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, decembar 2015.
Datum: 6.12.2015.

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 za simetrični multiprocesorski sistem primenjujući raspoređivanje 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()` na procesoru zadatom argumentom imaju ograničeno vreme izvršavanja koje ne zavisi od broja spremnih procesa (kompleksnost $O(1)$):

- Postoji P simetričnih (jednakih i ravnopravnih) procesora, pri čemu je P konfiguraciona konstanta.
- Za svaki od P procesora 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).
- U PCB procesa postoji polje `pri` koje predstavlja tekuću vrednost prioriteta procesa. Ova vrednost izračunata je pre nego što kernel pozove operaciju `Scheduler::put` da bi stavio proces u red spremnih.
- Novi spreman proces se smešta u red odgovarajućeg prioriteta i to za onaj procesor koji u svom redu datog prioriteta ima najmanji broj spremnih procesa od svih procesora. Na ovaj način se teži raspodeli opterećenja (engl. *load balancing*).
- Za izvršavanje na datom procesoru se bira proces iz reda sa najvišim prioritetom za taj procesor, a unutar istog reda po *FCFS* redosledu.
- Ukoliko su svi redovi prazni, operacija `Scheduler::get` treba da vrati 0.
- U strukturi `PCB` postoji polje `next` kao pokazivač tipa `PCB*` koji služi za ulančavanje struktura PCB u jednostruke liste.

Realizovati u potpunosti klasu `Scheduler`.

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

Rešenje:

2. (10 poena) Međuprocesna komunikacija pomoću deljene promenljive

Klasa `Gate`, čiji je interfejs dat dole, realizuje „kapiju“, jedan jednostavan koncept za međuprocesnu sinhronizaciju. Kapija može biti otvorena ili zatvorena, inicijalno je otvorena. Kapija se zatvara pozivom operacije `close()`, a otvara pozivom operacije `open()`. Kroz kapiju se mora proći pozivom operacije `pass()`; ako je kapija otvorena, pozivalac prolazi bez zaustavljanja (blokiranja), a ako je zatvorena, pozivalac se blokira sve dok se kapija ne otvori. Na jeziku Java implementirati klasu `Gate` sa potrebnom sinhronizacijom.

```
class Gate {
  public void close ();
  public void open ();
  public pass ();
}
```

Rešenje:

3. (10 poena) Međuprocesna komunikacija razmenom poruka

Na programskom jeziku Java implementirati sledeći serverski program koji radi kao osluškivač (engl. *listener*) na portu 6000.

Prilikom kreiranja servera, tj. instance klase `Server`, zadaje se niz stringova koji sadrže brojeve različitih portova. Na svakom od ovih portova osluškivaće po jedan osluškivač, tj. instanca klase `RequestListener`.

Server prima jedan po jedan zahtev sa klijenta na portu 6000, pri čemu je sadržaj svakog tog zahteva jedan ceo broj. Kada primi zahtev sa brojem *n*, server klijentu vraća broj porta *p* u ulazu *n* svog niza portova koji mu je zadat pri inicijalizaciji i odmah potom raskida vezu na portu 6000 (ukoliko je ceo broj *n* izvan granica niza, treba vratiti broj porta u ulazu 0). Kada primi ovaj odgovor sa brojem porta, klijent prelazi na komunikaciju preko „kanala“ na tom dobijenom portu *p*. Komunikacija sa svakim pojedinčanim klijentom na portu *p* treba da se obavlja konkurentno sa komunikacijom sa ostalim klijentima na istom portu, i tu komunikaciju treba da obavlja nit klase `RequestHandler`.

Implementirati sve tri pomenute klase. Konkretnu razmenu poruka u klasi `RequestHandler` nije potrebno implementirati, samo treba naznačiti mesto u programu gde se ona obavlja i treba pripremiti svu potrebnu infrastrukturu za nju.

Rešenje: