Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

*Predmet:* Operativni sistemi 1 (SI2OS1)
*Nastavnik:* prof. dr Dragan Milićev
*Odsek:* Softversko inženjerstvo
*Kolokvijum:* Prvi, mart 2016.
*Datum:* 26.3.2016.

# *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

**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)**

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* dma2Ctrl =...;     // DMA2 control register
REG* dma2Status =...;   // DMA2 status register
```

Inicijalno, svaki DMA kontroler je u *pasivnom* stanju. Prvi upis u upravljački registar u pasivnom stanju tumači se kao upis adrese bloka podataka za prenos. Naredni upis u isti registar tumači se kao upis veličine bloka podataka za prenos. Naredni upis nenulte vrednosti u ovaj registar tumači se kao pokretanje prenosa, čime se kontroler prebacuje u *aktivno* stanje. Kada završi prenos, bilo ispravno ili u slučaju greške, DMA kontroler zaustavlja prenos, generiše prekid procesoru i prebacuje se ponovo u pasivno stanje. U statusnom registru najniži bit je bit završetka transfera (*TransferComplete*), a bit do njega bit greške (*Error*). Završetak prenosa sa bilo kog DMA kontrolera postavlja njegov bit *TransferComplete* i 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 globali pokazivač `ioHead`, a na poslednji `ioTail`. Operacijom `transfer()` kernel stavlja jedan zahtev u ovu listu i po potrebi pokreće transfer na bilo kom trenutno slobodnom DMA kontroleru. 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, a zahtev izbaciti iz liste. Obratiti pažnju na to da oba DMA kontrolera mogu završiti prenos i generisati prekid istovremeno.

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

```
void transfer (IORequest* request);
interrupt void dmaInterrupt ();
```

Rešenje:

**2. (10 poena)**

Neki procesor obrađuje prekide (hardverske i softverske) u sistemskom, privilegovanom režimu rada procesora, u kome se izvršava kod kernela (čiji je deo i prekidna rutina), koristeći posebne registre SPC (*System Program Counter*), SPSW (*System Processor Status Word*) i SSP (*System Stack Pointer*) koji nisu dostupni u korisničkom (neprivilegovanom) režimu. Registre PC, PSW i SP, kao ni sve druge programski dostupne registre koji se koriste u neprivilegovanom režimu, procesor ne čuva nigde implicitno prilikom obrade prekida, jer se oni ni ne menjaju implicitno tokom skoka u prekidnu rutinu niti tokom izvršavanja te rutine (izvršavanje instrukcija u privilegovanom režimu koristi SPC, SPSW i SSP umesto PC, PSW i SP); njihove vrednosti se mogu menjati eksplicitno, uobičajenim instrukcijama u privilegovanom režimu (npr. `load` i `store`). Prema tome, izvršavanje u privilegovanom režimu koristi i poseban stek alociran u delu memorije koju koristi kernel. Taj stek je jedan i kernel ga ne menja.

Prilikom povratka iz prekidne rutine instrukcijom `iret` procesor ništa ne restaurira, samo se prebacuje na korišćenje registara PC, SP i PSW umesto njihovih sistemskih parnjaka SPC, SSP i SPSW.

Procesor je RISC, sa *load/store* arhitekturom i ima sledeće registre dostupne u neprivilegovanom režimu: 32 registra opšte namene (R0..R31), SP, PSW i PC. Poseban registar RX je dostupan samo u privilegovanom režimu rada procesora, pa ga korisnički procesi ne koriste i kernel ga može koristiti samo za svoje potrebe.

Registre dostupne u neprivilegovanom režimu treba sačuvati u odgovarajuća polja strukture PCB. U strukturi PCB postoje polja za čuvanje svih tih registara; pomeraji ovih polja u odnosu na početak strukture PCB označeni su simboličkim konstantama `offsPC`, `offsPSW`, `offsSP`, `offsR0`, ..., `offsR31`.

U kodu kernela postoji statički definisan pokazivač `running` koji ukazuje na PCB tekućeg procesa. Potprogram `scheduler`, koji nema argumente, realizuje raspoređivanje, tako što smešta PCB na koji ukazuje pokazivač `running` u listu spremnih procesa, a iz nje uzima jedan odabrani proces i postavlja pokazivač `running` na njega.

Na asembleru datog procesora napisati kod prekidne rutine `dispatch` koja vrši promenu konteksta korišćenjem potprograma `scheduler`. Ova prekidna rutina poziva se sistemskim pozivom iz korisničkog procesa, pri čemu se sistemski poziv realizuje softverskim prekidom.

Rešenje:

**3. (10 poena)**

U nastavku je data donekle izmenjena i pojednostavljena specifikacija dva sistemska poziva iz standardnog *POSIX thread* API (*Pthreads*). Obe ove funkcije vraćaju negativnu vrednost u slučaju greške, a 0 u slučaju uspeha.

- `int pthread_create(pthread_t *thread, void *(*start_routine)(void *), void *arg)`: kreira novu nit nad funkcijom na koju ukazuje `start_routine`, dostavljajući joj argument `arg`; identifikator novokreirane niti smešta u lokaciju na koju ukazuje `thread`;
- `int pthread_join(pthread_t thread, void **retval)`: čeka da se nit identifikovana sa `thread` završi (ukoliko se već završila, odmah vraća kontrolu); ako `retval` nije `NULL`, kopira povratnu vrednost funkcije `start_routine` koju je ta nit izvršavala u lokaciju na koju ukazuje `retval`.

Data je struktura `Node` koja predstavlja jedan čvor binarnog stabla. Korišćenjem datih sistemskih poziva implementirati funkciju `createSubtree` koja rekurzivno kreira binarno stablo dubine `n` i vraća pokazivač na njegov koreni čvor, tako što u istoj niti kreira levo podstablo, a u novokreiranoj niti uporedo kreira desno podstablo. Ignorisati eventualne greške.

```
struct Node {
  Node () : leftChild(NULL), rightChild(NULL) {}
  Node *leftChild, *rightChild;
  ...
};

Node* createSubtree (int n);
```

Rešenje: