Računarske mreže, Ispit - JUN2 2023

Pročitati sve zadatke **pažljivo** pre rada - sve što nije navedeno ne mora da se implementira!
Na `Desktop`-u se nalazi zip arhiva. Unutar arhive se nalazi direktorijum u formatu `rm_rok_Ime_Prezime_mXGGXXX` u kome se nalazi validan IntelliJ projekat. Izvući direktorijum iz arhive na Desktop i ubaciti svoje podatke u ime. Otvoriti IntelliJ IDEA, izabrati opciju `Open project` (ne `Import project`!) i otvoriti pomenuti direktorijum. Sve kodove ostaviti unutar već kreiranih Java fajlova. **Kodovi koji se ne prevode se neće pregledati. Nepoštovanje formata ulaza/izlaza nosi kaznu od -10% poena na zadatku!**

1. **Tokovi podataka i niti (20p)**
   Napisati program koji ispisuje ukupan broj pojavljivanja zadatog HTML tag-a u svim HTML fajlovima sa zadatog spiska URL-ova.

   - U datoteci `urls.txt` unutar direktorijuma `tests` na Desktop-u se nalazi spisak URL-ova (po jedan u svakoj liniji). Koristeći odgovarajuće **baferisane** ulazne tokove pročitati sadržaj pomenutog fajla i ispisati broj linija u tom fajlu. (2p)
   - Za svaku pročitanu liniju fajla `urls.txt` kreirati novi URL objekat koristeći `URL` klasu. Preskočiti sve linije koje ne predstavljaju validan URL. (1p)
   - Za svaki validni URL proveriti protokol koji se koristi. Ukoliko je protokol `FILE` i ukoliko putanja vodi do HTML fajla (ekstenzija `.html`), kreirati zasebnu nit koja će otvoriti **baferisani** ulazni tok do tog fajla putem URL klase i pročitati sadržaj fajla (detalji obrade su u narednoj stavci). Kodnu stranu prilikom učitavanja postaviti na UTF-8. Ukoliko fajl na datoj putanji ne postoji, ispisati odgovarajuću poruku i ugasiti nit koja je pokrenuta da ga obradi. (5p)
   - Pre parsiranja fajla `urls.txt`, sa standardnog ulaza učitati jednu nisku koja predstavlja naziv HTML tag-a. Prebrojati koliko se puta zadati tag pojavljuje u svim fajlovima iz prethodne stavke tako što će svaka nit prebrojati pojavljivanja za fajl koji joj je dodeljen. Ispisati ukupan broj na standardni izlaz (videti primere ispisa ispod teksta zadatka). Pritom, paziti na sinhronizaciju niti ukoliko se koristi deljeni brojač. (5p)
   - Postarati se da program ispravno barata specijalnim slučajevima (npr. ako fajl ne postoji na datoj putanji) i ispravno zatvoriti sve korišćene resurse u slučaju izuzetka. (2p)

```
ulaz:  p
izlaz: lines:     29
       not found: /home/ispit/Desktop/tests/404.html
       result:    50

ulaz:  html
izlaz: lines:     29
       not found: /home/ispit/Desktop/tests/404.html
       result:    10

ulaz:  a
izlaz: lines:     29
       not found: /home/ispit/Desktop/tests/404.html
       result:    3
```

2. **TCP Sockets (20p)**
Implementirati server, koji će imati ulogu da održava *in-memory* tabelu šahista i njihove trenutne rejtinge. Tabela ima kolone: `id (int)`, `naziv (String)` i `elo (int)`.

- Napraviti Java aplikaciju koja ima ulogu klijenta. Povezati se na lokalni server na portu 1996 koristeći **Socket** klasu. Nakon formiranja konekcije, klijent može poslati više zahteva serveru (zahtevi se unose sa standardnog ulaza), sve dok mu ne pošalje `bye`. Odgovori servera na zahtev se ispisuju na standardni izlaz. Mogući zahtevi su (implementirati u ovoj stavci samo slanje od strane klijenta): (3p)
  - *sel id* (`id` je tipa `int`)
  - *ins naziv* (`naziv` je tipa `String`)
  - *upd id elo* (`id` i `elo` su tipa `int`)
- Napraviti Java aplikaciju koja ima ulogu servera. Pokrenuti lokalni server na portu 1996, koristeći **ServerSocket** klasu. Server za svakog primljenog klijenta pokreće zasebnu nit u kojoj će se taj klijent obraditi tako što se ispiše poruka o pristiglom klijentu u formatu: `konektovan klijent <IP>:<PORT>`. (2p)
- Server pristigle zahteve obradjuje na sledeći način:
  - *sel id*: vraća naziv i elo šahiste sa datim identifikatorom `id` (2p)
  - *ins naziv*: ubacuje u tabelu šahistu sa datim imenom dodeljujući mu jedinstveni identifikator (sledeći slobodan ceo broj) i elo u vrednosti 1300 (to je najmanja moguća vrednost za elo) i vraća poruku o uspešnosti operacije (2p)
  - *upd id deltae*: vrši izmenu elo vrednosti šahiste sa identifikatorom `id` za `deltae` i vraća poruku o uspešnosti operacije (2p)
- Ukoliko bilo koji od ovih zahteva nije ispravno formiran ili nije naveden iznad, vratiti tekst kao u primerima ispod. (1p)
- Imajte u vidu da mogu da se dese konfliktne situacije (kao npr. da dva klijenta žele da promene elo istoj osobi). Obezbediti da se ovakvi zahtevi pravilno obrade. Takođe, obezbediti da u slučaju izuzetaka, resursi budu ispravno zatvoreni. (3p)

```
> ins Magnus Carlsen
ins je uspesno izvrsen
> sel 1
Magnus Carlsen: 1300
> upd 1 30
upd je uspesno izvrsen
> sel 1
Magnus Carlsen: 1330
> upd 1 -10
upd je uspesno izvrsen
> sel 1
Magnus Carlsen: 1320
> bye
```

```
> ins Fabiano Caruana
ins je uspesno izvrsen
> sel 1
Fabiano Caruana: 1300
> upd 1 1500
upd je uspesno izvrsen
> sel 1
Fabiano Caruana: 2800
> upd 1 -10000
upd je uspesno izvrsen
> sel 1
Fabiano Caruana: 1300
> bye
```

```
> ins Marko
ins je uspesno izvrsen
> sel 1
Marko: 1300
// drugi klijent: upd
> sel 1
Marko: 1400
> ins Marko
ins je uspesno izvrsen
// nije isti Marko
> sel 2
Marko: 1300
> bye
```

Napomena: Ohrabrujemo studente da koriste ***`netcat`*** *kako bi testirali delimične implementacije i otkrili greške pre vremena. Takodje, ukoliko se npr. preskoči implementacija servera, može se mock-ovati server putem* ***`netcat`****-a.*

Okrenite stranu!

3. **UDP Sockets (20p)**
Implementirati klijent-server aplikaciju koja koristi *Java Datagram API*.

- Napisati Java klasu koja ima ulogu UDP klijenta. Klijent treba da sa standardnog ulaza učita nisku *str* koju želi da hešira i ceo broj n tipa `int` koji predstavlja broj rundi heširanja koji želi da se izvrši. Zatim klijent pakuje te podatke u datagram na sledeći način: prva 4 bajta predstavljaju bajtove broja n, a nakon toga slede bajtovi niske *str*. Ovakav datagram se šalje serveru. (4p)
- Nakon poslatog paketa, klijent čeka da dobije odgovor od servera najviše 5s. Ako klijent ne dobije odgovor u ovom roku, datagram se šalje ponovo, sve dok se ne dobije odgovor. (2p)
- Klijent očekuje da od servera dobije jedan ceo broj koji predstavlja heš vrednost. Broj možete kodirati proizvoljno (kao nisku ili kao niz bajtova). Ako je dobijena heš vrednost `-1`, na standardni izlaz ispisati *Greska*, a u suprotnom ispisati dobijenu heš vrednost. (1p)
- Napisati Java klasu koja ima ulogu UDP servera koji će da vrši heširanje. Server prima poruke od klijenata i iz njih parsira ceo broj n i nisku *str*. Ako je n negativno ili nula, server šalje `-1` kao odgovor klijentu.(3p)
- Server zatim vrši n rundi heširanja i to na sledeći način: prva runda računa heš vrednost [`str`, `portKlijenta`], a svaka naredna runda računa heš vrednost [`str`, `prethodniRezultat`]. `portKlijenta` predstavlja vrednost porta klijenta sa koga je paket došao. *Napomena: za računanje heš vrednosti [niska, broj] možete koristiti metodu Objects.hash(Object...)*. Na primer, ako sa porta 1234 dobijemo nisku "test"i treba da izvršimo 3 runde heširanja, prvo bismo računali `Objects.hash("test", 1234)` i dobili vrednost `110253633`. Zatim bismo računali `Objects.hash("test", 110253633)` i dobili `220506032`. Konačno, izračunali bismo `Objects.hash("test", 220506032)` i dobijenu vrednost 330758431 bismo poslali klijentu. (2p)
- Ako je dobijena heš vrednost `-1`, klijentu poslati vrednost `0`. U suprotnom, klijentu poslati dobijenu vrednost. Broj možete kodirati proizvoljno. Očekuje se da način kodiranja broja na klijentu i na serveru bude isti.(1p)
- Postarati se da se sve greške pravilno obrađuju i da se svi resursi zatvaraju. (2p)