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, april 2013.
*Datum:* 27.4.2013.

# *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 | *Zadatak 4* | _____/10 |

**Ukupno:** _____/40 = _____% = _____/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)

U nastavku je dat kod uporednih procesa koji simuliraju policijski helikopter i automobil u poteri za beguncem, objašnjen na predavanjima. Korišćenjem standardnih brojačkih semafora obezbediti potrebnu sinhronizaciju koja eliminiše problem utrkivanja (*race condition*) koji postoji u ovom kodu, objašnjen na predavanjima, ali tako da se tokom izvršavanja potprograma `moveTo` ne sprečava napredovanje procesa `Helicopter`.

```
type Coord = record {
  x : integer;
  y : integer;
};

var sharedCoord : Coord;

process Helicopter
var nextCoord : Coord;
begin
  loop
    computeNextCoord(nextCoord);
    sharedCoord := nextCoord;
  end;
end;

process PoliceCar
begin
  loop
    moveTo(sharedCoord);
  end;
end;
```

Rešenje:

## 2. (10 poena)

Data je sledeća implementacija operacije `Semaphore::lock()` koja obezbeđuje zaključavanje semafora (međusobno isključenje operacija na semaforu) u kodu školskog jezgra za višeprocesorski sistem. Operacija `swap()` implementirana je pomoću odgovarajuće atomične instrukcije procesora. Objasniti zašto ova implementacija operacije `lock()` nije dobra, a onda je popraviti.

```
extern void swap (int*, int*);

void Semaphore::lock() {
  int zero = 0;
  mask_interrupts();
  while (!this->lck) {}
  swap(&zero,&(this->lck));
}
```

Rešenje:

**3. (10 poena)**

Neki sistem primenjuje kontinualnu alokaciju memorije za procese. U strukturi PCB procesa postoje sledeća polja:

- `Word* mem_base`: pokazivač na lokaciju u memoriji na kojoj je proces smešten; tip `Word` predstavlja jednu adresibilnu jedinicu memorije;
- `size_t mem_size`: veličina memorijskog segmenta koju zauzima proces; `size_t` je celobrojni tip za izražavanje veličina memorijskih segmenata, u jedinicama `Word`.

Sistemskim pozivom `mem_extend` proces može da traži povećanje svog memorijskog prostora za traženu veličinu (proširenje iza kraja). Sistem će ispuniti ovaj zahtev ukoliko iza prostora koji proces već zauzima postoji slobodan segment dovoljne veličine. U suprotnom, odbiće taj zahtev (ne radi relokaciju procesa).

Slobodni segmenti ulančani su u listu, a funkcija

```
int mem_alloc(Word* address, size_t by);
```

pokušava da pronađe slobodan segment na adresi datoj prvim argumentom i da iz njega alocira deo veličine date drugim argumentom. U slučaju uspeha, funkcija vraća 0. Ukoliko na datoj adresi ne počinje slobodan segment ili on nije dovoljne veličine, ova funkcija vraća negativnu vrednost (kod greške).

Korišćenjem već implementirane funkcije `mem_alloc`, implementirati funkciju

```
int mem_extend (PCB* p, size_t by);
```

koja se koristi u implementaciji opisanog sistemskog poziva `mem_extend`. Ova funkcija treba da vrati 0 u slučaju uspeha, a negativnu vrednost u slučaju greške.

Odgovor:

**4. (10 poena)**

Virtuelni adresni prostor nekog sistema je 16EB (eksabajta, $1E=2^{60}$) i organizovan je stranično, adresibilna jedinica je bajt, a stranica je veličine 1KB. Fizički adresni prostor je veličine 1TB (terabajt). PMT (*page map table*) je organizovana u tri nivoa, s tim da su i broj ulaza, kao i širina ulaza u PMT sva tri nivoa isti (PMT sva tri nivoa su iste veličine). PMT sva tri nivoa smeštaju se u memoriju uvek poravnate na fizički okvir, odnosno uvek počinju na početku okvira. Zbog toga se u jednom ulazu u PMT prvog/drugog nivoa čuva samo broj okvira u kom počinje PMT drugog/trećeg nivoa, dok se preostali biti do celog broja bajtova u ulazu ne koriste; vrednost 0 u svim bitima označava da preslikavanje nije dozvoljeno. U jednom ulazu PMT trećeg nivoa čuva se broj okvira u koji se stranica preslikava i još 2 bita koja koduju prava pristupa (00 – nedozvoljen pristup, stranica nije u memoriji, 01 – dozvoljeno samo izvršavanje instrukcije, 10 – dozvoljeno samo čitanje podataka, 11 – dozvoljeno i čitanje i upis podataka); jedan ulaz u PMT trećeg nivoa zauzima minimalan, ali ceo broj bajtova.

Kada sistem kreira nov proces, ne učitava inicijalno ni jednu njegovu stranicu, niti alocira ijednu PMT drugog i trećeg 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 i trećeg nivoa.

Odgovoriti na sledeća pitanja uz detaljna obrazloženja postupka.

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

Odgovor:

b)(3) Koliko memorije minimalno zauzima PMT alocirana za proces prilikom njegovog kreiranja?

Odgovor:

c)(4) Koliko ukupno memorije zauzimaju sve PMT alocirane za proces koji je u dosadašnjem toku izvršavanja adresirao najnižih 1GB svog virtuelnog adresnog prostora?

Odgovor:

Izrada: