Corso di Sistemi Operativi e Reti

Prova scritta 9 FEBBRAIO 2021

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. **Aiutati** con i numeri di linea per indicare le eventuali modifiche che vorresti fare al codice che ti è stato dato.
6. **Alla scadenza** termina *immediatamente* di scrivere, e attendi di essere chiamato, pena l'esclusione dalla prova;
7. **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, TURNO 1 - PROGRAMMAZIONE MULTITHREADED

Si introduca il metodo

```
richiediPistaGentilmente(self, id_squadra, numGiocatori)
```

Tale metodo si mette in attesa finchè non sono disponibili *numGiocatori* palle da bowling e una pista, proprio come il metodo `richiediPista`. Tuttavia, quando si invoca questo metodo, bisogna introdurre un meccanismo diverso di gestione della turnazione tra squadre.

Più in dettaglio, quando arriva il turno per una squadra S, e cioè si verifica che
`mioBollino == self.prossimo_da_servire` , ma contemporaneamente non è subito disponibile una pista oppure non c'è il numero di palle da bowling richiesto per la squadra S, allora ci si rimette subito in stato di wait, ma *retrocedendo di quattro posti nella lista di attesa*. Qualora i thread in attesa subito dopo S siano meno di 4, allora S si metterà in coda.

MATERIALE PER LA PROVA SULLA PROGRAMMAZIONE MULTI-THREADED

Il codice fornito implementa un piccolo sistema di gestione di una sala bowling. Una sala da bowling contiene un certo numero di piste P e un certo numero N di palle da bowling. Per poter giocare, una squadra composta da M giocatori necessità di avere a disposizione una pista e una palla da bowling per ciascun giocatore (dunque devono essere disponibili M palle da bowling).
Ogni pista può essere assegnata a una certa squadra, ammesso che sia disponibile un numero di palle da bowling pari al numero di giocatori per la data squadra. Ad esempio, se le palle a disposizione sono 10, e in un certo momento una squadra di 6 giocatori occupa la pista n.1, una eventuale squadra di 5 giocatori non potrà usufruire della pista n.2, pur essendo questa libera.
La classe Sala contiene principalmente due metodi:

```
richiediPista(self,numGiocatori :int) -> int
```

Il primo metodo consente di richiedere una pista e una dotazione di palle da bowling specificando un certo numero di giocatori. Se la quantità di palle richiesta non è disponibile, oppure non ci sono piste libere, la squadra viene messa in attesa. Viene restituito il codice della pista che è stata assegnata.

```
liberaPista(self,codicePista : int, numGiocatori : int)
```

Il secondo metodo prevede la liberazione di una pista precedentemente occupata e la restituzione delle palle da bowling non più utilizzate.

A corredo è fornito il codice di una classe Squadra che simula il comportamento casuale di una squadra attraverso un thread.
La soluzione fornita elimina i potenziali problemi di starvation accodando le richieste delle squadre secondo un certo numero di ordine di arrivo.

```
 1    #!/usr/bin/python3
 2
 3    from threading import RLock, Condition, Thread
 4    from time import sleep
 5    from random import random
 6
 7    class Sala:
 8
 9        def __init__(self, num_piste, num_palle):
10
11            self.lock = RLock()
12            self.condition = Condition(self.lock)
13            self.num_piste = num_piste
14            self.pista = [False] * num_piste
15            self.palleDisponibili = num_palle
16            self.prossimo_da_servire = 1
17            self.bollino = 1
18
19        #  senza starvation
20        def richiediPista(self, id_squadra, numGiocatori):
21
22            with self.lock:
23                mioBollino = self.bollino
24                self.bollino += 1
25                while (self.__cercaPista() == -1 or
26                       self.palleDisponibili < numGiocatori or
27                       mioBollino != self.prossimo_da_servire):
28
29                        print(f"La squadra {id_squadra} con {numGiocatori} giocatori deve
attendere il suo turno con bollino {mioBollino}")
30                        self.condition.wait()
31
32                #  fa scorrere il turno
33                self.prossimo_da_servire += 1
```

```
34                  #  si da la possibilita' immediata di provare a giocare al prossimo numero
da servire
35                  self.condition.notifyAll()
36                  self.palleDisponibili -= numGiocatori
37                  p = self.__cercaPista()
38                  self.pista[p] = True
39                  print(f"La squadra {id_squadra} ottiene la pista {p} con {numGiocatori}
giocatori e bollino {mioBollino}")
40                  return p
41
42          def liberaPista(self, numPista, numGiocatori):
43              with self.lock:
44                  self.palleDisponibili += numGiocatori
45                  self.pista[numPista] = False
46                  self.condition.notifyAll()
47
48          def __cercaPista(self):
49              for i in range(0,len(self.pista)):
50                  if not self.pista[i]:
51                      return i
52              return -1
53
54      class Squadra(Thread):
55
56          def __init__(self, id, sala):
57              super(Squadra, self).__init__()
58              self.sala = sala
59              self.id = id
60
61          def run(self):
62              while True:
63
64                  #  il giocatore fa altro prima di chiedere una pista
65                  sleep(int((random() * 6)))
66                  #  prova a chiedere una pista
```

```
67                  numGiocatori = int((random() * 20))+1
68                  print(f"La squadra {self.id} chiede una pista per {numGiocatori}
giocatori.")
69                  pista = self.sala.richiediPista(self.id, numGiocatori)
70                  print(f"La squadra {self.id} gioca sulla pista {pista} .")
71                  #  tempo di gioco
72                  sleep(int((random() * 4)))
73                  self.sala.liberaPista(pista, numGiocatori)
74                  print(f"La squadra {self.id} lascia la pista {pista}.")
75
76
77
78
79      if __name__ == '__main__':
80          s = Sala(3, 20)
81          for i in range(0,5):
82              Squadra(i, s).start()
```

ESERCIZIO 2, TURNO 1 - PERL

Il file `nmap.log` contiene una lista di indirizzi IP e delle relative porte (tcp o udp) aperte per ogni specifico host.
Un esempio di linea di output che indica una **porta** aperta per un determinato indirizzo **IP** è il seguente:

`Discovered open port 80/tcp on 192.168.1.107`

e significa che l'host con indirizzo IP **`192.168.1.107`** ha la porta **`80`** aperta.

Lo scopo dell'esercizio è quello di creare uno script Perl dal nome `scan.pl` che analizzerà il file `nmap.log` ed eseguirà determinate operazioni tenendo in considerazione solo ed esclusivamente gli host che presentano almeno 1 porta aperta.
Lo script leggerà come argomenti da linea di comando il nome del file di log (in questo caso `nmap.log`) ed una opzione tramite la quale si modificherà il comportamento dello script stesso.
Lo script dovrà essere quindi richiamato nel seguente modo

`./scan.pl nmap.log [--ip=IP_ADDRESS | --list]`

dove:

- **Nome File** = **`nmap.log`**
- **Option** = **`--ip=IP_ADDRESS`** **oppure** **`--list`**

N.B. La stringa IP_ADDRESS deve essere validata e corrispondere ad un indirizzo ip valido (ad esempio, 192.168.1.105 oppure 192.168.1.12 ...)

Se si inserisce l'opzione **`--ip=IP_ADDRESS`** lo script dovrà stampare in output la lista di tutte le porte aperte (se presenti) per l'indirizzo ip specificato e il numero totale di porte aperte trovate per quello stesso indirizzo IP.

Se l'utente specifica l'opzione **`--list`** lo script dovrà stampare su **un file dal nome** **`sorted_port.out`** la lista di tutti gli indirizzi IP menzionati nel file di log input, seguiti dal numero totale di porte aperte per ciascuno di essi. La stampa dovrà essere ordinata in ordine decrescente di numero di porte aperte.

ESEMPI DI ESECUZIONE A PAGINA SUCCESSIVA

Esempio 1:

Invocazione dello script: `./scan.pl nmap.log --ip=192.168.1.107`

Output su STDOUT:

Open ports on IP address 192.168.1.114
22
53
80

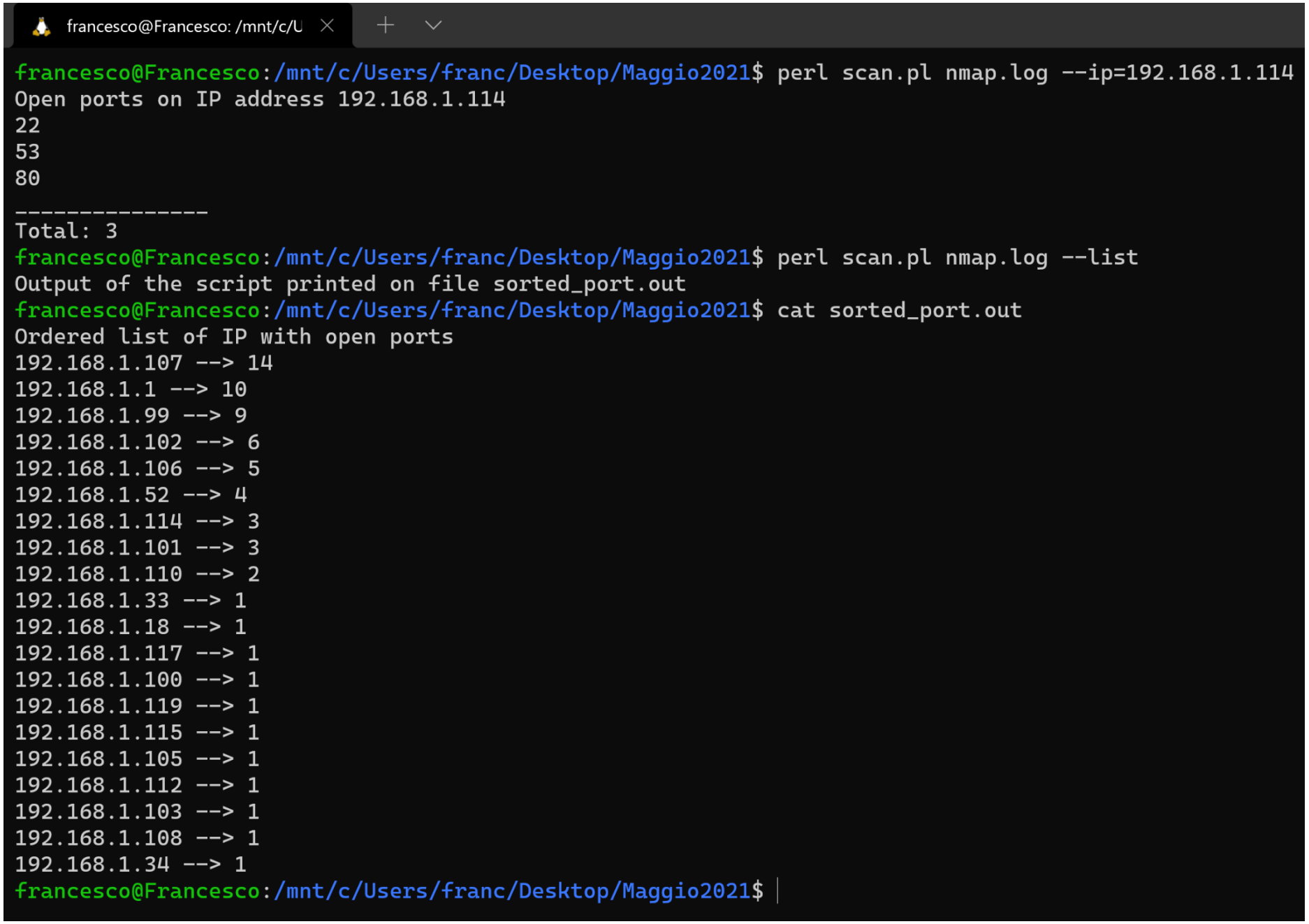
Total: 3

Esempio 2:

Invocazione dello script: `./scan.pl nmap.log --list`

Output su file "`sorted_port.out`":

Ordered list of IP with open ports
192.168.1.107 --> 14
192.168.1.1 --> 10
192.168.1.99 --> 9
192.168.1.102 --> 6
192.168.1.106 --> 5
192.168.1.52 --> 4
192.168.1.114 --> 3
192.168.1.101 --> 3
192.168.1.110 --> 2
192.168.1.108 --> 1
--- omissis ---


francesco@Francesco: /mnt/c/U
francesco@Francesco:/mnt/c/Users/franc/Desktop/Maggio2021$ perl scan.pl nmap.log --ip=192.168.1.114
Open ports on IP address 192.168.1.114
22
53
80

Total: 3
francesco@Francesco:/mnt/c/Users/franc/Desktop/Maggio2021$ perl scan.pl nmap.log --list
Output of the script printed on file sorted_port.out
francesco@Francesco:/mnt/c/Users/franc/Desktop/Maggio2021$ cat sorted_port.out
Ordered list of IP with open ports
192.168.1.107 --> 14
192.168.1.1 --> 10
192.168.1.99 --> 9
192.168.1.102 --> 6
192.168.1.106 --> 5
192.168.1.52 --> 4
192.168.1.114 --> 3
192.168.1.101 --> 3
192.168.1.110 --> 2
192.168.1.33 --> 1
192.168.1.18 --> 1
192.168.1.117 --> 1
192.168.1.100 --> 1
192.168.1.119 --> 1
192.168.1.115 --> 1
192.168.1.105 --> 1
192.168.1.112 --> 1
192.168.1.103 --> 1
192.168.1.108 --> 1
192.168.1.34 --> 1
francesco@Francesco:/mnt/c/Users/franc/Desktop/Maggio2021$

PROGRAMMAZIONE IN PERL - MATERIALE PRELIMINARE

Il file `nmap.log` contiene una lista di indirizzi IP e delle relative porte (tcp o udp) aperte per ogni specifico host.
Un esempio di **porta** aperta per un determinato indirizzo **IP** è il seguente:

Discovered open port 80/tcp on 192.168.1.107

e significa che l'host con indirizzo IP **192.168.1.107** ha la porta **80** aperta.

L'output del comando `nmap -v -p '*' 192.168.1.0/24` è il seguente:

```
Starting Nmap 7.80 ( https://nmap.org ) at 2021-05-18 10:59 CEST
Initiating Ping Scan at 10:59
Scanning 256 hosts [2 ports/host]
Completed Ping Scan at 10:59, 3.56s elapsed (256 total hosts)
Initiating Parallel DNS resolution of 256 hosts. at 10:59
Completed Parallel DNS resolution of 256 hosts. at 10:59, 0.01s elapsed
Nmap scan report for 192.168.1.0 [host down]
Nmap scan report for 192.168.1.2 [host down]
Nmap scan report for 192.168.1.3 [host down]
Nmap scan report for 192.168.1.4 [host down]
Nmap scan report for 192.168.1.5 [host down]
Nmap scan report for 192.168.1.6 [host down]
Nmap scan report for 192.168.1.7 [host down]
Nmap scan report for 192.168.1.8 [host down]
Nmap scan report for 192.168.1.9 [host down]
Nmap scan report for 192.168.1.10 [host down]
Nmap scan report for 192.168.1.11 [host down]
Nmap scan report for 192.168.1.12 [host down]
```

```
Nmap scan report for 192.168.1.13 [host down]
Nmap scan report for 192.168.1.14 [host down]
Nmap scan report for 192.168.1.15 [host down]
Nmap scan report for 192.168.1.16 [host down]
Nmap scan report for 192.168.1.17 [host down]
Nmap scan report for 192.168.1.19 [host down]
Nmap scan report for 192.168.1.20 [host down]
Nmap scan report for 192.168.1.21 [host down]
Nmap scan report for 192.168.1.22 [host down]
Nmap scan report for 192.168.1.23 [host down]
Nmap scan report for 192.168.1.24 [host down]
Nmap scan report for 192.168.1.25 [host down]
Nmap scan report for 192.168.1.26 [host down]
Nmap scan report for 192.168.1.27 [host down]
Nmap scan report for 192.168.1.28 [host down]
Nmap scan report for 192.168.1.29 [host down]
Nmap scan report for 192.168.1.30 [host down]
Nmap scan report for 192.168.1.31 [host down]
Nmap scan report for 192.168.1.32 [host down]
Initiating Connect Scan at 10:59
Scanning 4 hosts [8320 ports/host]
Discovered open port 53/tcp on 192.168.1.1
Discovered open port 443/tcp on 192.168.1.1
Discovered open port 80/tcp on 192.168.1.1
Discovered open port 21/tcp on 192.168.1.1
Discovered open port 139/tcp on 192.168.1.1
Discovered open port 445/tcp on 192.168.1.1
Discovered open port 80/tcp on 192.168.1.34
Discovered open port 80/tcp on 192.168.1.33
```

```
Increasing send delay for 192.168.1.33 from 0 to 5 due to 40 out of 131 dropped probes since last increase.
Discovered open port 263/tcp on 192.168.1.1
Discovered open port 1991/tcp on 192.168.1.1
Discovered open port 631/tcp on 192.168.1.1
Discovered open port 1990/tcp on 192.168.1.1
Completed Connect Scan against 192.168.1.1 in 2.15s (3 hosts left)
Discovered open port 2870/tcp on 192.168.1.18
Completed Connect Scan against 192.168.1.18 in 2.87s (2 hosts left)
Completed Connect Scan against 192.168.1.34 in 11.15s (1 host left)

--- omissis ---
```

OSSERVAZIONI SUGLI ERRORI PIU' COMUNI COMMESSI DURANTE LO SVOLGIMENTO DELLA TRACCIA

Errore 1. Implementare il meccanismo di cambio della posizione in coda semplicemente con

```
miobollino += min(4,self.prossimo_da_servire-self.bollino-1)
```

Così facendo si rischia di avere più thread il cui valore locale di `miobollino` è sovrapposto.
Può essere un ottimo esercizio riflettere sul perché questo potrebbe portare una squadra a perdere il turno per sempre.

Errore 2. Non fare l'esercizio.

```
miobollino = trovaValoreBollino()
```

Se non si fornisce l'implementazione di `trovaValoreBollino()` l'esercizio è valutabile tanto quanto

```
int main()
{
     return faiTuEFaiTutto();
}
```

è un ottimo tool di riconoscimento vocale.

Errore 3. Wait su condition in sequenza.

Bisogna sempre essere molto prudenti quando nel codice si introducono due wait di fila sulla stessa condition, poichè *potrebbe* essere un errore. Esempio:

```
while(self.palleDisponibili < numGiocatori)
      self.condition.wait()
while(self.__cercaPista() == -1)
      self.condition.wait()
```

Siamo sicuri che, mentre si è in attesa sulla seconda wait, per qualche motivo (le azioni di un'altra squadra, ad esempio) la condizione di attesa della prima wait non sia più *False?* Chi ci dice che `self.palleDisponibili` sia ancora >= di `numGiocatori`?

Errore 4. Introdurre una nuova struttura dati ma non consultarla mai.

Alcuni hanno avuto l'intuizione di modificare il meccanismo di assegnazione dei turni attraverso l'incremento di self.bollino e self.prossimoDaServire, introducendo una lista di attesa. In generale se si aggiunge una struttura dati S al proprio codice, dovete avere bene in mente:

1. Quando inizializzate S; (CREATE)
2. Quando aggiornate S e perchè; (UPDATE)
3. Quando **CONSULTATE** il contenuto di S nel contesto della logica del vostro codice; (READ)
4. Quando S termina il suo ciclo di vita. (DELETE)

Se create una lista di attesa, la riempite di valori, ma non ne prelevate mai, o non guardate mai dentro la lista agendo di conseguenza, c'è evidentemente qualcosa che non va!