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 2019.
Datum: 23. 3. 2019.

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 ulaznog uređaja:

```
typedef unsigned long REG;
REG* ioCtrl =...;     // control register
REG* ioStatus =...;   // status register
REG* ioData =...;     // data register
REG* ioBlock =...;    // block address register
const unsigned int BLOCK_SIZE = ...;
const REG START_READING = ...;
```

Sa ovog ulaznog uređaja se jednom operacijom prenosi čitav blok podataka veličine `BLOCK_SIZE`. Adresa bloka na uređaju koji treba pročitati zadaje se upisom u registar na adresi `ioBlock`. Da bi se uređaju zahtevao prenos jednog bloka podataka, potrebno je najpre zadati adresu (broj) traženog bloka na uređaju, a potom u upravljački registar upisati vrednost predstavljenu simboličkom konstantom `START_READING`. Kontroler uređaja poseduje interni memorijski modul koji služi za prihvatanje celog pročitanog bloka podataka (bafer). Kada završi dohvatanje traženog bloka u ovaj svoj interni bafer, kontroler uređaja postavlja bit u razredu 0 svog statusnog registra na 1; ovaj signal *ne* generiše prekid procesoru. U slučaju greške, bit 1 postavlja takođe na 1. Kada je pročitan blok spreman u baferu, može se pročitati jedna po jedna reč tog bafera sukcesivnim čitanjem istog registra za podatke na adresi `ioData` (ovo obezbeđuje interni hardver kontrolera: svako naredno čitanje sa ove adrese inkrementira interni adresni registar za adresiranje unutar bafera; upis `START_READING` u upravljački registar zapravo resetuje ovaj adresni registar). Zbog toga se sukcesivne reči mogu čitati sa ove adrese bez ikakvog čekanja (bez sinhronizacije), u sukcesivnim ciklusima čitanja.

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

```
struct IORequest {
  REG* buffer; // Data buffer (data block) in memory
  REG block;   // The block number (address)
  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.

Ovaj isti sistemski stek se koristi tokom izvršavanja bilo kog koda kernela; kernel ima samo jedan takav stek (nema više niti). Prilikom promene konteksta, programski dostupne registre treba sačuvati u odgovarajućim poljima strukture PCB, u kojoj postoji polje za čuvanje svakog od programski dostupnih registara; pomeraj ovog polja u odnosu na početak strukture PCB označen je simboličkim konstantama `offsPC`, `offsSP`, `offsPSW`, `offsR0`, `offsR1` itd.

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:

- `fork(), execlp(const char*)`: kao u sistemu Unix i njemu sličnim;
- `int wait(int pid, unsigned timeout)`: suspenduje pozivajući roditeljski proces dok se ne završi proces-dete sa zadatim PID, ali ga suspenduje najduže onoliko koliko je zadato drugim argumentom (vreme čekanja u milisekundama). Ako je vrednost prvog argumenta NULL, pozivajući proces se suspenduje dok se ne završe sva njegova deca (ili ne istekne vreme čekanja); ako je vrednost drugog argumenta 0, poziv odmah vraća rezultat, bez čekanja. Vraćena vrednost 0 označava da su svi procesi koji su se čekali završili (pre isteka vremena čekanja); vraćena vrednost veća od 0 znači da je vreme čekanja isteklo pre nego što je neki od procesa koji su se čekali završili.
- `int kill(int pid)`: gasi proces sa zadaitm PID.

Svi ovi sistemski pozivi vraćaju negativan kod greške u slučaju neuspeha.

Na jeziku C napisati program koji se poziva sa jednim argumentom koji predstavlja stazu do *exe* fajla. Ovaj program treba da kreira *N* procesa koji izvršavaju program u *exe* fajlu zadatom argumentom (*N* je konstanta definisana u programu), a potom da čeka da se svi ti procesi-deca završe u roku od 5 sekundi. Sve procese-decu koji se nisu završili u tom roku treba da ugasi. Obraditi sve greške u sistemskim pozivima ispisom odgovarajuće poruke.