del file è così composta:

```
TIMESTAMP IP IP_SORGENTE.PORTA_SORGENTE > IP_DESTINAZIONE.PORTA_DESTINAZIONE: PROTOCOLLO, altri_parametri
```

############################## Esempio Reale ##############################

```
14:25:51.932550 IP 160.97.62.90.62536 > 224.0.0.252.5355: UDP, length 21
14:26:10.171155 IP 23.6.123.119.443 > 192.168.0.100.44664: Flags [.], ack 1, win 990, options [nop,nop,TS val 254276428
ecr 3274039351], length 0
14:25:53.114205 IP 192.168.0.1.50080 > 239.255.255.250.1900: UDP, length 273
15:25:53.218232 IP 192.168.0.1.50080 > 239.255.255.250.1900: UDP, length 336
14:22:53.322898 IP 192.168.0.1.62536 > 192.168.0.100.32865: UDP, length 21
14:25:53.322916 IP 192.168.0.100 > 192.168.0.1: ICMP 192.168.0.100 udp port 32865 unreachable, length 148
14:27:53.323570 IP 192.168.0.1.53 > 192.168.0.100.48932: ICMP 192.168.0.100 udp port 32865 unreachable, length 148
16:25:53.323589 IP 192.168.0.100 > 192.168.0.1: ICMP 192.168.0.100 udp port 48932 unreachable, length 130
```