Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (SI3OS2, IR3OS2)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Drugi, decembar 2021.
Datum: 05. 12. 2021.

Drugi 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) Mrtva blokada

U školskom jezgru implementiran je koncept `Mutex` za međusobno isključenje kritičnih sekcija niti na način dat dole. Ovu implementaciju potrebno je proširiti podrškom za sprečavanje mrtve blokade, tako što operacija `wait` treba da vrati pozivaocu -1 (grešku) ukoliko bi suspenzija niti napravila mrtvu blokadu. Drugim rečima, potrebno je ugraditi detekciju mrtve blokade koja bi nastala ukoliko bi se nit suspendovala i tada odbiti suspenziju vraćanjem greške iz poziva.

```
class Mutex {
public:
  Mutex () : val(1), holder(0) {}
  int wait();
  int signal();
protected:
  void block (); // Implementacija ista kao i za Semaphore
private:
  int val;
  ThreadQueue blocked;
  Thread* holder;
};

int Mutex::wait () {
  lock();
  if (--val<0) {
    block();
  } else
    holder = Thread::running;
  unlock();
  return 0;
}

int Mutex::signal () {
  if (holder!=Thread::running) return -1; // Error
  lock();
  if (val<0) {
    val++;
    Thread* t = blocked.get();
    holder = t;
    Scheduler::put(t);
  } else {
    val = 1;
    holder = 0;
  }
  unlock();
  return 0;
}
```

Rešenje:

2. (10 poena) Upravljanje memorijom

Neki sistem primenjuje lokalnu zamenu stranica FIFO algoritmom. Evidencija okvira vodi se u statičkom nizu `ProcessFrames::frames`, čiji je svaki element tipa strukture `FrameDesc`. Svakom okviru broj *i* fizičke memorije (u opsegu `0..NUM_FARMES`) odgovara *i*-ti element ovog niza. Polje `page` u strukturi `FrameDesc` sadrži broj stranice koja je smeštena u dati okvir. Svakom procesu pridružen je jedan objekat klase `ProcessFrames` koji implementira operacije potrebne za zamenu stranica datog procesa. Ovaj objekat realizuje FIFO red okvira. Elementi niza `frames` koji predstavljaju okvire datog procesa ulančani su dvostruko u kružnu listu tako što polja `prev` i `next` strukture `FrameDesc` sadrže indekse u nizu `frames` prethodnog, odnosno sledećeg elementa u listi. (Svaki proces ima svoju kružnu listu za zamenu sopstvenih stranica.) Atribut `cursor` sadrži indeks elementa koji je sledeći na redu za zamenu u FIFO poretku. Procesu je uvek dodeljen najmanje jedan okvir (dodeljuje se pri inicijalizaciji objekta `ProcessFrames`). Potpuna definicija klase `ProcessFrames` data je dole.

Implementirati sve operacije ove klase koje poziva ostatak kernela u odgovarajućim koracima postupka zamene stranica, tako da rade sledeće:

- `getPage` vraća broj stranice koja je smeštena u dati okvir;
- `setPage` postavlja broj stranice koja je smeštena u dati okvir;
- `getVictimFrame` vraća broj okvira u kome je stranica-žrtva koja je na redu za izbacivanje (bez promene kurzora);
- `replaceVictimFrame` obaveštava ovaj objekat da je stranica-žrtva zamenjena i da treba pomeriti kurzor;
- `addFrame` poziva kernel kada datom procesu dodeljuje nov okvir za njegove stranice.

```
class ProcessFrames {
public:
  ProcessFrames (size_t frame);

  static size_t getPage (size_t frame) const;
  static void   setPage (size_t frame, size_t page);

  size_t getVictimFrame () const;
  void   replaceVictimFrame ();
  void   addFrame (size_t frame);

private:
  struct FrameDesc {
    size_t page;
    size_t next, prev;
  };
  static FrameDesc frames[NUM_FRAMES];
  size_t cursor;
};
```

Rešenje:

3. (10 poena) Upravljanje memorijom

Neki sistem ima alokator paranjaka (*buddy*) koji alocira segmente veličine 2^n blokova veličine `BLOCK_SIZE`, gde je $0 \leq n \leq$ `MAX_BUCKET`. Ovaj alokator koristi se za potrebe alokacije ploča u alokatoru sa pločama (*slab*). Za keš čija je veličina jednog slota (pojedinačnog objekta koji se alocira) `slotSize`, veličina ploče određuje se tako da se može alocirati u alokatoru parnjaka, ali i tako da je od svih veličina segmenata od 2^n blokova koje može da alocira alokator parnjaka, preostali slobodan fragment (neiskorišćen za slotove) najmanji mogući. Implementirati pomoćnu operaciju koja se koristi za pronalaženje one veličine n koja daje takav najmanji fragment:

```
unsigned getBucket (size_t slotSize);
```

Rešenje: