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 2013.
Datum: 27.10.2013.

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 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 HP.
- Proces kome je istekao vremenski kvantum smešta se u MP ako je prethodno bio uzet iz HP, odnosno u LP ako je prethodno bio uzet iz MP ili 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 `lastQueue` tipa `Queue` predstavlja red u kome je proces bio kada mu je dodeljen procesor i ne menja se tokom izvršavanja tog procesa, između poziva `get()` i `put()`. 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. 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`).

```
const int timesliceHP = ..., timesliceMP = ..., timesliceLP = ...;
enum Queue {HP,MP,LP};
extern PCB* const idle;

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

Rešenje:

2. (10 poena) Međuprocesna komunikacija pomoću deljene promenljive

Korišćenjem klasičinih monitora i uslovnih promenljivih, realizovati monitor `Mutex` da dve operacije `lock` i `unlock`, koji se može koristiti za međusobno isključenje. Proces na ulasku u kritičnu sekciju zaštićenu jednom instancom ovog monitora treba da pozove `lock`, a na izlasku `unlock`.

Rešenje:

3. (10 poena) Međuprocesna komunikacija razmenom poruka

Potrebno je realizovati serverski proces na programskom jeziku Java koji ima mogućnost izvršavanja zadataka sa zadatim prioritetima, tako da raspoređivanje izvršavanja ovih zadataka se vrši po prioritetima (*priority schceduling*), pri čemu je, radi sprečavanja izgladnjivanja, potrebno implementirati mehanizam starenje (*aging*) na sledeći način: svaki put kada se neki zadatak izabere za izvršavanje, ostali u listi spremnih koji još čekaju dobijaju za jedan viši nivo prioriteta. U kontekstu serverskog procesa svaki zadatak izvršava se u vidu jedne niti tipa `WorkerThread` čiji je interfejs i deo realizacije dat u nastavku.

```
public class WorkerThread extends Thread {
 public static final int maxPri = ...;
 protected int taskId, priority;
 protected PrintWriter outStream;
 protected boolean finished = false;

 public WorkerThread(int taskId, int priority, PrintWriter outStream) {...}

 public void run() {
  //... Running task with "taskId"
  synchronized(this){ finished = true; notifyAll();}
 }
 public synchronized void waitToFinish(){
  while(!finished) try { wait(); } catch (InterruptedException e) {...}
}
```

Argument `taskId` označava jedinstveni identifikator zadatka, `priority` zadati početni prioritet, a `outStream` izlazni tok (*stream*) priključnice po kojoj je zahtev poslat. Klijentski proces može proizvoljno zadati više ovakvih zahteva za izvršavanje, pa tek nakon toga čekati na odgovore. Poruka koja predstavlja jedan zahtev od klijenta je u sledećem formatu "`#taskId#priority#`". U kontekstu serverskog procesa posebna nit je zadužena za raspoređivanje pristiglih zahteva koja se nakon započinjanja izvršavanja zadatka blokira pozivom njegove metode `waitToFinish`, a po završetku obaveštava klijenta slanjem odgovora koji sadrži `taskId` zadatka koji je upravo završen. Za realizaciju skupa zadataka na raspolaganju je klasa `ArrayList<WorkerThread>` sa standardnim metodama `add(WorkerThread)` i `remove(WorkerThread)` za smeštanje odnosno uzimanje jednog elementa iz liste, kao i dohvatanje elementa na zadatoj poziciji bez izbacivanja `WorkerThread get(int index)`. Ova klasa nije predviđena za konkurentno pozivanje (*thread-safe*), a obratiti pažnju takođe da njenu strukturu nije dozvoljeno menjati za vreme obilaska njenih elemenata. 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: