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, maj 2015.
Datum: 9.5.2015.

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)

Na jedan isti ulazno-izlazni kontroler vezana su dva ulazno-izlazna uređaja. Kontroler može da obavlja uporedne prenose sa ova dva uređaja preko dva logička „kanala“. Date su deklaracije pokazivača preko kojih se može pristupiti registrima ovog kontrolera:

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

U (samo jednom) upravljačkom registru dva najniža bita su biti *Start* kojim se pokreće prenos na prvom, odnosno drugom kanalu, a dva naredna bita definišu smer prenosa podataka na ova dva kanala (0-ulaz, 1-izlaz). Upravljački registar vezan je tako da se može i čitati. U (samo jednom) statusnom registru dva najniža bita su biti spremnosti (*Ready*), a dva bita do njih su biti greške (*Error*). Svi registri su veličine jedne mašinske reči (tip `unsigned int`). Kada je uređaj na jednom kanalu obavio prenos jednog podatka i spreman je za sledeći, odgovarajući bit *Ready* se postavlja na 1, a kontroler generiše prekid (isti prekid za oba kanala). U slučaju greške u prenosu, uređaj generiše isti prekid kao i u slučaju završetka prenosa jednog podatka. Na dve susedne adrese počev od adrese `ioData` nalaze se dva registra za podatke, za dva kanala.

Zahtevi za ulaznim i izlaznim operacijama prenosa blokova podataka vezani su u jednostruko ulančanu listu. Zahtev se može opslužiti na bilo kom slobodnom kanalu. 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
};
```

Polja u ovoj strukturi mogu se koristiti kao promenljive tokom prenosa (ne mora se očuvati njihova početna vrednost nakon prenosa). Na prvi zahtev u listi pokazuje globali pokazivač `ioHead`. Kada u praznu listu kernel stavi prvi zahtev ili nekoliko zahteva odjednom, pozvaće operaciju `transfer()` koja treba da pokrene prenos za prvi zahtev (ili prva dva). 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) i pokrenuti prenos na kanalu koji je završio 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.). Zahtev koji je opslužen ne treba više da bude u listi (ali ne treba brisati samu strukturu zahteva).

Potrebno je napisati kod operacije `transfer()`, zajedno sa odgovarajućom prekidnom rutinom `ioInterrupt()` za prekid od kontrolera, 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.[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 rada procesora. Na tom steku samo se obrađuju prekidi, i uvek je isti (kernel ga ne menja).

Prilikom obrade prekida, procesor na ovom posebnom steku čuva: pokazivač vrha steka (SP) koji je koristio prekinuti tok kontrole, 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 ovog steka, a time prelazi na drugi stek.

Procesor ima dva režima rada: privilegovani (sistemski), u kome se izvršava kod kernela, i korisnički, u kome se izvršavaju korisnički procesi. Prilikom prihvatanja prekida, procesor obavezno prelazi u privilegovani režim rada. Prilikom povratka iz prekidne rutine, procesor *ne* menja implicitno režim rada, već ostaje u istom režimu rada u kome se nalazi prilikom obrade instrukcije `iret`. Ako je potrebno preći u korisnički režim rada, potrebno je izvršiti eksplicitnu instrukciju `setusr` koja prebacuje procesor u korisnički režim.

Operativni sistem za ovaj procesor je višenitni (engl. *multithreaded*). Sve funkcije jezgra, uključujući i obradu sistemskog poziva, raspoređivanje itd. obavljaju interne kernel niti. Svi sistemski pozivi izvršavaju se kao softverski prekid koji skače na prekidnu rutinu označenu kao `sys_call`, dok se sama identifikacija sistemskog poziva i njegovi parametri prenose kroz registre procesora.

Procesor je RISC, sa *load/store* arhitekturom i ima sledeće programski dostupne registre: 32 registra opšte namene (R0..R31), SP, PSW i PC. Registre PC, SP, PSW i R0..R31 treba sačuvati u odgovarajuća polja strukture PCB. U strukturi PCB, koja postoji i ista je i za korisničke procese i za kernel niti, 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 postoje dva statički definisana pokazivača, `userRunning` i `kernelRunning`, koji ukazuju na PCB tekućeg korisničkog procesa, odnosno kernel niti, respektivno.

a)(7) Na asembleru datog procesora napisati prekidnu rutinu `sys_call`. Ova rutina treba samo da izvrši promenu konteksta sa tekućeg korisničkog procesa na tekuću kernel nit.

b)(3) Na asembleru datog procesora napisati rekidnu rutinu `switch_to_user` (dovoljno je i precizno objasniti razlike u odnosu na rutinu `sys_call`) koja se poziva softverskim prekidom iz kernel koda, kada je kernel završio ceo posao i kada želi da promeni kontekst sa tekuće kernel niti na korisnički proces i vrati se u korisnički režim rada.

Rešenje:

1 Na ovaj način rade mnogi procesori. Na primer, ovaj opis 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 (neispravan) program čija je zamisao da izvršava dva para uporednih niti, pri čemu u svakom paru jedna nit učitava znak po znak sa standardnog ulaza i taj znak prenosi kroz promenljivu `c[0]/c[1]` drugoj niti u istom paru, koja taj primljeni znak ispisuje na standardni izlaz, i tako neograničeno.

a)(5) Precizno objasniti problem koji ovaj program ima (zašto je neispravan).

b)(5) Prepraviti samo funkciju `pipe()` tako da se umesto dva para niti koje vrše razmenu znakova, formira `N` parova takvih niti; svaki par niti `i` predstavlja odvojeni „tok“ koji treba da razmenjuje podatke preko `c[i]` i `flag[i]`.

```
#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;
  }
}

const int N = ...; // N>2
char c[N];
int flag[N];

void main () {
  for (int i=0; i<N; i++) flag[i]=0;
  pipe();
}

void pipe () {
  if (fork())
    writer(&c[0],&flag[0]);
  else
    reader(&c[0],&flag[0]);
  if (fork())
    writer(&c[1],&flag[1]);
  else
    reader(&c[1],&flag[1]);
}
```

Rešenje: