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, april 2019.
Datum: 5. 5. 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)

Na jedan isti kontroler vezana su dva ulazna 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 volatile 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. 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 *ne* generiše prekid. U slučaju greške u prenosu, uređaj postavlja *Ready* bit 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 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 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 globalni 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 dealocirati/brisati samu strukturu zahteva).

Potrebno je napisati kod operacije `transfer()`, pri čemu prenos treba vršiti tehnikom programiranog ulaza-izlaza prozivanjem.

```
void transfer ();
```

Rešenje:

2. (10 poena)

Neki sitem čuva kontekst niti u strukturi PCB u kojoj postoje polja za čuvanje vrednosti svih programski dostupnih registara procesora: pokazivača na vrh steka niti (SP), programske statusne reči (PSW) i programskog brojača (PC), kao i registara opšte namene R0..R31. Adrese i svi registri su 32-bitni, pa su takvi i tipovi `int` i svi pokazivački tipovi na jeziku C.

Dole je dat segment asemblerskog koda kojim ovaj sistem restaurira kontekst niti nakon svakog sistemskog poziva. Simboličke konstante `pc`, `psw`, `sp` itd. predstavljaju pomeraje (engl. *offset*) istoimenih polja u strukturi PCB. Registar R0 prenosi vrednost koja se vraća iz sistemskog poziva, pa se zato ne restaurira.

Potrebno je dopuniti implementaciju operacije `createThread` kodom na mestu označenom sa `/***/`. Ovu funkciju poziva kernel u implementaciji sistemskog poziva za kreiranje nove niti, a kod koji nedostaje treba da formira inicijalni procesorski kontekst. Nit treba kreirati nad korisničkom funkcijom na koju ukazuje argument `pf`, tako da ta nit izvršava tu funkciju sa argumentom datim u `arg`. Nakon izvršavanja ove korisničke funkcije `pf`, nit treba da se ugasi sistemskim pozivom implementiranim u funkciji `exit`.

Funkcija `createPCB` alocira novu strukturu PCB i adresu te strukture upisuje u pokazivač na koga ukazuje argument. Funkcija `allocateStack` alocira stek za novu nit i adresu najviše slobodne lokacije steka upisuje u pokazivač na koga ukazuje argument. Stek raste ka nižim adresama, a pokazivač vrha steka ukazuje na prvu slobodnu lokaciju.

```
                ; Restore the new context
                load  r1, [running]
                load r2, #sp[r1] ; restore SP
                push r2
                load r2, #psw[r1] ; restore PSW
                push r2
                load r2, #pc[r1] ; restore PC
                push r2
                load r31, #r31[r1] ; restore R31
                ...                    ; restore r30-r2
                load r1, #r1[r1] ; restore R1
                ; R0 holds the return value from the system call
                ; Return from system call (interrupt)
                iret
```

```
void exit ();  // Terminate the calling thread
const int PSW_INIT = ...; // Initial value for PSW

int createThread (void (*pf) (void*), void* arg) {
  PCB* pcb;
  int stat = createPCB(&pcb);
  if (stat!=0) return stat;
  void** sp;
  stat = allocateStack(&sp);
  if (stat!=0) return stat;
  /***/
  pcb->psw = PSW_INIT;
  stat = Scheduler::put(pcb);
  return stat;
}
```

Rešenje:

3. (10 poena)

U nekom operativnom sistemu postoje sledeći sistemski pozivi:

- `int fork()`: kao u sistemu Unix i njemu sličnim, samo što ne kreira proces, nego nit; u kontekstu niti-roditelja vraća identifikator niti-deteta (PID), a u kontekstu niti-deteta vraća 0;
- `exit()`: završava pozivajuću nit;
- `wait(int pid)`: suspenduje pozivajuću roditeljsku nit dok se ne završi nit-dete sa zadatim PID.

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

Data je globalna matrica `mat` celih brojeva (`int`), dimenzija $M \times N$ (M i N su konstante). Potrebno je napisati potprogram `par_sum` koji treba da izračuna ukupan zbir svih elemenata matrice `mat`, ali na sledeći način: najpre treba da izračuna zbir svake, i-te vrste ove matrice, uporedo, u M uporednih niti (zbir jedne vrste izračunava jedna nit), i da taj zbir smesti u i-ti element jednog pomoćnog niza; potom treba da sabere ovako izračunate zbirove vrsta i vrati rezultat. Ignorisati eventualne greške i prekoračenje.

```
const int M = ..., N = ...;
int mat[M][N];
int par_sum ();
```

Rešenje: