Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2
Nastavnik: prof. dr Dragan Milićev
Odsek: Računarska tehnika i informatika
Kolokvijum: Prvi, novembar 2022.
Datum: 4. 12. 2022.

Prvi kolokvijum iz Operativnih sistema 2

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 90 minuta. 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 raspoređivanje procesa koje je aproksimacija „potpuno pravednog“ algoritma (*Completely Fair Scheduler*, CFS) implementirano klasom `Scheduler` datom u nastavku. Za svaki proces se u polju `execTime` njegovog PCB-a čuva ukupno vreme izvršavanja na procesoru otkako je taj proces poslednji put došao u red spremnih. Kada se u red spremnih proces stavlja iz stanja suspenzije operacijom `Scheduler::put`, argument je `wasBlocked=true`; kada se u red spremnih proces stavlja zato što mu je istekao vremenski odsečak ili je proces pozvao sistemski poziv ili napravio izuzetak, pa ga sistem ponovo stavlja u red spremnih, argument je `wasBlocked=false`, a argument `execTime` predstavlja izmereno vreme trajanja završenog naleta izvršavanja. Kako bi se operacije `put` i `get` klase `Scheduler` implementirale ograničeno po vremenu izvršavanja vremenom koje ne zavisi od broja spremnih procesa (kompleksnost $O(1)$), spremni procesi se razvrstavaju po svom ukupnom vremenu izvršavanja tekućeg naleta (`PCB::execTime`) u `MaxPri+1` kategorija sa rezolucijom `TIME_PRI_RESOL`.

Data implementacija klase `Scheduler` nema podršku za dodelu vremenskog odsečka odabranom procesu po CFS algoritmu. Dopuniti ovu klasu ovom podrškom tako što u polje `PCB::timeSlice` procesa koje je odabran za izvšravanje treba upisati vrednost dodeljenog vremenskog odsečka. Na raspolaganju je statička funkcija `Time::getTime()` koja vraća trenutno realno procesorsko vreme tipa `Time`, što je neki dovoljno velik celobrojni tip sa dovoljno velikom rezolucijom (npr. vreme u nanosekundama proteklo od određenog trenutka). Može se proširivati struktura PCB potrebnim poljima. Jasno naglasiti dodate ili izmenjene delove koda.

Rešenje:

```
const Time TIME_PRI_RESOL = ...;

class Scheduler {
public:
  Scheduler () : count(0) {}
  PCB* get ();
  void put (PCB*, bool wasBlocked, Time execTime=0);

private:
  class ProcList {
  public:
    ProcList () : head(0), tail(0) {}
    void put (PCB* p);
    PCB* get ();
  private:
    PCB *head, *tail;
  };

  ProcList ready[MaxPri+1];
  unsigned long count;
};

inline void Scheduler::ProcList::put (PCB* p) {
  if (tail) tail = tail->next = p;
  else head = tail = p;
  p->next = 0;
}

inline PCB* Scheduler::ProcList::get () {
  PCB* ret = head;
  if (head) head = head->next;
  if (!head) tail = 0;
  return ret;
}

PCB* Scheduler:: put (PCB* p, bool wasBlocked, Time execTime) {
  if (pcb==0) return; // Exception!
  if (wasBlocked)
    pcb->execTime = 0;
  else
    pcb->execTime += execTime;
  unsigned long pri = pcb->execTime/TIME_PRI_RESOL;
  if (pri>MaxPri) pri = MaxPri;
  ready[pri].put(p);
  count++;
}

PCB* Scheduler::get () {
  for (int i=0; i<=MaxPri; i++) {
    PCB* p = ready[i].get();
    if (p) {
      count--;
      return p;
    }
  }
  return 0;
}
```

2. (10 poena) Međuprocesna komunikacija pomoću deljene promenljive

U školskom jezgru implementira se podrška za monitore i klasične uslovne promenljive. Data je implementacija klase `MonitorController` koja je namenjena za kontrolu međusobnog isključenja u monitoru: pri svakom ulasku u monitor koji kontroliše jedan objekat ove klase treba pozvati operaciju `entry`, a pri svakom izlasku operaciju `exit` ove klase. Data je i definicija klase `Condition` koja treba da implementira uslovne promenljive monitora koga kontroliše objekat klase `MonitorController` zadat kao argument konstruktora. Implementirati operacije `wait` i `signal` klase `Condition`, tako da nit koja je uradila `signal` na uslovnoj promenljivoj čeka da nit koja je time deblokirana prva završi svoju operaciju monitora.

Rešenje:

```
inline void Thread::dispatch () {
  Thread* oldT = Thread::running;
  Thread* newT = Scheduler::get();
  Thread::running = newT;
  Thread::yield(oldT,newT);
}

class MonitorController {
public:
  MonitorController () : isOpen(true), waitingToContinue(0) {}
  void enter ();
  void exit ();

protected:
  friend class Condition;
  void open ();

private:
  bool isOpen;
  Queue waitingEntry;
  Thread* waitingToContinue;
};

inline void MonitorController::enter () {
  if (isOpen)
    isOpen = false;
  else {
    waitingEntry.put(Thread::running);
    Thread::dispatch();
  }
}

inline void MonitorController::open () {
  Thread* t = waitingEntry.get();
  if (t)
    Scheduler::put(t);
  else
    isOpen = true;
}

inline void MonitorController::exit () {
  if (waitingToContinue) {
    Scheduler::put(waitingToContinue);
    waitingToContinue = 0;
  } else
    open();
}


class Condition {
public:
  Condition (MonitorController* m) : mc(m) {}
  void wait ();
  void signal ();

private:
  MonitorController* mc;
  Queue waitingCond;
};
```

3. (10 poena) Međuprocesna komunikacija razmenom poruka

Napisati kod klijenta koji komunicira sa više servera. Klijent treba da ima metodu:

```
void processDataFromServer();
```

koja asinhrono pokreće komunikaciju sa jednim serverom. Komunikacija se sastoji od traženja podataka i primanja podataka. Sve podatke koji se prime treba spojiti u jedan string i proslediti metodi:

```
void processData(String data);
```

koja je data i ne treba je realizovati.

U klasi klijenta postoji već inicijalizovan niz:

```
static String[] SERVERS = {"srv1.etf.bg.rs", "srv2.etf.bg.rs"};
```

koji sadrži adrese dostupnih servera. Svi serveri pružaju identičnu uslugu. Klijent treba podjednako da koristi oba servera u cilju postizanja podjednakog opterećenja svih servera. Svi serveri osluškuju na portu 5555.

Rešenje: