Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (SI3OS2, IR3OS2)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Prvi, septembar 2013.
Datum: 20.8.2013.

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 klasa `Scheduler`, čiji je delimična implementacija 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 $O(1)$). Promenljiva `maxPri` služi da ubrza pristup do najprioritetnijeg procesa. Između spremnih procesa sa istim prioritetom raspoređivanje je *FCFS*. Konstanta `MAXPRI` je maksimalna vrednost prioriteta (prioriteti su u opsegu `0..MAXPRI`, s tim da je 0 najniži prioritet). 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 `priority` tipa `int` predstavlja prioritet procesa, a polje `next` pokazivač tipa `PCB*` koji služi za ulančavanje struktura PCB u jednostruke liste.

Realizovati operaciju `Scheduler::get()`.

```
const int MAXPRI = …;
extern PCB* idle;

class Scheduler {
public:
  Scheduler ();
  PCB* get ();
  void put (PCB*);
private:
  PCB* head[MAXPRI+1];
  PCB* tail[MAXPRI+1];
  int maxPri;
};

Scheduler::Scheduler () : maxPri(-1) {
  for (int i=0; i<MAXPRI+1; i++)
    head[i]=tail[i]=0;
}

void Scheduler::put (PCB* pcb) {
  if (pcb==0) return; // Exception!
  int p = pcb->priority;
  if (p<0 || p>MAXPRI) return; // Exception!
  pcb->next = 0;
  if (tail[p]==0)
    tail[p] = head[p] = pcb;
  else
    tail[p] = tail[p]->next = pcb;
  if (p>maxPri) maxPri=p;
}
```

Rešenje:

2. (10 poena) Međuprocesna komunikacija pomoću deljene promenljive

Korišćenjem klasičnih uslovnih promenljivih, napisati kod za monitor `account` koji realizuje bankovni račun sa promenljivom stanja na računu (suma novca trenutno raspoloživa na računu). Monitor ima dve operacije u svom interfejsu:

- `credit(amount:real)`: na račun dodaje dati iznos;
- `debit(amount:real)`: sa računa skida dati iznos, ali tek kada na računu ima dovoljno novca (više ili jednako traženom iznosu).

Rešenje:

3. (10 poena) Međuprocesna komunikacija razmenom poruka

Na programskom jeziku Java, koristeći priključnice (*sockets*) i mehanizam prosleđivanja poruka (*message passing*), realizovati sistem koji udaljeno računa zbir zadatih redova matrice. Sistem se sastoji od jednog procesa servera i jednog procesa klijenata. Proces server u svom kontekstu izvršava više niti-radnika (*workers*) koje su realizovane po principu skupa zadataka (*bag-of-tasks*), gde svaka nit-radnik se izvršava na sledeći način:

```
while (true) {
 // dohvati jedan zadatak (indeks reda matrice) iz skupa zadatih
 if (nema preostalih zadataka) break;
 //izvrši zadatak, tj. sumiraj zadati red i upisi rezultat
}
```

Na ovaj način redovi matrice se sumiraju konkurentno tako što svaka nit-radnik uzima po jedan zadatak i izvršava ga. Klijentski proces na početku šalje poruku za započinjanje (*start*) zajedno sa brojem niti-radnika za izvršavanje, nakon čega sledi poruka za računanje (*calculate*) sa brojevima indeksa redova matrice koje treba sumirati. Serverski proces zatim ove indekse smešta u skup zadataka (*bag-of-tasks*) nad kojim pokreće potreban broj niti-radnika. Po završetku izračunavanja serverski proces vraća rezultat klijentskom procesu. Primer klijentskog procesa dat je u nastavku:

```
PrintWriter out = ...; BufferedReader in = ...;
out.println("#start#4#"); //zapocni operaciju i zadaj br. niti-radnika
int N = Integer.parseInt(in.readLine()); //dohvati br. redova matrice

String indexes = ""; //postavi indekse redova koji se sumiraju
for (int i = 0; i < N; i++) if(i%2==0) indexes+=i+"#";
out.println("#calculate#"+indexes);  //posalji zahtev

System.out.println("Result :"+in.readLine()); //dohvati rezulat
```

Pretpostaviti da serverski proces „osluškuje“ port 1033 preko koga prima zahteve i da matrica nad kojom se vrše izračunavanja je inicijalizovana i poznatih dimenzija. Za realizaciju skupa zadataka na raspolaganju je klasa `List<Integer>` sa metodama `add()` i `remove(0)` za smeštanje odnosno uzimanje jednog elementa iz liste, koje su predviđene za konkurentno pozivanje (*thread-safe*). Takođe, na raspolaganju je metoda `countTokens()` klase `StringTokenizer` koja vraća broj preostalih prepoznatih tokena u nizu koji se obrađuje. Napisati kompletan kod serverskog procesa i niti-radnika. Nije potrebno proveravati uspešnost izvršavanja operacija (*try/catch* klauzule).

Rešenje: