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 2012.
Datum: 1.12.2012.

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), uz dodatni mehanizam koji ga približava ponašanju SJF algoritma. Svakom procesu pridružena je procena trajanja narednog naleta izvršavanja τ koja je ceo broj.

- 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 je *Round-Robin* (RR), samo sa različitim vremenskim kvantumom koji se dodeljuje procesima.
- Proces se smešta u red prema proceni τ: ako je $\tau \leq 5$, smešta se u HP, ako je $5 < \tau \leq 10$, smešta se u MP, inače se smešta u LP.

Klasa `Scheduler`, čiji je interfejs dat dole, implementira opisani raspoređivač spremnih procesa. Implementirati 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, treba vratiti proces na koga ukazuje `idle` (to je uvek spreman proces najnižeg prioriteta). U strukturi `PCB` polje `tau` tipa `int` predstavlja procenu τ, a polje `next` pokazivač tipa `PCB*` koji služi za ulančavanje struktura PCB u jednostruke liste. Operacija `get()` treba da postavi polje `timeslice` u `PCB` odabranog procesa na vrednost konstante koja odgovara vremenskom kvantumu za red iz koga je proces izvađen. Vrednost `tau` procesa koji se smešta pomoću `put()` je već izračunata spolja pre poziva te operacije.

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

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

Rešenje:

2. (10 poena) Međuprocesna komunikacija pomoću deljene promenljive

Jedna varijanta uslovne sinhronizacije unutar monitora je sledeća. Svaki monitor ima samo jednu, implicitno definisanu i anonimnu (bez imena) uslovnu promenljivu, tako da se u monitoru mogu pozivati sledeće dve sinhronizacione operacije:

- `wait()`: bezuslovno blokira pozivajući proces i oslobađa ulaz u monitor;
- `notifyAll()`: deblokira *sve* procese koji čekaju na uslovnoj promenljivoj, ako takvih ima (naravno, mešusobno isključenje tih procesa je i dalje obezbeđeno).

Pomoću ovakvog monitora implementirati standardni brojački semafor sa inicijalnom vrednošću 1.

Rešenje:

3. (10 poena) Međuprocesna komunikacija razmenom poruka

U nekom sistemu mrtva blokada (*deadlock*) sprečava se zabranom kružnog čekanja tako što su resursi numerisani celim brojevima *0..RN-1*, pa svaki udaljeni proces sme da traži resurse samo u rastućem poretku ove numeracije. Konstanta *RN* je unapred poznata. Na programskom jeziku Java implementirati Veb servis koji će krajnjem korisniku pružiti sledeći interfejs:

```
class DeadlockPreventionException {
 public DeadlockPreventionException (int resourceID);
 public int getID();
}

public class NetResource {
 public NetResource(String host, int port, int resourceID)
 public void alloc () throws DeadlockPreventionException;
 public void dealloc ();
}
```

Servis treba da obezbedi usluge alokacije i dealokacije udaljenog resursa, pri čemu operacija `alloc()` treba da baci izuzetak datog tipa u slučaju prekršaja navedenog pravila sprečavanja mrtve blokade, u suprotnom obavlja potrebne radnje alokacije resursa sa zadatom numeracijom `resourceID`. Operacija `dealloc()` oslobađa dati resurs. `NetResource` se povezuje sa odgovarajućim serverom preko koga se vrši sinhronizacija, ali sama provera pravila sprečavanja mrtve blokade se obavlja na klijentskoj strani. Na serverskoj strani na raspolaganju je klasa `Resource`:

```
class Resource {
 private boolean allocated = false;
 public synchronized void alloc() {
   while (allocated) wait();
   allocated = true;
 }
 public synchronized void dealloc() {
  allocated = false;
  notifyAll();
 }
}
```

Međuprocesnu komunikaciju realizovati preko priključnica (*socket*) i razmenom poruka (*message passing*). 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: