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 2024.
Datum: 30. 11. 2024.

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 po prioritetu. Klasa `Scheduler`, čiji su interfejs i deo implementacije dati dole, realizuje raspoređivač spremnih procesa. Spreman proces smešta se u jednu od `MAX_PRI+1` lista koje su elementi niza `ready` prema njegovom tekućem prioritetu (operacija `Scheduler::put`). Svaki put kada se proces smesti u neki red, u njemu se zabeleži (operacijom `PCB::setTimestamp`) trenutno vreme (`Timing::getTime()`) tipa `Time`; ovaj tip je neoznačen celobrojni tip koji predstavlja broj nanosekundi od momenta uključenja računara, dovoljno velikog opsega da ne može doći do prekoračenja. Operacija `PCB::getTimestamp` vraća ovako zabeleženo vreme.
Da bi se izbeglo izgladnjivanje, sistem povremeno poziva operaciju `Scheduler::aging`. Ova operacija treba da svaki proces koji je u jednoj listi datog prioriteta proveo više od `TIME_THRESHOLD` vremena prebaci u red za jedan većeg prioriteta. Implementirati ovu operaciju.

```
class Scheduler {
public:
  Scheduler () {}
  PCB* get ();
  void put (PCB* p);
  void aging ();

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

  static const int MAX_PRI;
  ProcList ready[MAX_PRI+1];
};

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;
}

inline void Scheduler::put (PCB* p) {
  p->setTimestamp(Timing::getTime());
  ready[p->getPriority()].put(p);
}
```

Rešenje:

2. (10 poena) Međuprocesna komunikacija pomoću deljene promenljive

Data je klasa `ReadersWriters` sa operacijama `startRead`, `stopRead`, `startWrite`, `stopWrite` koje implementiraju protokol više čitalaca – jedan opisac (*multiple readers – single writer*). Pokazati kako bi neki pretprocesor teksta mogao da dopuni programski kod za klasu `Resource` i njene operacije čitanja i izmena nekog deljenog resursa, predstavljene operacijama `opRead` i `opWrite`, tako da se na ove operacije primenjuje pomenuti protokol i da se resurs oslobodi (pozivom operacija `stopRead` tj. `stopWrite`) bez obzira na to kako se ove operacije napuštaju (naredbom `return` sa izrazom ili podignutim izuzetkom). Implementirati svu podršku potrebnu za tako umetnut kod.

```
class ReadersWriters {
public:
  ReadersWriters ();
  void startRead();
  void stopRead();
  void startWrite();
  void stopWrite();
  ...
};

class Resource {
public:
  Resource ();
  void opRead (...);
  void opWrite (...);
private:
  ...
};

void Resource::opRead (...) {
  ...
}

void Resource::opWrite (...) {
  ...
}
```

Rešenje:

3. (10 poena) Međuprocesna komunikacija razmenom poruka

Napisati kod klijenta koji pristupa serveru. Server omogućava slanje poruka između dva klijenta. Klijent se prijavljuje serveru i tom prilikom mu dostavlja svoje korisničko ime. Nakon prijavljivanja može da koristi server. Klijent može da pošalje poruku bilo kom prijavljenom korisniku. Poruka ne može sadržati znakove # i prelazak u novi red. Svaku poruku koju primi klijent treba da je ispiše na standardnom izlazu. Klijent sa standardnog ulaza čita kome i koju poruku treba da pošalje. U jednoj linije se nalazi ime korisnika i poruka. Slanje i primanje poruka klijent radi konkurentno. Server osluškuje na portu 5555 na lokalnom računaru.

Rešenje: