

Corso di Sistemi Operativi e Reti

Prova scritta 28 MARZO 2022

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.

CI SONO DEI PUNTI AMBIGUI NELLA TRACCIA? COMPLETA TU

È parte integrante di questo esercizio completare le specifiche date nei punti non esplicitamente definiti, introducendo nuove strutture dati, o estendendo quelle preesistenti laddove si ritenga necessario, risolvendo eventuali ambiguità. Si può cambiare il codice dei metodi esistenti dove serve.

POSSO CAMBIARE IL PROTOTIPO DEI METODI RICHIESTI O DI QUELLI ESISTENTI? NO

Non è consentito modificare il prototipo dei metodi se questo è stato fornito. Potete aggiungere qualsivoglia campo e metodo di servizio, e qualsivoglia classe ausiliaria, ma NON variare l'interfaccia dei metodi pubblici già specificati. Analogamente, i metodi esistenti possono essere modificati nel loro codice, ma non se ne deve cambiare il risultato finale o il significato.

CHE LINGUAGGIO POSSO USARE? PYTHON 3.X

Il linguaggio da utilizzare per l'implementazione è Python 3.6 o successivo. Ricorda che l'operatore di formattazione `f` (esempio, `f"Ciao sono la stringa {testo}"`) è disponibile solo dalla versione 3.6 di Python in poi, ma può essere sostituito con `"Ciao sono la stringa %s" % testo`

MA IL MAIN() LO DEVO AGGIORNARE? E I THREAD DI PROVA? SI

E' obbligatorio implementare esplicitamente IL codice di prova oppure modificare il codice di prova pre-esistente, e accertarsi che giri senza errori prima della consegna.

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

Il codice fornito implementa la funzione `contaPrimiMultiThread(min,max)` che restituisce quanti numeri primi ci sono nell'intervallo tra `min` e `max` (inclusi gli estremi).

Ad esempio `contaPrimiMultiThread(9,19) = 4`, poichè 11, 13, 17, 19 sono i numeri primi compresi nell'intervallo di input.

`ContaPrimiMultithread` fa uso di più thread, i quali si distribuiscono i numeri primi da verificare sfruttando un `DistributoreNumeri` e una `Barriera`.

Il thread display è gestito da un opportuno thread **separato** che aggiorna le informazioni a video quando necessario. Si è volutamente evitato di fare stampe a video direttamente nel corpo dei quattro metodi di cui sopra (ad esclusione delle stampe di debug). Solo il thread display può stampare.

Tutti i metodi richiesti sono stati implementati garantendo la necessaria thread safety; sono state evitate situazioni di deadlock; si è cercato di migliorare l'accessibilità concorrente alle strutture dati e si sono evitate potenziali situazioni di starvation. Queste garanzie dovranno ovviamente essere mantenute anche nel codice che svilupperai in sede di esame.

ESERCIZIO 1 - PROGRAMMAZIONE MULTITHREADED

Punto 1:

Si modifichi la logica di funzionamento del *DistributoreNumeri* in maniera tale da distribuire D numeri da verificare per volta. La quantità D, che di default vale 10, deve essere modificabile dinamicamente (e cioè anche durante la fase di calcolo dei Macinatori) attraverso il metodo thread safe `DistributoreNumeri.setQuantita(d);`

Invece, al posto del metodo `DistributoreNumeri.getNextNumber()` i Macinatori dovranno usare il metodo `DistributoreNumeri.getNextInterval()` che restituisce un intervallo composto da D numeri da far calcolare al Macinatore chiamante. L'intervallo assegnato può essere più piccolo di D nel caso in cui i numeri restanti da testare siano di meno.

Ad esempio, supponiamo che D=20 ed nthread=2.

Quando si invoca `contaPrimiMultiThread(101,175)`, il distributore di numeri assegnerà ai due Macinatori, mano a mano che questi ne fanno richiesta, gli intervalli (101,120), (121,140), (141,160), (161,175).

La modifica deve essere compatibile con eventuali Macinatori che continuino a usare `getNextNumber` (e cioè che continuano a prelevare un numero per volta)

Punto 2:

Si noti che ciascun Macinatore lavora su un suo totale parziale che viene infine usato per calcolare il totale finale con lo spezzone di codice:

```
totale = 0
for i in range(nthread):
    totale += ciucci[i].getTotale()
return totale
```

Si modifichi il codice in maniera tale che ciascun Macinatore, nell'arco del suo processo di calcolo, aggiorni direttamente una variabile `Totale` condivisa. Il ciclo di cui sopra dovrebbe dunque poter essere sostituito con qualcosa di simile a:

```
return Totale.getTotale()
```

che restituisce il totale globale senza la necessità di sommare tutti i totali parziali.

SALVA SPESSO

ESERCIZIO 2, TURNO 1 - PERL

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
```

Lo scopo dell'esercizio è quello di creare uno script Perl dal nome `dump.pl` che prenderà in input il nome del file di log (in questo caso `dump.log`) e 2 interi positivi S ed F , con $S \leq F$ ed $F < 24$.

Lo script dovrà essere quindi richiamato nel seguente modo:

```
./dump.pl dump.log 14 15
```

dove:

```
Nome File = dump.log
```

```
S = 14
```

```
F = 15
```

```
S ≤ F
```

Lo script dovrà essere in grado di rintracciare tutte le connessioni avvenute tra le ore S e le ore F .

Esempio:

Nel caso in cui lo script sia invocato con S = 14 ed F = 15, solo le connessioni avvenute tra le 14 e le 15 andranno prese in considerazione e tutte le altre andranno tralasciate

```
14:25:53.325453 IP 192.168.0.100.47172 > 216.58.205.200.443: UDP, length 1350 --> OK
15:34:07.952457 IP 31.13.86.36.443 > 192.168.0.100.54946: Flags [P.], seq 165408:165546, ack 20306, win 425, options [nop,nop,TS val 3121993885 ecr 78163997], length 138 --> OK
16:34:08.136165 IP 192.168.0.100.49736 > 216.58.205.195.443: UDP, length 41 --> NO
```

e infine produrre in output 2 file di testo che chiameremo `udp.log` e `flags.log` e che rispettino le specifiche riportate di seguito.

Il primo file (`udp.log`) conterrà il `TIMESTAMP` seguito da `IP Sorgente.PORTA > IP Destinazione.PORTA` di tutte le connessioni di rete che avranno usato il protocollo UDP ordinate cronologicamente.

Alla fine del file è necessario stampare il numero totale delle connessioni di cui si è tenuto traccia (vedi esempio).

Esempio del formato file `udp.log` (Nota l'ordine cronologico per data/ora):

```
14:25:51.932550 --> 160.97.62.90.62536 > 224.0.0.252.5355
14:25:52.895087 --> 172.217.23.106.443 > 192.168.0.100.41997
```

Totale: 2

Il secondo file (`flags.log`) conterrà il `TIMESTAMP` seguito da `IP Sorgente.PORTA > IP Destinazione.PORTA` di tutte le connessioni di rete che avranno usato un protocollo diverso da UDP ordinate in ordine cronologico inverso.

Alla fine del file è necessario stampare il numero totale delle connessioni di cui si è tenuto traccia (vedi esempio).

Esempio del formato file `flags.log` (Nota l'ordine cronologico per data/ora):

```
14:26:10.171155 --> 23.6.123.119.443 > 192.168.0.100.44664
14:25:53.357586 --> 23.6.123.119.52152 > 192.168.0.100.160.97.62.1.443
```

Totale: 2