

# Corso di Sistemi Operativi e Reti

Prova scritta 14 SETTEMBRE 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.

**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. 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 del codice di prova oppure modificare il codice di prova pre-esistente, e accertarsi che giri senza errori prima della consegna.

## ESERCIZIO 1 - PROGRAMMAZIONE MULTITHREADED

### Punto 1 (FACILE):

Si estenda il codice della `TimedBlockingQueue` con il metodo

```
getExpiryTime(e)
```

tale metodo restituisce il tempo approssimativo che manca alla scadenza dell'elemento `e`. Qualora l'elemento `e` non sia presente in coda, viene restituito il valore di `-1`.

**ESEMPIO:** sia `Q` una `TimedBlockingQueue`. Si supponga di eseguire, all'interno di un singolo thread, il seguente spezzone di codice:

```
Q.timedPut(Object, 5)
time.sleep(3)
t = getExpiryTime(Object)
```

Il valore di `t` dovrebbe essere di circa 2 secondi.

### Punto 2 (MEDIO):

Si estenda il codice della `TimedBlockingQueue` con il metodo

```
getNextTimeoutElement()
```

Tale metodo restituisce il riferimento al prossimo elemento `e` di cui è prevista la scadenza, **senza rimuovere l'elemento** dalla coda stessa.

### ESEMPIO:

Si supponga che all'interno della `TimedBlockingQueue Q` siano stati inseriti a più riprese vari elementi:

```
Q.timedPut(O1, 15)
time.sleep(1)
Q.timedPut(O2, 7)
time.sleep(4)
Q.timedPut(O3, 10)
```

Abbiamo dunque all'interno di `Q`, in ordine di estrazione FIFO: `O1`, `O2`, `O3`. Subito dopo l'inserimento di `O3`, abbiamo che:

- `O1` è stato inserito con 15 secondi di tempo di scadenza, dunque mancano circa 10 secondi alla sua rimozione automatica;
- `O2` è stato inserito con 7 secondi di scadenza, dunque mancano circa 3 secondi alla sua rimozione automatica;
- `O3` è stato inserito con 10 secondi di scadenza, i quali sono appena iniziati.

Dunque l'elemento più prossimo alla scadenza dovrebbe essere `O2`. Di conseguenza

```
E = getNextTimeoutElement()
```

dovrebbe porre `E = O2`

### Punto 3 (DIFFICILE):

Si estenda il codice della `TimedBlockingQueue` con il metodo

```
postpone (e, t)
```

Tale metodo, qualora l'elemento `e` sia ancora in scadenza all'interno della coda, ne aumenta il tempo di scadenza di `t` secondi, uscendo immediatamente e restituendo il valore `True`. Se l'elemento `e` non è presente in coda, il metodo restituisce immediatamente `False`.

ESEMPIO: Si supponga di eseguire, all'interno di un singolo thread, il seguente spezzone di codice:

```
Q.timedPut (O1, 5)  
time.sleep (4)
```

A questo punto l'oggetto `O1` dovrebbe trovarsi a un secondo dalla scadenza. Una immediata chiamata

```
Q.postpone (O1, 7)
```

dovrebbe riportare il tempo restante, prima della rimozione automatica dell'oggetto `1`, a circa 8 secondi.

### Suggerimenti:

Tutti e tre i punti si possono risolvere tenendo opportunamente traccia del tempo di inserimento di un elemento e della rispettiva scadenza. Per conoscere il tempo corrente espresso in secondi, si può usare la funzione `time()` appartenente al modulo `time`. Per conoscere il momento approssimativo di inserimento di un elemento, è sufficiente invocare questa funzione nel momento opportuno, ad es:

```
insertionTime = time()
```

Il momento approssimativo della scadenza di un certo elemento può essere ricavato facilmente sommando il valore noto di `timeOut` ad `insertionTime`.

Per sapere quanto tempo manca approssimativamente alla scadenza di un elemento, è sufficiente considerare

```
time() - insertionTime
```

# MATERIALE PER LA PROVA SULLA PROGRAMMAZIONE MULTI-THREADED

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

Il codice fornito implementa una `BlockingQueue` FIFO con i tradizionali metodi di inserimento e rimozione (`put` = inserimento bloccante; `get` = prelievo bloccante), modificata per fornire alcune funzioni aggiuntive. Questa nuova tipologia di coda è stata detta `TimedBlockingQueue`. Le funzioni aggiuntive sono le seguenti:

`timedPut(e, timeout)`.

Inserisce l'elemento `e` all'interno della `TimedBlockingQueue`. Qualora la coda sia piena, il codice attende finché non si libera un posto in coda, e a quel punto l'elemento `e` viene inserito secondo la disciplina FIFO usuale. Dal momento dell'inserimento in poi, `timedPut` termina ed esce. Tuttavia, la presenza dell'elemento `e` in coda è soggetta a una scadenza di `timeout` secondi. Qualora trascorressero più di `timeout` secondi senza che `e` venga prelevato da un qualche thread attraverso una operazione `get`, allora `e` viene cancellato automaticamente dalla coda.

`waitFor(e)`.

Se l'elemento `e` è inserito nella coda, questo metodo attende finché `e` non viene estratto da qualche thread, oppure finché `e` non viene rimosso a causa della scadenza di un eventuale periodo di `timeout`. Questo metodo restituisce `true` se l'uscita è stata causata dalla normale estrazione dell'elemento `e` dalla coda. Viene altrimenti restituito `false` se l'elemento `e` risulta eliminato per scadenza di un `timeout`. Viene inoltre restituito `false` immediatamente anche quando l'elemento `e` non è correntemente presente nella coda.

ESEMPI (Si supponga di lavorare con la coda `Q`, inizialmente vuota):

1. Supponiamo di inserire l'oggetto `t` all'interno di `Q` con `timeout` di `0.5` secondi, mediante la chiamata `Q.timedPut(t, 0.5)`. Questa operazione inserisce immediatamente `t` e termina dopo aver avviato un timer interno che durerà esattamente *0.5 secondi*. Durante questo intervallo di tempo, `t` potrebbe essere normalmente prelevato attraverso una chiamata al metodo `get`. Trascorsi gli `0.5` secondi `t` verrà invece eliminato dalla coda e non sarà più prelevabile.
2. Supponiamo che il thread `T1` inserisca l'oggetto `t` con la chiamata `Q.timedPut(t, 1.0)`, la quale inserisce l'elemento in `Q` ed esce. Successivamente il thread `T1` invoca `Q.waitFor(t)`. Quest'ultima chiamata si blocca in attesa. A questo punto potrebbe succedere che un thread `T2` invochi una operazione di `get` ed estragga `t`. In tal caso la chiamata a `Q.waitFor(t)` fatta da `T1` terminerebbe

restituendo `true`. Tuttavia, se `t` dovesse restare inserito in coda per più di un secondo senza essere estratto, `t` sarebbe eliminato dalla coda. T1 in questo caso esce dalla chiamata `Q.waitFor(t)`, ma il valore restituito dalla chiamata è `false`.

**SALVA SPESSO**

# SALVA SPESSO

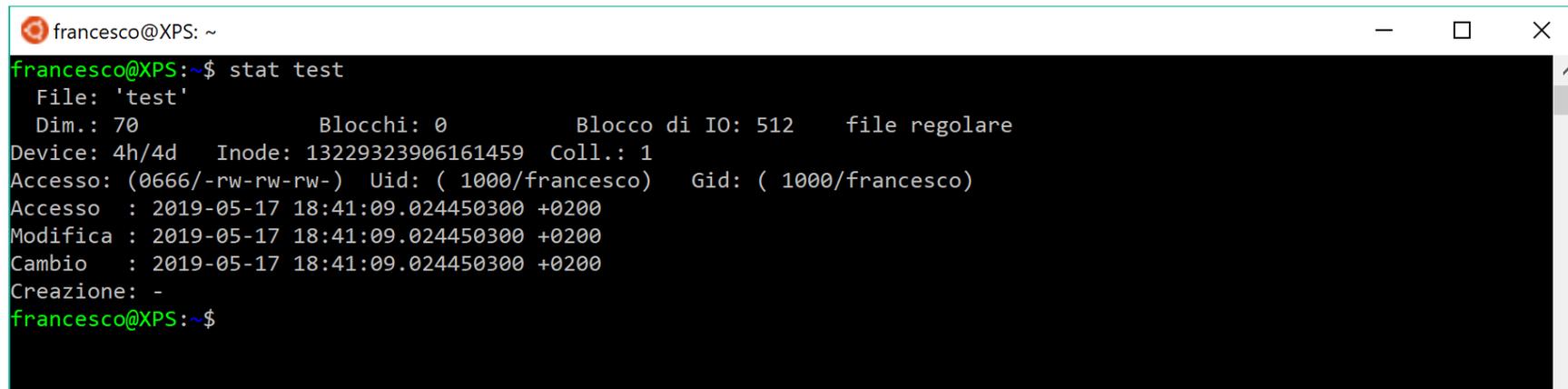
## ESERCIZIO 2, TURNO 1 - PERL

Nei sistemi Unix/Linux, il comando `stat` mostra informazioni dettagliate riguardo un determinato file o filesystem. Si scriva uno script Perl dal nome `info.pl` in grado di estendere le funzionalità del comando shell `stat`. In particolare, lo script prenderà da linea di comando una serie di opzioni (non obbligatorie) e il nome di una directory (obbligatoriamente).

**Sinossi del comando da implementare:**

```
./info.pl path/to/directory [OPTIONS]
```

**Output del comando `stat` standard impartito su un file di prova chiamato 'test':**



```
francesco@XPS: ~
francesco@XPS:~$ stat test
  File: 'test'
  Dim.: 70          Blocchi: 0          Blocco di IO: 512   file regolare
Device: 4h/4d  Inode: 13229323906161459  Coll.: 1
Accesso: (0666/-rw-rw-rw-)  Uid: ( 1000/francesco)  Gid: ( 1000/francesco)
Accesso  : 2019-05-17 18:41:09.024450300 +0200
Modifica : 2019-05-17 18:41:09.024450300 +0200
Cambio   : 2019-05-17 18:41:09.024450300 +0200
Creazione: -
francesco@XPS:~$
```

Sia  $SizeB$  la dimensione dei blocchi occupati da un singolo file  $F$ .  $SizeB$  è definito come il prodotto dei valori di  $Blocchi$  e  $Blocchi$  di IO ( $SizeB = Blocchi * Blocchi$  di IO).

Sia  $SizeFolder$  la dimensione dei blocchi occupati da tutti i file contenuti in una specifica cartella  $D$ .  $SizeFolder$  è definito come la

somma di  $SizeB$  per ogni file  $F$  contenuto all'interno di una cartella  $D$  ( $SizeFolder = \sum_{F \in D} SizeB_F \forall F \in D$ ).

Lo script dovrà operare nel seguente modo:

1. nel caso in cui non ci dovessero essere opzioni in input, lo script mostrerà in standard output il valore di `SizeFolder` della cartella `path/to/directory`. Infine, saranno riportati anche i nomi dei file con valore di `SizeB` minimo e massimo (seguiti dai rispettivi valori);
2. se si inserisce l'opzione `-b` lo script stamperà su un file dal nome `output.log` il nome di tutti i file contenuti all'interno della `directory` specificata seguiti dalla rispettiva `SizeB`; la stampa dovrà essere ordinata in ordine decrescente di valore e, a parità di `SizeB`, andrà seguito anche l'ordine lessicografico sul nome del file;
3. se si inserisce l'opzione `-t="tipoFile"` bisognerà effettuare la stessa operazione del punto 1 ma filtrando per i file specificati con l'opzione `-t`; se, ad esempio, usassimo l'opzione `-t="file regolare"` dovremo aggregare ed effettuare le somme solo sui file testuali (le informazioni sul tipo di file sono specificate subito dopo il parametro `Blocco di IO`);
4. NON SONO AMMESSE ALTRE OPZIONI.