Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (SI3OS2)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo
Kolokvijum: Prvi, oktobar 2017.
Datum: 27.10.2017.

Prvi kolokvijum iz Operativnih sistema 2

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 1,5 sat. Dozvoljeno je korišćenje literature.

Zadatak 1 _____/10 *Zadatak 3* _____/10
Zadatak 2 _____/10

Ukupno: _____/30 = _____%

Napomena: Ukoliko u zadatku nešto nije dovoljno precizno definisano, student treba da uvede razumnu pretpostavku, da je uokviri (da bi se lakše prepoznala prilikom ocenjivanja) i da nastavi da izgrađuje preostali deo svog odgovora na temeljima uvedene pretpostavke. Ocenjivanje unutar potpitanja je po sistemu "sve ili ništa", odnosno nema parcijalnih poena. Kod pitanja koja imaju ponuđene odgovore treba **samo zaokružiti** jedan odgovor. Na ostala pitanja odgovarati **čitko, kratko i precizno**.

1. (10 poena) Raspoređivanje procesa

U nekom sistemu klasa `Scheduler`, čija je delimična definicija data dole, realizuje raspoređivač spremnih procesa po prioritetu (engl. *priority scheduling*), tako da i operacija dodavanja novog spremnog procesa `put()` i operacija uzimanja spremnog procesa koji je na redu za izvršavanje `get()` imaju ograničeno vreme izvršavanja koje ne zavisi od broja spremnih procesa (kompleksnost $O(1)$ u odnosu na broj spremnih procesa). Postoji N redova spremnih procesa, pri čemu je N konfiguraciona konstanta. Red sa nižim indeksom ima viši prioritet (tj. red sa indeksom 0 je najvišeg prioriteta). Za izvršavanje se bira proces iz reda sa najvišim prioritetom, a unutar istog reda po *FCFS* redosledu.

Da bi sprečio izgladnjivanje procesa, raspoređivač ograničava vreme čekanja procesa na procesor na najviše `MaxWaitingTime` vremena od trenutka dolaska u red spremnih. To znači da će se za izvršavanje izabrati proces kome je isteklo ovo vreme (ako ima više takvih, onda prvi sa najvišim prioritetom), bez obzira na to što postoje spremni procesi višeg prioriteta.

U strukturi `PCB` postoji polje `next` kao pokazivač tipa `PCB*` koji služi za ulančavanje struktura PCB u jednostruke liste, celobrojno polje `prioririty` koje sadrži prioritet procesa, kao i polje `deadline` tipa `Time` koje služi za smeštanje apsolutnog vremena isteka roka čekanja. Na raspolaganju je apstraktni tip podataka `Time` koji predstavlja apsolutno vreme (vremenski trenutak izražen kao ceo broj milisekundi od nekog referentnog trenutka), kao i statička operacija `Time::now()` koja vraća trenutno vreme. Ukoliko su svi redovi prazni, operacija `Scheduler::get` treba da vrati 0. Zanemariti problem prekoračenja opsega tipa `Time`.

Realizovati u potpunosti klasu `Scheduler`.

```
class Scheduler {
public:
  Scheduler ();
  PCB* get ();
  void put (PCB*);
private:
  static const int N;
  static const Time MaxWaitingTime;
  ...
};
```

Rešenje:

2. (10 poena) Međuprocesna komunikacija pomoću deljene promenljive

Svaka instanca monitora `ResourceAllocator` kontroliše jedan deljeni resurs. Svaki proces koji želi da koristi taj resurs, mora najpre da ga alocira pozivom procedure `alloc` ovog monitora, a kada je završio korišćenje resursa, da ga oslobodi pozivom procedure `free` ovog monitora. Korišćenjem standardnih monitora i uslovnih promenljivih, implementirati monitor `ResourceAllocator` tako da dozvoli uporedno korišćenje deljenog resursa od strane najviše *N* uporednih procesa.

Rešenje:

3. (10 poena) Međuprocesna komunikacija razmenom poruka

Na programskom jeziku Java implementirati server za vođenje evidencije o rezervaciji karata. Data je delimična implementacija servera. Potrebno je implementirati delove koji nedostaju a označeni su sa #### (dozvoljeno je dodavati druge klase). Metoda `startServer()` prima kao parametar broj karata koje su na raspolaganju za rezervaciju. Klijenti od servera mogu da zahtevaju rezervaciju nekog broja karata. Server treba da vrati broj uspešno rezervisanih karata klijentu i zatvori prikljucnicu ka klijentu. Klijenti se kod servera indentifikuju IP adresom i portom na kojem imaju otvorenu serversku priključnicu preko koje mogu da komuniciraju, sve dok ne uspeju da rezervišu sve karte ili dok se ne završi rad servera. Isti klijent može proizvoljan broj puta da traži karte. Klijent može da otkaže rezervaciju određenog broja karata (smatrati da će otkazivati samo one karte koje je uspešno rezervisao i da neće otkazivati karte dok ne dobije rezervaciju svih karata koje je tražio). Karte koje su bile rezervisane mogu da se raspodele klijentima koji su tražili a nisu dobili karte, po *FCFS* redosledu. Protokol komunikacije sa klijentima osmisliti prema potrebi. Dozvoljena je upotreba koda sa vežbi (kod nije potrebno prepisivati, već samo precizno navesti koja klasa se koristi).

```
public class Server extends Thread {
    private boolean work;
    private int port;
    ####

    public Server(int port) {
        this.port = port;
    }

    public void run() {
        ServerSocket serverSocket = null;
        List<Thread> threads = new ArrayList<Thread>();
        try {
            serverSocket = new ServerSocket(port);

            while(work) {
                Socket clientSocket = null;
                try {
                    clientSocket = serverSocket.accept();
                    RequestHandler handler = ####;
                    threads.add(handler);
                    handler.start();

                }
                catch (IOException e) {
                    e.printStackTrace();
                }
            }

            waitThreads(threads);

            cleanUp();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void startServer(int n) {
        work = true;
        init(n);
        start();
    }

    public synchronized void stopServer() {
        work = false;
        interrupt();
    }
```

```
    private void init(int n) {
      ####;
    }

    private void cleanUp() throws IOException {
      ####;
    }

    private void waitThreads(List<Thread> threads) {
        for (Thread thread : threads) {
            try {
                thread.join();
            } catch (InterruptedException e) {
            }
        }
    }
}
```

Rešenje: