Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (IR3OS2)
Nastavnik: prof. dr Dragan Milićev
Odsek: Računarska tehnika i informatika
Kolokvijum: Prvi, novembar 2013.
Datum: 30.11.2013.

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 koristi se *Multilevel Feedback-Queue Scheduling* (MFQS) na sledeći način:

- Postoje tri reda spremnih procesa: HP (*High Priority*), MP (*Medium Priority*) i LP (*Low Priority*).
- Globalni algoritam raspoređivanja je po prioritetu, s tim da HP ima najviši, a LP najniži prioritet.
- Raspoređivanje u svim redovima je *Round-Robin* (RR), samo sa različitim vremenskim kvantumom koji se dodeljuje procesima. Procesima koji se uzimaju iz HP dodeljuje se vremenski kvantum `timesliceHP`, onima koji se uzimaju iz MP vremenskim kvantum `timesliceMP`, a onima iz LP kvantum `timesliceLP`.
- Proces koji je tek postao spreman (bio je blokiran ili je tek kreiran) smešta se u jedan od redova na osnovu procene dužine njegovog sledećeg naleta izvršavanja: ako je procenjena dužina sledećeg naleta manja ili jednaka od `tauHP`, smešta se u HP, inače, ako je manja ili jednaka od `tauMP`, smešta se u MP, inače se smešta u LP. Procena se vrši eksponencijalnim usrednjavanjem sa koeficijentom 1/2.
- Proces kome je istekao vremenski kvantum smešta se u prvi sledeći red nižeg prioriteta od onoga iz koga je bio uzet za izvršavanje.

Klasa `Scheduler`, čiji je interfejs dat dole, implementira opisani raspoređivač spremnih procesa. Implementirati u potpunosti ovu klasu tako da i operacija dodavanja novog spremnog procesa `put()` i operacija uzimanja spremnog procesa koji je na redu za izvršavanje `get()` budu ograničene po vremenu izvršavanja vremenom koje ne zavisi od broja spremnih procesa (kompleksnost $O(1)$). U slučaju da nema drugih spremnih procesa, operacija `get()` vraća proces na koga ukazuje `idle` (to je uvek spreman proces najnižeg prioriteta). Drugi argument operacije `put()` govori da li je proces postao spreman zato što je prethodno bio blokiran (`wasBlocked=1`) ili mu je istekao vremenski kvantum (`wasBlocked=0`).

U strukturi `PCB` postoje sledeća polja:

- Polje `time` predstavlja vreme proteklo u izvršavanju procesa (ukupno vreme korišćenja procesora u jednom naletu izvršavanja); svaki put kada procesu oduzme procesor, jezgro (van klase `Scheduler`) na ovu vrednost (kumulativno) dodaje vreme proteklo od kada je proces poslednji put dobio procesor (pre eventualnog poziva `Scheduler::put()`). Ovu vrednost treba resetovati kada proces postaje spreman nakon što je bio blokiran (u funkciji `Scheduler::put(wasBlocked==1)`).
- Polje `tau` predstavlja procenu dužine sledećeg naleta izvršavanja; ovu vrednost treba postaviti na novu procenu kada proces postaje spreman nakon što je bio blokiran (u funkciji `Scheduler::put(wasBlocked==1)`), a pre smeštanja u odgovarajući red. Pri kreiranju procesa polja `tau` i `time` su postavljena na unapred definisanu vrednost.
- Polje `timeslice` predstavlja vrednost dodeljenog vremenskog kvantuma; operacija `get()` postavlja ovo polje odabranog procesa na vrednost konstante koja odgovara vremenskom kvantumu za red iz koga je proces uzet.
- Polje `lastQueue` predstavlja red u kome je proces poslednji put bio i ne menja se između uzimanja i vraćanja procesa u red spremnih (između poziva `get()` i `put()`).
- Polje `next` je pokazivač na sledeći PCB u listi (za ulančavanje u jednostruke liste).

```
const int timesliceHP = ..., timesliceMP = ..., timesliceLP = ...,
          tauHP = ..., tauMP = ...;
extern PCB* const idle;

class Scheduler {
public:
  Scheduler ();
  PCB* get ();
```

```
  void put (PCB*, int wasBlocked);
private:
  PCB* head[3];  // 0-HP, 1-MP, 2-LP
  PCB* tail[3];
  static const int timeslice[3];
};

const int Scheduler::timeslice[] = {timesliceHP,timesliceMP,timesliceLP};

Scheduler::Scheduler () {
  for (int i=0; i<2; i++) head[i]=tail[i]=0;
}

PCB* Scheduler::get () {
  for (int i=0; i<2; i++)
    if (head[i]) {
      PCB* ret = head[i];
      head[i] = head[i]->next;
      if (head[i]==0) tail[i]=0;
      ret->next = 0;
      ret->timeslice = timeslice[i];
      ret->lastQueue = i;
      return ret;
    }
  return idle;
}
```

2. (10 poena) Međuprocesna komunikacija pomoću deljene promenljive

Na programskom jeziku Java, korišćenjem koncepta monitora podržanog od strane ovog jezika, realizovati klasu `Buffer` koji predstavlja neograničeni bafer i sadrži elemenate tipa `Object`. Na raspolaganju je kolekcija `List` koja predstavlja jednostruko ulančanu listu sa standardnim operacijama:

- `void add(Object o)` - umeće jedan element na početak liste.
- `Object remove()`- dohvata jedan element sa kraja liste.
- `boolean isEmpty()`- proverava da li je lista prazna.

Rešenje:

3. (10 poena) Međuprocesna komunikacija razmenom poruka

Na programskom jeziku Java implementirati veb servis koji omogućava udaljeni pristup fajl sistemu (fajl server) i pruža krajnjem korisniku sledeći interfejs:

```
public class FileClient extends Usluga {
 public FileClient(String host, int port);
 public String getFile(String file); //returns the file content
 public ArrayList<String> getIndex(String dir);//ret. the directory content
                                        //(list of files and subdirectories)
}
```

Servis treba da ima mogućnost opsluživanja više korisnika istovremeno. Prilikom pokretanja serverskog procesa zadaje se putanja koja predstavlja koreni direktorijum ovog fajl servera. Nadalje, svi korisnički zahtevi predstavljeni su relativnim pomerajem u odnosu na ovaj direktorijum. Operacije koje nudi fajl server su dohvatanje sadržaja jednog tekstualnog fajla i čitanje sadržaja direktorijuma (liste poddirektorijuma i svih fajlova koje sadrži). Grešku prilikom pristupa (nepostojanje ili pogrešan pristup fajlu/direktorijumu) treba prijaviti krajnjem korisniku. Na serverskoj strani na raspolaganju je klasa `File` koja ima sledeći interfejs:

```
public class File {
 public File(String pathname); //Creates a new File for the pathname
 public boolean exists();
 public boolean isDirectory();
 public String[] list() // Returns an array of strings naming the files and
           //directories in the directory denoted by this pathname
}
```

Međuprocesnu komunikaciju realizovati preko priključnica (*socket*) i razmenom poruka (*message passing*). Čitanje sadržaja fajla red po red vrši se pomoću operacije `readLine()` na isti način kao i prilikom pristupa ulaznom toku (*input stream*) priključnice. S tim što ova operacija u slučaju kraja fajla vraća `null` i takođe, objekat tipa `BufferedReader` potrebno je inicijalizovati na sledeći način:

```
new BufferedReader(new FileReader(File f));
```

Dozvoljeno je korišćenje koda prikazanog na vežbama (kod sa vežbi ne treba prepisivati, nego precizno navesti koja klasa ili koji metod se koriste i/ili menjaju, nasleđuju, ...). Nije potrebno proveravati uspešnost izvršavanja operacija (*try/catch* klauzule).

Rešenje: