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, decembar 2021.
Datum: 05. 12. 2021.

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

Neki sistem primenjuje MQS raspoređivanje procesa sa dinamičkom promenom prioriteta na sledeći način. Klasa `GlobalScheduler` realizuje globalni algoritam raspoređivanja. Postoji `N_SCHED_CLASS` kategorija (klasa) procesa (`N_SCHED_CLASS` je mala celobrojna konstanta) razvrstanih po svom trenutnom tekućem dinamičkom prioritetu zapisanom u polju `pri` tipa `unsigned` strukture PCB (niža vrednost označava viši prioritet). Za svaku kategoriju procesa primenjuje se drugačiji algoritam raspoređivanja koji implementira jedan objekat klase izvedene iz klase `Scheduler`, a koji se registruje pozivom operacije `register` za datu kategoriju procesa; ova operacija poziva se prilikom inicijalizacije sistema, čime se konfigurišu raspoređivači za različite kategorije procesa.

Granice opsega prioriteta za kategorije procesa definisane su konstantama u nizu `maxPri`: najvišoj kategoriji 0 pripadaju procesi sa tekućim prioritetima u opsegu počev od 0 zaključno sa `maxPri[0]`, itd, kategoriji *i* pripadaju procesi sa tekućim prioritetima u opsegu počev od `maxPri[i-1]+1` zaključno sa `maxPri[i]`. Procesi iz niže kategorije se raspoređuju samo ako nema spremnih procesa iz neke više kategorije. Da bi se delimično izbeglo izgladnjivanje, procesi koji su dobili procesor, pa im je on oduzet zbog isteka vremena, ne smeju ponovo da dobiju procesor dok svi ostali spremni procesi ne dobiju procesor barem po jednom. (Izgladnjivanje uzrokovano stalno nadolazećim novoaktiviranim procesima ne treba tretirati.)

Tokom izvršavanja procesu se menja tekući dinamički prioritet tako što se nakon reaktivacije (kada proces stiže u red spremnih iz stanja suspenzije) prioritet povišava za jedan (u funkciji `GlobalScheduler::put(wasBlocked==1)`), a nakon oduzimanja procesora zbog isteka vremena prioritet spušta za 1 (u funkciji `GlobalScheduler::put(wasBlocked==0)`; vrednost `pri` se menja obrnuto po svojoj vrednosti), s tim da proces ne može napustiti svoju kategoriju prioriteta (prioritet procesa ne može izaći iz opsega prioriteta njegove kategorije).

Dat je interfejs klase `Scheduler` koji zadovoljavaju svi raspoređivači za različite kategorije procesa. Operacija `Scheduler::clone` pravi i vraća identičnu praznu kopiju (bez eventualnih procesa u redu) datog raspoređivača. Data je i delimična definicija klase `GlobalScheduler`. Implementirati klasu `GlobalScheduler` u potpunosti.

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

class GlobalScheduler {
public:
  GlobalScheduler () {}

  void register (Scheduler* s, unsigned priClass);
  PCB* get ();
  void put (PCB*, bool wasBlocked);

private:
  static const unsigned maxPri[N_SCHED_CLASS];
  ...
};
```

Rešenje:

2. (10 poena) Međuprocesna komunikacija pomoću deljene promenljive

Na jeziku C++ realizuje se monitor `MessageQueue` za razmenu poruka između uporednih niti. Poruke su nizovi znakova (završeni znakom `'\0'`) proizvoljne, ali ograničene dužine. Kada pošiljalac pošalje poruku operacijom `put`, sadržaj poruke kopira se u red poruka `MessageQueue`, koji poruke ulančava u listu elemenata tipa `Message`. Prostor za smeštanje poruka u redu treba alocirati dinamički, bibliotečnom funkcijom `malloc`. Red poruka prima poruke sve dok ima raspoložive memorije, odnosno dok `malloc` uspeva da alocira prostor. Kada memorije nema dovoljno, pošiljalac se blokira u pozivu operacije `put` sve dok se iz reda ne izvadi jedna ili više poruka i tako obezbedi potreban memorijski prostor. Primaoci uzimaju poruke operacijom `get` i sami su odgovorni za to da obezbede dovoljan prostor za smeštanje poruke na mesto na koje ukazuje parametar `buffer`. Ukoliko je red prazan, primalac se blokira dok u red ne stigne neka poruka.

Dat je interfejs klase `Semaphore` u školskom jezgru. Operacija `value` vraća trenutnu vrednost semafora (može biti negativna ako ima niti blokiranih na semaforu), a operacija `signalWait` radi izolovano (nedeljivo) operaciju `signal` na prvom i `wait` na drugom datom semaforu. Korišćenjem ovih semafora, implementirati u portpunosti klasu `MessageQueue` čija je delimična definicija data.

```
class Semaphore {
public:
  Semaphore (unsigned val=1);
  int value () const;
  void wait ();
  void signal ();
  static signalWait (Semaphore* toSignal, Semaphore* toWait);
  ...
};

class MessageQueue {
public:
  MessageQueue ();
  void put (const char* message);
  void get (char* buffer);
private:
  struct Message {
    Message* next;
    char* msg;
  };

  Message *head, *tail;
  ...
};
```

Rešenje:

3. (10 poena) Međuprocesna komunikacija razmenom poruka

Data je klasa `Data` sa sledećim interfejsom:

```
private static class Data {
        public void update(String data);
        public void draw();
    }
```

Metode `update()` i `draw()` nisu bezbedne za rad u višenitnom okruženju (tj. nisu thread-safe).

Napisati kod servera koji na zahtev klijenata poziva metodu `update()`, a metodu `draw()` poziva u određenim trenucima. Sever sadrži samo jedan objekat klase Data. Klijent može da traži samo jednu operaciju u toku jedne sesije. String koji se prosleđuje metodi `update()` je jedan red nekog teksta koji ne sadrži znak `#`. Server kada završi operaciju samo treba klijentu da pošalje poruku `Ok`. Ukoliko se u toku deset sekundi ne pojavi ni jedan klijent, server treba da pozove metodu `draw()`. Za vreme izvršavanja metode `draw()` server treba da i dalje osluškuje na serverskoj priključnici i obrađuje klijente. Server osluškuje na portu 5555.

Rešenje: