Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

*Predmet:* Operativni sistemi 1 (SI2OS1, IR2OS1)
*Nastavnik:* prof. dr Dragan Milićev
*Odsek:* Softversko inženjerstvo, Računarska tehnika i informatika
*Kolokvijum:* Drugi, jun 2017.
*Datum:* 11.6.2017.

*Drugi kolokvijum iz Operativnih sistema 1*

*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 = _____% = _____/15

**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)

Školsko jezgro treba proširiti konceptom *barijere* („ograda“, „rampa“, engl. *barrier*) čiji je interfejs dat dole. Semantika ovog koncepta i operacija nad njim je sledeća:

- Barijera može biti u jednom od dva stanja: *otvorena* ili *zatvorena*. Barijera se inicijalizuje zadatim stanjem (argument konstruktora). Nit u čijem kontekstu se kreira barijera (tj. izvršava konstruktor) je njen *vlasnik*.
- `void Barrier::pass()`: ukoliko pozivajuća nit nije vlasnik ove barijere, ova operacija nema efekta – nit nastavlja svoje izvršavanje; ukoliko je pozivajuća nit vlasnik barijere, a barijera zatvorena, nit se suspenduje (blokira) sve dok se barijera ne otvori; ako je barijera otvorena, nit nastavlja izvršavanje bez blokade;
- `void Barrier::close()`: zatvara barijeru;
- `void Barrier::open()`: otvara barijeru i eventualno deblokira nit koja je na barijeri blokirana.

Implementirati ovu klasu `Barrier`. Ukoliko proširujete klasu `Thread`, precizno napisati kojim članovima.

```
class Barrier {
public:
  Barrier (int open=1);
  void open();
  void close();
  void pass();
};
```

Rešenje:

## 2. (10 poena)

U nastavku je data implementacija jednog programa koji ciklično vrši neku obradu podeljenu u dve faze. Prva faza obrade koristi (čita i menja) podatke koji se samo koriste u toj fazi; svi ovakvi podaci grupisani su u jednu veliku strukturu podataka tipa `Phase1Data`; ali koristi i podatke koji su zajednički za obe faze, odnosno preko kojih obrade u ove dve faze razmenjuju informacije; ovi podaci grupisani su u strukturu tipa `CommonData`. Analogno radi i obrada u drugoj fazi.

Sa ciljem značajnog smanjenja potrebe za operativnom memorijom, potrebno je restrukturirati ovaj program tako da koristi preklop (*overlay*) u koji se smeštaju podaci prve, odnosno druge faze koji se ne koriste istovremeno. Za čuvanje izbačenog sadržaja treba kreirati dva privremena fajla u tekućem direktorijumu procesa. Za rad sa fajlovima na raspolaganju su sledeći sistemski pozivi; sve ove funkcije vraćaju 0 u slučaju uspeha, a negativnu vrednost u slučaju greške:

- `FILE fopen(char* filename)`: otvara fajl sa zadatim imenom za čitanje i upis; ako fajl ne postoji, kreira ga;
- `int fread(FILE file, int offset, int size, void* buffer)`: iz datog fajla, sa pozicije rednog broja bajta `offset` (numeracija počev od 0), čita niz bajtova dužine `size` u memoriju na lokaciju na koju ukazuje `buffer`; ukoliko se čitanjem prekorači granica sadržaja fajla, vraća grešku;
- `int fwrite(FILE file, int offset, int size, void* buffer)`: u dati fajl, na poziciju rednog broja bajta `offset` (numeracija počev od 0), upisuje niz bajtova dužine `size` sa lokacije na koju ukazuje `buffer`; ukoliko se upisom prekorači granica sadržaja fajla, sadržaj fajla se proširuje tako da primi sav upisani sadržaj;
- `int fclose(FILE file)`: zatvara dati fajl.

```
struct Phase1Data p1data;
struct Phase2Data p2data;
struct CommonData cdata;

void main () {
  init_cdata(&cdata);   // Initialize common data (cdata)
  init_p1data(&p1data); // Initialize data for phase 1 (p1data)
  init_p2data(&p2data); // Initialize data for phase 2 (p2data)

  do {
    process_phase_1(&p1data,&cdata);  // Perform phase 1 processing
    process_phase_2(&p2data,&cdata);  // Perform phase 2 processing
  } while (!cdata.completed);
}
```

Rešenje:

**3. (10 poena)**

Virtuelni adresni prostor nekog sistema je 4GB i organizovan je stranično, adresibilna jedinica je bajt, a stranica je veličine 4KB. Fizički adresni prostor je veličine 256MB. PMT (*page map table*) je organizovana u dva nivoa, s tim da su i broj ulaza, kao i širina ulaza u PMT prvog i drugog nivoa isti (PMT oba nivoa su iste veličine).

Kada sistem kreira nov proces, ne učitava inicijalno nijednu njegovu stranicu, niti alocira ijednu PMT drugog nivoa, već samo alocira PMT prvog nivoa, čije sve ulaze inicijalizuje nulama. Stranice se potom dohvataju na zahtev, tokom izvršavanja procesa, kada se po potrebi alociraju i PMT drugog nivoa. Prilikom incijalizacije procesa, sistem samo kreira memorijski kontekst kao skup deskriptora logičkih memorijskih segmenata definisanih u izvršnom fajlu. Deskriptor segmenta je predstavljen strukturom `SegDesc` u kojoj, između ostalog, postoje sledeća polja:

- `unsigned long startingPage`: početna stranica segmenta u virtuelnom prostoru;
- `unsigned long size`: veličina segmenta izražena u broju stranica;
- `SegDesc *next, *prev`: sledeći i prethodni deskriptor segmenta u dvostruko ulančanoj listi segmenata istog procesa; na prvi segment u listi ukazuje polje `segDesc` u PCB procesa.

Kada stranica nije u memoriji ili ne pripada alociranom segmentu, u odgovarajućem ulazu u PMT nalazi se vrednost 0 koja hardveru indikuje da preslikavanje nije moguće; procesor tada generiše izuzetak tipa stranične greške (*page fault*).

a)(3) Prikazati logičku strukturu virtuelne i fizičke adrese i označiti veličinu svakog polja.

Odgovor:

b)(7) Implementirati sledeću funkciju:

```
int resolvePageFault (PCB* pcb, unsigned long addr, SegDesc** ret);
```

Ovu funkciju poziva kod kernela kada obrađuje izuzetak tipa stranične greške za proces na čiji PCB ukazuje prvi argument, prilikom adresiranja adrese date drugim argumentom. Ova funkcija treba da proveri da li je ovo adresiranje dozvoljeno ili nije; ukoliko nije, treba da vrati rezultat `MEM_ACCESS_FAULT`, na osnovu koga će kernel ugasiti proces; u suprotnom, radi se o slučaju kada je adresiranje dozvoljeno, ali stranica nije u memoriji i treba je učitati, pa treba vratiti rezultat `LOAD_PAGE`, a u izlazni argument na koga ukazuje argument `ret` upisati pokazivač na deskriptor segmenta kom pripada data adresa.

Rešenje: