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 2017.
Datum: 25.3.2017.

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 izlaznog uređaja:

```
typedef unsigned int REG;
REG* ioCtrl =...;     // control register
REG* ioStatus =...;   // status register
REG* ioData =...;     // data register
const unsigned int BLOCK_SIZE = ...;
const REG START_SENDING = ..., END_SENDING = ...;
```

Na ovaj izlazni uređaj se jednom operacijom prenosi čitav blok podataka veličine `BLOCK_SIZE`. Da bi se uređaju zadao prenos jednog bloka podataka, potrebno je najpre u upravljački registar upisati vrednost predstavljenu simboličkom konstantom `START_SENDING`. Potom treba jednu po jednu reč upisivati na istu adresu registra za podatke (`ioData`). Kontroler uređaja poseduje memorijski modul koji služi za prihvatanje celog bloka podataka (bafer), tako da se reči upisane redom na adresu registra za podatke upisuju redom u ovaj bafer (ovo obezbeđuje interni hardver kontrolera: svaki naredni upis na ovu adresu inkrementira interni adresni registar za adresiranje unutar bafera; upis `START_SENDING` u upravljački registar zapravo resetuje ovaj adresni registar). Zbog toga se sukcesivne reči mogu upisivati na ovu adresu bez ikakvog čekanja (bez sinhronizacije), u sukcesivnim ciklusima upisa. Kada se završi upis celog bloka podataka, potrebno je u upravljački registar upisati vrednost predstavljenu simboličkom konstantom `END_SENDING`. Ovim se kontroleru nalaže da započne izlaznu operaciju zadatog bloka podataka.

Kada završi izlaznu operaciju prenosa celog bloka podataka, kontroler periferije postavlja bit u razredu 0 statusnog registra, ali *ne* generiše prekid procesoru. U statusnom registru bit 1 je tada bit greške u prenosu (*Error*).

Zahtevi za izlaznim operacijama na ovom uređaju vezani su u jednostruko ulančanu listu. Zahtev ima sledeću strukturu:

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

Na prvi zahtev u listi pokazuje globalni pokazivač `ioHead`, a na poslednji `ioTail`. Operacijom `transfer()` kernel stavlja jedan zahtev u ovu listu i po potrebi pokreće transfer na uređaju. 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 za sledeći zapis u listi, a zahtev izbaciti iz liste.

Napisati kod operacije `void transfer (IORequest* request)`.

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 SSP procesora koji je dostupan samo u privilegovanom režimu.

Prilikom obrade prekida, procesor na ovom steku čuva: pokazivač vrha korisničkog steka (SP) koji je koristio prekinuti proces, 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.

Svaki proces ima takav sopstveni sistemski stek. Prilikom promene konteksta, ostale programski dostupne registre treba sačuvati na ovom sistemskom steku prekinutog procesa. U strukturi PCB postoji polje za čuvanje vrednosti SSP steka procesa; pomeraj ovog polja u odnosu na početak strukture PCB označen je simboličkom konstantom `offsSSP`.

Procesor je RISC, sa *load/store* arhitekturom i ima sledeće programski dostupne registre: 32 registra opšte namene (R0..R31), SP, PSW i PC.

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 operativnom sistemu postoje sledeći sistemski pozivi:

- `int thread_create(void(*)(void*),void*)`: kreira nit nad funkcijom na koju ukazuje prvi argument; ta funkcija prima jedan argument tipa `void*` i ne vraća rezultat; novokreirana nit poziva tu funkciju sa stvarnim argumentom jednakim drugom argumentu ovog sistemskog poziva; sistemski poziv vraća PID kreirane niti;
- `void wait(int pid)`: suspenduje pozivajuću roditeljsku nit dok se ne završi nit-dete sa zadatim PID; ako je vrednost argumenta 0, pozivajuća nit se suspenduje dok se ne završe sve niti-deca;
- `char getc(FILE* stream)`: iz fajla na čiji deskriptor ukazuje prvi argument učitava i vraća jedan znak; svaki poziv učitava naredni znak iz fajla; ukoliko je učitavanje stiglo do kraja fajla, funkcija vraća vrednost `EOF`.

U nekom programu već je otvoreno *N* ulaznih fajlova na čije deskriptore ukazuju elementi niza `streams`:

```
const int N = ..., M = ...;
FILE* streams [N];
char text [N][M];
```

Na jeziku C napisati deo programa koji u *N* uporednih niti-dece učitava najviše po *M* znakova u svaki red tabele `text`; svaka nit učitava po jedan red tabele iz različitog fajla, u *i*-ti red iz *i*-tog fajla. Kada se učitavanje u potpunosti završi, program treba da ispiše sve redove tabele `text` na standardni izlaz.

Rešenje: