Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo
Kolokvijum: Prvi, novembar 2021.
Datum: 13. 11. 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.

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. Data je i delimična definicija klase `GlobalScheduler`. Implementirati klasu `GlobalScheduler` u potpunosti.

```
class Scheduler {
public:
  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

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. Klasa `Mutex` predviđena je za korišćenje u monitorima kako je pokazano na primeru klase `BoundedBuffer` za ograničeni bafer.

a)(5) Korišćenjem semafora sa datim interfejsom, implementirati klasu `Condition` koja realizuje klasičnu uslovnu promenljivu predviđenu za korišćenje u monitorima, kao što je dato u primeru klase `BoundedBuffer`.

b)(5) Implementirati operacije `put` i `get` klase `BoundedBuffer`. Ne treba implementirati sam pristup podacima u baferu (mesta označena sa ...), već samo sinhronizaciju potrebnu na mestima označenim numerisanim komentarima između `/*` i `*/`.

```
class Semaphore {
public:
  Semaphore (unsigned val=1);
  int value () const;

  void wait ();
  void signal ();
  static signalWait (Semaphore* toSignal, Semaphore* toWait);
  ...
};

class Mutex {
public:
  Mutex (Semaphore* s) : sem(s) { sem->wait(); }
 ~Mutex (Semaphore* s)          { sem->signal(); }
private:
  Semaphore *sem;
};

class Condition {
public:
  Condition (Semaphore* mutex);
  void wait ();
  void signal ();
private:
  ...
};

class BoundedBuffer {
public:
  BoundedBuffer () : numberInBuffer(0) {}

  void  put (Data* d);
  Data* get ();

private:
  static const unsigned BUFFER_SIZE = ...;
  unsigned numberInBuffer;

  Semaphore mutex;
  Condition slotAvailable, itemAvailable;
  ...
};
```

```
void BoundedBuffer::put (Data* d) {
  Mutex dummy(&mutex);
  /* 1 */
  numberInBuffer++;
  ...
  /* 2 */
}

Data* BoundedBuffer::get () {
  Mutex dummy(&mutex);
  /* 3 */
  numberInBuffer--;
  Data* d = ...;
  ...
  /* 4 */
  return d;
}
```

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()` u okviru jednog objekta klase nisu bezbedne za rad u višenitnom okruženju (tj. nisu thread-safe). Metode `update()` i `draw()` različitih objekata su bezbedne za rad u višenitnom okruženju.

Napisati kod servera koji sadrži proizvoljan broj objekata klase `Data` i na zahtev klijenata poziva metode `update()` i `draw()`. Svaki objekat klase `Data` ima jedinstveno ime. Klijent mora dostaviti ime objekta klase `Data` nad kojim želi da izvrši neku operaciju. Ukoliko ne postoji objekat sa traženim imenom, server treba da napravi novi objekat sa tim imenom prilikom poziva metode `update()`. Smatrati da će svaki klijent prvo pozvati metodu `update()`. String koji se prosleđuje metodi `update()` je jedan red nekog teksta koji ne sadrži znak `#`. Server kada završi neku operaciju samo treba da pošalje klijentu poruku `Ok`. Klijent u toku jedne sesije može zadati proizvoljan broj operacija. Server treba zahteve različitih klijenata da obrađuje paralelno. Server osluškuje na portu 5555.

Rešenje: