Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

*Predmet:* Operativni sistemi 1 (IR2OS1)
*Nastavnik:* prof. dr Dragan Milićev
*Odsek:* Računarska tehnika i informatika
*Kolokvijum:* Prvi, april 2014.
*Datum:* 27.4.2014.

# *Prvi 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)

Date su deklaracije pokazivača preko kojih se može pristupiti registrima dva DMA kontrolera:

```
typedef unsigned int REG;
REG* dma1Ctrl =...;     // DMA1 control register
REG* dma1Status =...;   // DMA1 status register
REG* dma1Address =...;  // DMA1 block address register
REG* dma1Count =...;    // DMA1 block size register
REG* dma2Ctrl =...;     // DMA2 control register
REG* dma2Status =...;   // DMA2 status register
REG* dma2Address =...;  // DMA2 block address register
REG* dma2Count =...;    // DMA2 block size register
```

U upravljačkom registru najniži bit je bit *Start* kojim se pokreće prenos jednog bloka preko DMA, a u statusnom registru najniži bit je bit završetka prenosa (*TransferComplete*), a bit do njega bit greške (*Error*). Svi registri su veličine jedne mašinske reči (tip `unsigned int`). Kada DMA konotroler završi zadati prenos, on se automatski zaustavlja (nije ga potrebno zaustavljati upisom u upravljački registar). Završetak prenosa sa bilo kog DMA kontrolera generiše isti zahtev za prekid procesoru (signali završetka operacije sa dva DMA kontrolera vezani su na ulazni zahtev za prekid preko OR logičkog kola).

Zahtevi za ulaznim operacijama na nekom uređaju sa kog se prenos blokova vrši preko bilo kog od ova DMA kontrolera vezani su u jednostruko ulančanu listu. Zahtev ima sledeću strukturu:

```
struct IORequest {
  REG* buffer; // Data buffer (block)
  unsigned int size; // Buffer (blok) size
  int status; // Status of operation
  IORequest* next; // Next in the list
};
```

Na prvi zahtev u listi pokazuje globalni pokazivač `ioHead`. Kada kernel u listu stavi novi zahtev, pozvaće operaciju `transfer()` koja treba da pokrene prenos za taj zahtev na bilo kom trenutno slobodnom DMA kontroleru (u slučaju da su oba kontrolera zauzeta ne treba ništa uraditi). Zahtev koji se dodeli nekom od DMA kontrolera na obradu izbacuje se iz liste. Kada se završi prenos zadat jednim zahtevom na jednom DMA kontroleru, potrebno je u polje `status` date strukture preneti status završene operacije (0 – ispravno završeno do kraja, -1 – greška) i pokrenuti prenos za sledeći zapis u listi na tom DMA kontroleru, i potom izbaciti zahtev iz liste. Obratiti pažnju na to da oba DMA kontrolera mogu završiti prenos i generisati prekid istovremeno. Ako zahteva u listi više nema, ne treba uraditi više ništa (kada bude stavljao novi zahtev u listu, kernel će proveriti i videti da je ona bila prazna, pa pozvati ponovo operaciju `transfer()` itd.)

Potrebno je napisati kod operacije `transfer()`, zajedno sa odgovarajućom prekidnom rutinom `dmaInterrupt()` za prekid od DMA kontrolera.

```
void transfer ();
interrupt void dmaInterrupt ();
```

Rešenje:

**2. (10 poena)**

Neki računar podržava segmentnu organizaciju virtuelne memorije, pri čemu je virtuelna adresa 20-bitna, fizički adresni prostor je veličine 16MB, a adresibilna jedinica je bajt. Maksimalna veličina segmenta je 4KB. U sledećoj tabeli dat je sadržaj nekoliko ulaza u tabeli preslikavanja nekog procesa (sve vrednosti su heksadecimnalne):

| *Segment #* | *Size* | *In memory?* | *Starting physical address* |
|---|---|---|---|
| 70 | 500 | Yes | 250000 |
| 12 | 700 | Yes | AFF000 |
| A0 | 700 | No | - |
| B0 | 400 | No | - |
| C0 | 600 | Yes | D67000 |

Popuniti sledeću tabelu na sledeći način: ako data virtuelna adresa uzrokuje grešku prekoračenja - napisati X, ukoliko izaziva straničnu grešku (*page fault*) - napisati P, a ukoliko je preslikavanje uspešno, upisati fizičku adresu u koju se preslikava (heksadecimalno).

| *Virtual address (hex)* | *Mapping result (hex)* |
|---|---|
| 12FA0 | |
| C00F0 | |
| 70750 | |
| B0140 | |
| C02AB | |

Rešenje:

3. **(10 poena)**

Na asembleru datog procesora napisati kod operacije

```
void yield (PCB* cur, PCB* nxt);
```

poput one date na predavanjima, a koja u PCB na koji ukazuje prvi argument čuva kontekst izvršavanja koje se napušta i restaurira kontekst koji je sačuvan u PCB na koga ukazuje drugi argument.

Procesor je RISC, sa *load/store* arhitekturom i ima sledeće registre dostupne korisničkim procesima: 32 registra opšte namene (R0..R31), SP, PSW i PC. Za podršku efikasnoj promeni konteksta, ovaj procesor ima sledeće:

- registar `base` kome se može pristupati samo u privilegovanom režimu rada procesora (nije dostupan korisničkim procesima i nije deo konteksta procesa); sve instrukcije i načini adresiranja koriste ovaj registar kao i svaki drugi registar opšte namene;
- instrukciju `saveregs` koja prepisuje sadržaj redom svih registara dostupnih korisničkim procesima, osim PC, u memoriju počev od adrese na koju ukazuje sadržaj registra `base`; ova instrukcija je dostupna samo u privilegovanom režimu rada procesora;
- instrukciju `loadregs` koja učitava redom sve registre dostupne korisničkim procesima, osim PC, iz memorije počev od adrese na koju ukazuje sadržaj registra `base`; ova instrukcija je dostupna samo u privilegovanom režimu rada procesora.

U strukturi PCB postoji polje za čuvanje svih registara dostupnih procesima; pomeraj ovog polja u odnosu na početak strukture PCB označen je simboličkom konstantom `offsContext`.

Rešenje:

## 4. (10 poena)

U nekom sistemu svi sistemski pozivi vrše se softverskim prekidom broj 44h, pri čemu u registru `r0` sistem očekuje identifikator sistemskog poziva. U ovom sistemu postoji sistemski poziv za kreiranje i pokretanje niti nad potprogramom koji prihvata jedan argument tipa pokazivača. Ovaj poziv u registrima očekuje sledeće argumente:

- u `r0` treba da bude identifikator ovog poziva, 0 u ovom slučaju;
- u `r1` treba da bude adresa potprograma nad kojim se kreira nit;
- u `r2` treba da bude argument potprograma nad kojim se kreira nit.

Svaki sistemski poziv vraća rezultat u registru `r0`, a za ovaj sistemski poziv rezultat je identifikator niti u jezgru (ceo broj veći od 0), odnosno kod greške (ceo broj manji od 0).
U asemblerskim blokovima unutar C/C++ koda može se koristiti simbolička konstanta sa nazivom formalnog argumenta funkcije unutar koje se nalazi dati asemblerski kod; ova simbolička konstanta ima vrednost odgovarajućeg pomeraja lokacije u kojoj se nalazi dati formalni argument u odnosu na vrh steka (kao što je pokazano u primerima na predavanjima). C/C++ funkcije vraćaju vrednost u registru `r0` ukoliko je tip povratne vrednosti odgovarajući.

a)(5) Implementirati funkciju

```
int create_thread (void (*f)(void*), void* arg);
```

koja vrši opisani sistemski poziv (kreira nit nad funkcijom `f` sa argumentom `arg`) i koja može da se poziva iz C koda sa zadatim argumentima. Ova funkcija vraća identifikator kreirane niti koji se može koristiti u ostalim sistemskim pozivima da identifikuje tu nit u jezgru, odnosno kod greške.

b)(5) Korišćenjem prethodne funkcije, implementirati funkciju `start` klase `Thread`. Ova klasa obezbeđuje koncept niti u objektnom duhu, poput onog u školskom jezgru (kreira nit nad virtuelnom funkcijom `run`).

```
class Thread {
public:
  Thread () : pid(0) {}
  int start ();
  virtual void run () {}
private:
  int pid;  // Thread ID
};
```

Rešenje: