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 2015.
Datum: 28.3.2015.

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 kontrolera jednog ulazno-izlaznog uređaja:

```
typedef unsigned int REG;
REG* ioCtrl =...;   // control register
REG* ioStatus =...; // status register
REG* ioData =...;   // data register
```

U upravljačkom registru najniži bit je bit *Start* kojim se pokreće prenos, a bit do njega definiše smer prenosa podataka (0-ulaz, 1-izlaz). U statusnom registru najniži bit je bit spremnosti (*Ready*), a bit do njega bit greške (*Error*). Svi registri su veličine jedne mašinske reči (tip `unsigned int`). U slučaju greške u prenosu, uređaj generiše isti prekid kao i u slučaju spremnosti za prenos novog podatka.

Zahtevi za ulaznim i izlaznim operacijama prenosa blokova podataka na ovom uređaju vezani su u jednostruko ulančanu listu. Zahtev ima sledeću strukturu:

```
struct IORequest {
  REG* buffer; // Data buffer (block)
  unsigned int size; // Buffer (blok) size
  int dir; // Transfer direction: 0-input, 1-output
  int status; // Status of operation
  IORequest* next; // Next in the list
};
```

Na prvi zahtev u listi pokazuje globali pokazivač `ioHead`. Kada u praznu listu kernel stavi prvi zahtev, pozvaće operaciju `transfer()` koja treba da pokrene prenos za taj prvi zahtev. Kada se završi prenos zadat jednim zahtevom, potrebno je u polje `status` date strukture preneti status završene operacije (0 – ispravno završeno do kraja, -1 – greška), izbaciti obrađeni zahtev iz liste i pokrenuti prenos za sledeći zapis u listi. 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 `io↑Interrupt()` za prekid od uređaja, pri čemu prenos treba vršiti tehnikom programiranog ulaza-izlaza uz korišćenje prekida.

```
void transfer ();
interrupt void ioInterrupt ();
```

Rešenje:

2. (10 poena)

Neki procesor obrađuje prekide (hardverske i softverske) tako što tokom izvršavanja prekidne rutine koristi poseban stek koji se koristi u sistemskom, privilegovanom režimu rada procesora, u kome se izvršava kod kernela (čiji je deo i prekidna rutina).[1] Taj stek alociran je u delu memorije koju koristi kernel, a na vrh tog steka ukazuje poseban registar procesora koji je dostupan samo u privilegovanom režimu. Taj stek je uvek isti i kernel ga ne menja.

Prilikom obrade prekida, procesor na ovom sistemskom steku čuva: pokazivač vrha korisničkog steka (SP) koji je koristio prekinuti program, programsku statusnu reč (PSW) i programski brojač (PC), tim redom, ali *ne* i ostale programski dostupne registre. Prilikom povratka iz prekidne rutine instrukcijom `iret`, procesor restaurira ove registre sa sistemskog steka i vraća se u korisnički režim, a time i na korisnički stek.

Procesor je RISC, sa *load/store* arhitekturom i ima sledeće programski dostupne registre: 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 PC, SP, PSW i R0..R31 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:

1 Na ovaj način rade mnogi procesori. Na primer, opis koji sledi je donekle izmenjen i pojednostavljen opis načina funkcionisanja savremenih Intel procesora sa arhitekturom IA-32 i IA-64.

3. (10 poena)

U nekom sistemu sistemski poziv `fork()` kreira nit – klon pozivajuće niti, sa iskopiranim celokupnim stekom, slično istoimenom sistemskom pozivu na sistemu Unix (osim što se ovde radi o nitima, a ne procesima).
Dole je dat program čija je zamisao da izvršava dve uporedne niti, jednu koja učitava znak po znak sa stadardnog ulaza i taj znak prenosi kroz promenljivu `c` drugoj niti, koja taj primljeni znak ispisuje na standardni izlaz, i tako neograničeno.

a)(5) Precizno objasniti problem koji ovaj program ima i ispraviti taj problem.

b)(5) Prepraviti samo funkciju `pipe()` tako da se umesto jednog para niti koje vrše razmenu znakova, formiraju dva para takvih niti; svaki par niti predstavlja odvojeni „tok“, pa je potrebno definisati dva para promenljivih `c` i `flag` (npr. `c1`, `c2`, `flag1` i `flag2`).

```
#include <iostream>

void writer (char* c, int* flag) {
  while (1) {
    while (*flag==1);
    cin>>(*c);
    *flag = 1;
  }
}

void reader (char* c, int* flag) {
  while (1) {
    while (*flag==0);
    cout<<(*c);
    *flag = 0;
  }
}

void pipe () {
  char c;
  int flag = 0;
  if (fork())
    writer(&c,&flag);
  else
    reader(&c,&flag);
}

void main () {
  pipe();
}
```

Rešenje: