Corso di Sistemi Operativi e Reti

Prova scritta 13 LUGLIO 2021

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.

SALVA SPESSO

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, PUNTI DA 1 a 4 - PROGRAMMAZIONE MULTITHREADED

Punto 1

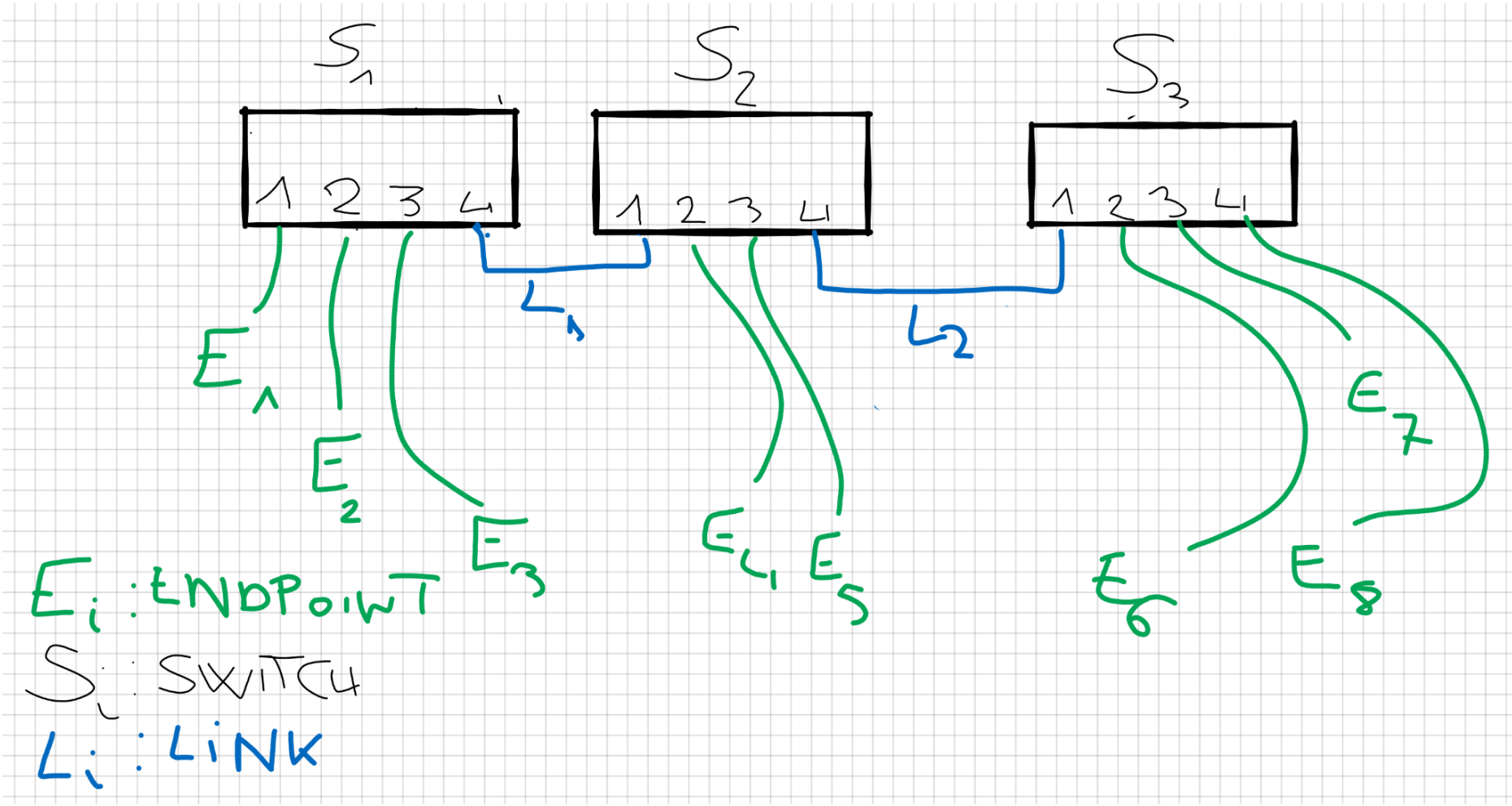
Si aggiunga al codice esistente la classe `LinkSimulator`. Tale classe è costruita indicando due switch S1 ed S2, e due porte fisiche P1 e P2 appartenenti rispettivamente a S1 e S2. Quando attivo, un LinkSimulator deve prelevare il traffico in uscita da P1 e inserirlo su P2, e viceversa.

Punto 2

Si aggiunga al codice esistente la classe `EndPointSimulator`. Un `EndPointSimulator` è costruito indicando uno Switch S, una porta fisica P di quest'ultimo, a cui l'endpoint è connesso, e un MAC address sorgente M. Quando un `EndPointSimulator` è attivo, esso genera, in tempi casuali, dei frame casuali con MAC sorgente M e MAC Destinazione scelto casualmente tra tutti i Mac Address facenti parte della simulazione. Tali frame vengono posti nel buffer di ingresso della porta P. Un `EndPointSimulator` inoltre riceve e stampa a video tutti i Frame in uscita dalla medesima porta P.

Punto 3

Si modifichi il codice principale esistente per simulare una configurazione con 3 switch da 4 porte ciascuno connessi come in figura:


S1
S2
S3
1 2 3 4
1 2 3 4
1 2 3 4
E1
E2
E3
E4
E5
E6
E7
E8
L1
L2
Ei : ENDPOINT
S : SWITCH
Li : LINK

Punto 4:

Si noti che nel codice fornito il metodo `Switch.__getFrame()` restituisce `None` se ogni buffer di ingresso è vuoto. Si migliori il metodo `Switch.__getFrame()` in maniera tale da andare correttamente in attesa bloccante fino a quando non c'è almeno un Frame disponibile in un qualche buffer di ingresso, restituendo dunque sempre un Frame valido.

MATERIALE PER LA PROVA SULLA PROGRAMMAZIONE MULTI-THREADED

ESERCIZIO 1 (Programmazione multithread. Punti: 0-20)

Il codice fornito simula il comportamento di uno switch hardware. Uno `switch` consiste di *N* `porte` e di K thread `Worker` che elaborano internamente i frame in ingresso alle singole porte. Una `porta` è composta da 2 Blocking Queue distinte, una di uscita e una di ingresso. Ciascuna blocking queue può contenere un certo numero di `Frame`. Un frame è così composto:

```
class Frame
{
      String MAC_Sorgente;
      String MAC_Destinazione;
      String messaggio;
}
```

La classe switch fornisce i seguenti due metodi pubblici:

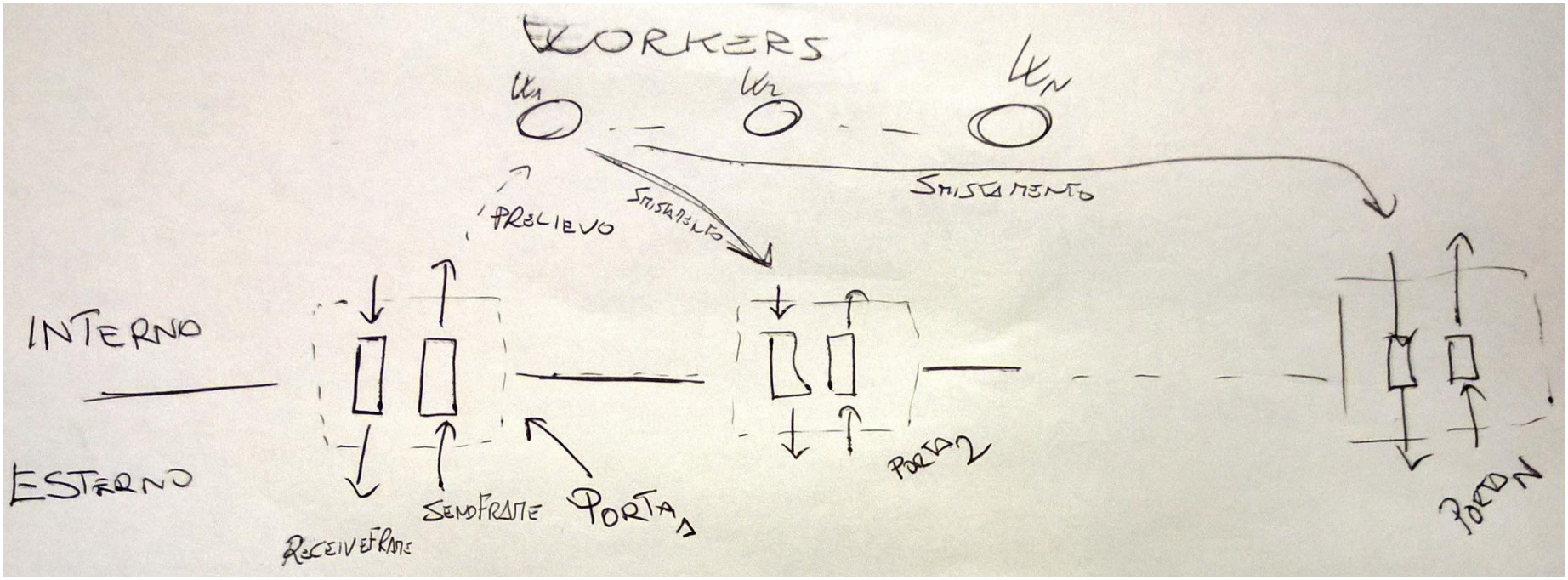
`sendFrame(F : Frame, P : Porta)`. Inserisce il frame F nella porta di indice P. Il metodo è bloccante nel caso in cui il **buffer di input** della porta P è pieno.

`receiveFrame(int P) -> Frame`. Preleva un frame dalla porta di indice P. Il metodo è bloccante nel caso in cui il **buffer di output** della porta P è vuoto.

Un Worker lavora internamente allo switch usando i metodi privati `__getFrame` e `__putFrame` e ripete continuamente le seguenti istruzioni:

1. **Preleva** un frame `F` da una delle N **code di ingresso**;
2. **Inserisce** `F` in una o tutte le **code di uscita** secondo i seguenti criteri, determinati in base al valore di `F.MAC_Destinazione`:
    a. Se il valore `F.MAC_Destinazione` non è mai comparso in passato nel ruolo di MAC Sorgente, allora bisogna inserire `F` in tutte le code di uscita, ad esclusione della coda da cui `F` è stato ricevuto;

b. Se il valore `F.MAC_Destinazione` è già comparso in altre occasioni nel ruolo di MAC sorgente, durante l'elaborazione di un precedente frame, allora bisogna inserire `F` nella coda di uscita dell'ultima porta da cui, in ordine temporale, è stato ricevuto un frame G con `G.MAC_Sorgente == F.MAC_Destinazione`.



Si noti che i Worker ESTRAGGONO dai buffer di INGRESSO delle porte, e INSERISCONO nei buffer di USCITA delle porte. Pertanto un Worker non usa `sendFrame` e `receiveFrame`, poichè questi metodi, rispettivamente INSERISCONO nei buffer di INGRESSO delle porte ed estraggono dai buffer di USCITA (l'esatto opposto del lavoro di elaborazione di un Worker che lavora internamente a uno switch).

Segue l'implementazione delle strutture dati fin qui descritte. Sono inclusi una classe `FrameGenerator` e del codice di prova.

```python
from threading import RLock, Thread
from queue import Queue,Empty
from random import random,randint
from time import sleep

class Frame:
    def __init__(self,s,d,m):
        self.MAC_Sorgente = s
        self.MAC_Destinazione = d
        self.messaggio = m
        #
        # Porta di provenienza. Valorizzata da un worker al momento del prelievo
        #
        self.__provenienza = None


class Porta:
    def __init__(self):
        self.N = 20
        # Buffer di ingresso
        self.in_ = Queue(self.N)
        # Buffer di uscita
        self.out = Queue(self.N)


class Switch:

    def __init__(self,NPorte,NWorker):
        self.switchTable = {}
        self.lock = RLock()
        self.porte = [Porta() for i in range(0,NPorte)]
        self.workers = [Worker(self) for i in range(0,NWorker)]
        for i in range(0,NWorker):
            self.workers[i].start()
```

```
def __aggiornaChiave(self, s, p):
    with self.lock:
        self.switchTable[s] = p

def __leggiChiave(self, s):
    with self.lock:
        return self.switchTable.get(s)

def sendFrame(self,f : Frame, p : Porta):
    p.in_.put(f)

def receiveFrame(self,p : Porta):
    return p.out.get()

def __putFrame(self, f : Frame):
    port = self.__leggiChiave(f.MAC_Destinazione)
    if port != None:
        port.out.put(f)
        print("UNICAST %s : %s->%s = %s" % (port,f.MAC_Sorgente,f.MAC_Destinazione,f.messaggio))
    else:
        print("BROADCAST")
        for i in range(0,len(self.porte)):
            if self.porte[i] != f.__provenienza:
                self.porte[i].out.put(f)
                print("%s : %s->%s" % (self.porte[i],f.MAC_Sorgente,f.MAC_Destinazione))


def __getFrame(self):
    for i in range(0,len(self.porte)):
        try :
            f = self.porte[i].in_.get_nowait()
            if f != None:
                f.__provenienza = self.porte[i]
                self.__aggiornaChiave(f.MAC_Sorgente, f.__provenienza)
                return f
        except Empty:
```

```
                pass
        return None


class Worker(Thread):

    def __init__(self, s : Switch):
        super(Worker, self).__init__()
        self.s = s

    def run(self):
        while True:
            f = self.s._Switch__getFrame()
            if f != None:
                self.s._Switch__putFrame(f)



destinations = [ "01:01:01:01:01:01",
                 "01:01:01:01:01:02",
                 "01:01:01:01:01:03",
                 "01:01:01:01:01:04",
                 "01:01:01:01:01:05"]

class FrameGenerator(Thread):

    def __init__(self,s : Switch, porta : Porta, mac : str):
        super(FrameGenerator, self).__init__()
        self.porta = porta
        self.mac = mac
        self.s = s

    def run(self):
        while(True):
            sleep(random())
            self.s.sendFrame(Frame(self.mac,destinations[randint(0,4)],"Lorem Ipsum:"+chr(randint(64,84))),self.porta)
```

```
if __name__ == '__main__':
    cisco2900 = Switch(5,2)
    generators = [FrameGenerator(cisco2900,cisco2900.porte[i],"01:01:01:01:01:0"+chr(ord("1")+i)) for i in range(0,5)]
    for gen in generators:
        gen.start()
```

SALVA SPESSO

SALVA SPESSO

ESERCIZIO 2, TURNO 1 - PERL

Si crei uno script perl dal nome `reverso.pl`, in grado di tradurre una o più parole da una lingua ad un'altra facendo uso di un file di dizionario passato come parametro allo script.

Lo script sarà invocato nel seguente modo

```
reverso.pl OPZIONI [path/to/file_dictionary] [string]
```

Il parametro `OPTION` è obbligatorio, mentre i parametri `path/to/file_dictionary` e il parametro `string` sono in genere opzionali, ma in base all'opzione utilizzata diventano o meno obbligatori (**si veda sotto**).

I possibili valori del parametro `OPZIONi` sono i seguenti:

- **`-r`**

  Quando si usa l'opzione `-r` il parametro `path/to/file_dictionary` è obbligatorio mentre il parametro `string` rimane opzionale. Bisogna effettuare la traduzione di una o più parole separate da virgola "," passate come parametro (nel caso sia presente il parametro `string`) oppure sarà necessario ricevere l'input delle parole da tradurre tramite standard input (STDIN).

  Nota che il match all'interno del file dictionary dovrà essere ***case insensitive*** (non si fa distinzione tra lettere maiuscole e minuscole).

  `Esempio:`

  Lo script potrà essere lanciato nel seguente modo:

  ```
  ./reverso.pl -r it_en "ricciolo,ricordare"
  ```

e dovrà produrre in output:

```
La traduzione di ricciolo è:  CURL;  LOCK;  LOVELOCK;  RINGLET; CURLY
La traduzione di ricordare è:  REMEMBER; RECALL; MENTION; MIND; REMIND
```

alternativamente sarà possibile eseguire lo script nel seguente modo:

```
./reverso.pl -t it_en
```

lo script continuerà chiedendo all'utente di inserire in input una serie di parole separate da virgola

```
Programma: "Inserire le parole da tradurre separate da virgola (,):"

Utente:     "ricciolo,ricordare"
```

ed infine dovrà produrre in output:

```
La traduzione di ricciolo è:  CURL;  LOCK;  LOVELOCK;  RINGLET; CURLY
La traduzione di ricordare è:  REMEMBER; RECALL; MENTION; MIND; REMIND
```

- **-h**

Conta la frequenza delle parole tradotte dalla prima all'ultima esecuzione dello script e le stampa in output in ordine decrescente di frequenza e, a parità di frequenza, anche in ordine lessicografico della parola ricercata.

Esempio:

Si supponga di aver eseguito più volte lo script con parametro `-t` come di seguito:

```
./reverso.pl -t it_en "lei,riflesso,ciao"
```

```
./reverso.pl -t it_en "leguminoso,riflesso,restare"
./reverso.pl -t it_en "ciao,abbandono,dove"
```

Lo script dovrà produrre il seguente output:

```
ciao --> 2
riflesso --> 2
abbandono --> 1
dove --> 1
leguminoso --> 1
lei --> 1
restare --> 1
```

Si noti che la frequenza della parola **ciao** è la stessa della parola **riflesso** ma viene stampata prima **ciao** in quanto ha precedenza in ordine alfabetico (lo stesso vale per le parole con frequenza 1).