Corso di Sistemi Operativi e Reti

Prova scritta 15 FEBBRAIO 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 del 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 realizza un sistema di gestione code di utenti in attesa presso una certa sede INPS. Una `Sede` è composta da N uffici/sportelli distinti, ciascuno identificato da una lettera progressiva (Ufficio 'A', Ufficio 'B', ...). Ciascun ufficio è dotato di una propria coda d'attesa gestita con dei ticket di prenotazione progressivi. Ad esempio i ticket rilasciati dall'ufficio B saranno nel formato "B001", "B002", ecc.

Un utente della sede preleva un ticket di accesso all'ufficio desiderato e attende di essere chiamato per poter essere servito. In ogni ufficio ci sono degli impiegati che servono i rispettivi utenti chiamandoli in base all'ordine di erogazione dei ticket di prenotazione; un thread display visualizza i codici degli ultimi 5 ticket che sono stati chiamati, indipendentemente dall'ufficio di appartenenza. Ad esempio, in un certo momento, sul display potrebbe essere visualizzata la sequenza:

B004
C076
B005
B006
D113

I metodi che una `Sede` implementa sono:

`prendiTicket(self,uff)`. Rilascia il prossimo ticket disponibile per l'ufficio `uff`. Restituisce una stringa contenente il codice del ticket rilasciato. Invocato tipicamente dagli utenti per prenotarsi e avere un numero in coda all'ufficio scelto.

`chiamaTicket(self,uff)`. Chiama il prossimo ticket non ancora servito per l'ufficio `uff`. Ciò che viene visualizzato sul display deve essere aggiornato in accordo. Si pone in attesa bloccante se il prossimo ticket da chiamare non è stato ancora rilasciato. Invocato tipicamente dagli impiegati degli uffici quando desiderano chiamare il prossimo utente.

`waitForTicket(self,ticket)`. Si mette in attesa che il ticket identificato da `ticket` venga chiamato. Esce quando `ticket` viene chiamato oppure se il ticket risulta tra gli ultimi 5 chiamati e visualizzati sul display. Un thread utente può, se vuole, utilizzare questo metodo per essere automaticamente svegliato nel momento in cui il proprio ticket viene chiamato.

`printAttese(self)`. Interrompe la visualizzazione corrente del display mostrando a video il numero di utenti in attesa per ciascun ufficio. Può essere utilizzato quando si desidera visualizzare un riepilogo della situazione degli utenti in coda.

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 noti che il metodo `waitForTicket(self,ticket)` può rimanere in attesa all'infinito qualora venisse invocato troppo in ritardo da un thread utente. In particolare questa anomalia si può verificare se `waitForTicket` viene invocata se `ticket` è stato ormai chiamato ed è infine sparito dalla lista degli ultimi cinque ticket chiamati. Si modifichi il codice di `Utente` in maniera tale da verificare che l'errore sia effettivamente possibile; si programmi quindi una nuova versione di `waitForTicket`, chiamata `waitForTicketSafe` **che abbia lo stesso identico comportamento** (e cioè si va in attesa bloccante finchè il ticket in input non viene chiamato), salvo che si restituisce `True` se il ticket è stato effettivamente chiamato oppure si trova nella lista degli ultimi ticket. Se invece `ticket` è stato chiamato da molto tempo, e dunque non risulta più negli ultimi cinque ticket chiamati, bisogna, anziché finire in attesa indefinita, restituire `False`.
Si programmi infine un `UtenteSafe` che usi il nuovo metodo `waitForTicketSafe` anziché il precedente `waitForTicket` e se ne verifichi il funzionamento corretto.

Punto 2:

Si noti che è possibile prendere ticket da più di un ufficio contemporaneamente, ad esempio lo spezzone di codice:

```
ticket1 = self.sede.prendiTicket("B"))
ticket2 = self.sede.prendiTicket("F"))
ticket3 = self.sede.prendiTicket("D"))
```

Consente di prelevare tre ticket da tre differenti uffici in sequenza. Si introduca il metodo

`waitForTickets(self, L : List)`

Questo metodo prende in input una lista `L` di ticket, e si mette in attesa contemporanea di tutti i ticket presenti in `L`. Il metodo esce restituendo `True` quando uno qualsiasi dei ticket presenti in `L` risulta presente in `self.ultimiTicket`.

Analogamente al metodo `waitForTicketSafe`, bisogna restituire `False` se *nessun* ticket presente in `L` risulta ormai chiamabile (poiché chiamato in passato e ormai non presente in `self.ultimiTicket`).
Si programmi un thread `UtenteFurbetto` pensato per testare il nuovo metodo.

Punto 3:

Si introduca il metodo `incDecSizeUltimi(self,n : int)`.

Tale metodo incrementa (o diminuisce, a seconda del segno di `n`) la dimensione di `self.ultimiTicket` .
Se `n < 0 and -n >= len(self.ultimiTicket)` l'operazione deve essere ignorata. Si verifichi che tutto il codice pre-esistente tenga conto del fatto che la taglia di `self.ultimiTicket` possa variare, facendo le eventuali modifiche. Si programmi un thread di test del nuovo metodo introdotto.

SALVA SPESSO

ESERCIZIO 2, TURNO 1 - PERL

Si scriva un script dal nome **`isOnline.pl`** in grado di controllare quali tra gli hosts attualmente presenti nella arp table del proprio device sono online. Lo script dovrà essere eseguito con la seguente sintassi:

`./isOnline.pl PATH_TO_FILE`

Lo script riceve come argomento obbligatorio il path ad un file contenente una serie di associazioni **`mac_address#nome_utente`**. Un possibile file di esempio è il seguente:

```
00:08:74:4C:7F:1D#Francesco
13:45:74:2E:7F:1C#Giovambattista
19:21:12:EE:6F:7A#Denise
…
```

In particolare, una volta avviato, lo script dovrà consultare la propria arp table tramite il comando shell `arp -an` e dovrà ricavare dall'output di questo comando le informazioni relative all'indirizzo ip e al mac address di ogni host. Un possibile output del comando `arp -an` è il seguente:

```
? (192.168.1.18) associato a 00:08:74:4C:7F:1D [ether] su enp1s0
? (192.168.1.113) associato a <incompleto> su enp1s0
? (192.168.1.130) associato a <incompleto> su enp1s0
? (192.168.1.4) associato a <incompleto> su enp1s0
? (192.168.1.99) associato a 13:45:74:2E:7F:1C [ether] su enp1s0
? (192.168.1.117) associato a <incompleto> su enp1s0
? (192.168.1.107) associato a 19:21:12:EE:6F:7A [ether] su enp1s0
… omissis …
```

Si noti che le righe da cui estrarre l'indirizzo ip e il mac address sono quelle evidenziate in rosso poiché sono le uniche a contenere sia l'informazione relativa all'indirizzo ip che al mac address. Tutte le altre righe **DEVONO** essere scartate.

Una volta ricavate queste informazioni, lo script dovrà controllare quali tra gli indirizzi ip precedentemente trovati sono attualmente online utilizzando il comando shell `ping -c1` seguito dall'indirizzo ip, (ad esempio, `ping -c1 192.168.1.18`).

Se il comando `ping` restituisce una stringa contenente la stringa `100% packet loss` allora l'indirizzo ip che si sta cercando di contattare non è attualmente raggiungibile ed è quindi da considerare **`offline`**.

Al termine di questa procedura, lo script stamperà su `stdout` i **`nomi degli utenti`** attualmente online ordinati lessicograficamente (dalla A alla Z) e, successivamente, tutti quelli offline ordinati in ordine lessicografico inverso (dalla Z alla A).
Si ricorda che, i nomi degli utenti sono ottenibili controllando l'associazione **`mac_address`** e **`nome_utente`** che si trova nel file passato tramite `argv` allo script.

`N.B. Tramite il comando arp -an si ottiene una corrispondenza tra indirizzi_ip e mac_address mentre il file passato in argv allo script contiene una corrispondenza mac_address e nome_utente.`