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 2022.
Datum: 7. 11. 2022.

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 koristi se raspoređivanje procesa koje je aproksimacija „potpuno pravednog“ algoritma (*Completely Fair Scheduler*, CFS) na sledeći način. 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` sa argumentom `wasBlocked=true`, ovo polje se resetuje. 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 (`wasBlocked=false`), na ovo ukupno vreme dodaje se vreme koje je proces koristio procesor u tom izvršavanju; to vreme zadato je parametrom `execTime` operacije `Scheduler::put`.

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`: ako je to ukupno vreme manje od konstante `TIME_PRI_RESOL`, kategorija je 0; ako je veće ili jednako `TIME_PRI_RESOL` i manje od `2*TIME_PRI_RESOL`, kategorija je 1 itd., ako je veće ili jednako `MaxPri*TIME_PRI_RESOL`, kategorija je `MaxPri`.

Klasa `Scheduler`, čiji je interfejs dat dole, realizuje opisani raspoređivač spremnih procesa. Implementirati u potpunosti ovu klasu tako da i operacija dodavanja novog spremnog procesa `put()` i operacija uzimanja spremnog procesa koji je na redu za izvršavanje `get()` budu ograničene po vremenu izvršavanja vremenom koje ne zavisi od broja spremnih procesa (kompleksnost $O(1)$). U slučaju da nema drugih spremnih procesa, operacija `get()` vraća *null*. Polje `next` strukture `PCB` služi za ulančavanje u liste.

```
const unsigned long TIME_PRI_RESOL = ...;

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

Rešenje:

2. (10 poena) Međuprocesna komunikacija pomoću deljene promenljive

Korišćenjem semafora u školskom jezgru napravljena je dole prikazana klasa `Condition` koja obezbeđuje koncept uslovne promenljive za korišćenje unutar klasa koje realizuju monitore u korisničkom programu. Svaka takva klasa koja realizuje monitor treba da sadrži semafor `mutex` i obezbedi međusobno isključenje ulaska u operacije monitora, kako je to pokazano na predavanjima. Uslovna promenljiva je jedan objekat klase `Condition` koji se inicijalizuje pokazivačem na ovakav semafor `mutex`. Statička operacija jezgra `Semaphore::signalWait` atomično (nedeljivo) radi operaciju `signal` na prvom i potom odmah operaciju `wait` na drugom datom semaforu; operacija `Semaphore::val` vraća trenutnu vrednost semafora (negativna ako na semaforu ima blokiranih niti). Dati koncizne, ali precizne i kompletne odgovore na sledeća pitanja:

a)(3) Zašto je u operaciji `Condition::wait` bilo potrebno pozvati `Semaphore::signalWait(mutex,&cond)`, a ne prosto posebno `mutex->signal()` pa potom `cond.wait()`?

b)(7) Navesti razlike i nedostatke prikazanog rešenja u odnosu na semantiku klasičnih uslovnih promenljivih.

```
class Condition {
public:
  Condition (Semaphore* mutex) : myMutex(mutex), cond(0) {}

  void wait ();
  void signal ();

private:
  Semaphore *myMutex, cond;
};

inline void Condition::wait () {
  Semaphore::signalWait(mutex,&cond);
}

inline void Condition::signal () {
  if (cond.val()<0) {
    cond.signal();
    mutex->wait();
  }
}
```

Rešenje:

3. (10 poena) Međuprocesna komunikacija razmenom poruka

Napisati kod servera koji čuva proizvoljan broj članaka. Članak se sastoji od naslova, koji je jedinstven, i teksta. Naslov i tekst ne sadrže znak za prelazak u novi red i znak #. Klijent od servera može da traži članak sa datim naslovom. Klijent može da dostavi serveru članak koji treba zapamtiti. Ako već postoji članak sa istim naslovom, novi članak se pamti kao nova verzija istog članka. Prilikom traženja članka klijent može dostaviti redni broj verzije koja mu je potrebna. Ako ne dostavi, server mu dostavlja najnoviju verziju. Kada uspostavi vezu sa serverom, klijent može tražiti proizvoljan broj usluga od servera. Server treba da ima mogućnost obrade zahteva od više klijenata uporedo. Server osluškuje na portu 5555. U slučaju bilo kakve greške prilikom dohvatanja članka potrebno je obavestiti korisnika i preći na sledeći zahtev.

Rešenj