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 2018.
Datum: 29. 4. 2018.

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 četiri uređaja; prva dva uređaja su ulazni, a druga dva su izlazni:

```
typedef unsigned int REG;
REG* io1Ctrl =...;   // Device 1 control register
REG* io1Status =...; // Device 1 status register
REG* io1Data =...;   // Device 1 data register
REG* io2Ctrl =...;   // Device 2 control register
REG* io2Status =...; // Device 2 status register
REG* io2Data =...;   // Device 2 data register
REG* io3Ctrl =...;   // Device 3 control register
REG* io3Status =...; // Device 3 status register
REG* io3Data =...;   // Device 3 data register
REG* io4Ctrl =...;   // Device 4 control register
REG* io4Status =...; // Device 4 status register
REG* io4Data =...;   // Device 4 data register
```

U upravljačkim registrima najniži bit je bit *Start* kojim se pokreće uređaj, a u statusnim registrima najniži bit je bit spremnosti (*Ready*). Svi registri su veličine jedne mašinske reči (tip `unsigned int`). Samo uređaj broj 2 generiše prekid kada je spreman.

Potrebno je napisati proceduru `transfer` i odgovarajuću prekidnu rutinu za prekid sa uređaja broj 2 koja učitava po jednu reč sa uređaja 1 i tu reč odmah prebacuje na uređaj 3, odnosno učitava po jednu reč sa uređaja 2 i tu reč odmah prebacuje na uređaj 4. Ova dva prenosa se obavljaju uporedo, sve dok se sa bilo kog ulaznog uređaja ne učita vrednost 0. Prenose sa uređajima 1, 3 i 4 vršiti programiranim ulazom/izlazom sa prozivanjem (*polling*), a sa uređajem 2 korišćenjem prekida. Smatrati da je periferijski uređaj 4 veoma brz, tj. da vrlo brzo (reda vremena izvršavanja nekoliko desetina instrukcija procesora) od prijema podatka biva spreman za prijem novog podatka.

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 programski brojač (PC) i programsku statusnu reč (PSW), 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.

Postoji samo jedan kernel stek koji se koristi za izvršavanje celog koda kernela. Svi potprogrami kernela pisani su tako da na ovom steku čuvaju (i sa njega potom restauriraju) sve registre procesora koje koriste. Za sve vreme izvršavanja kernel koda prekidi su maskirani (procesor ih implicitno maskira prilikom prihvatanja prekida, a kernel kod ih ne demaskira).

U strukturi PCB postoje polja za čuvanje vrednosti svih programski dostupnih registara procesora; pomeraj polja za neki registar R*i* u odnosu na početak strukture PCB označen je simboličkom konstantom `offsRi` (npr. `offsSP`, `offsPC`, `offsPSW`, `offsR0`, ..., `offsR31`).

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

U kodu kernela postoji statički definisan pokazivač `running` koji ukazuje na PCB tekućeg procesa. Potprogram `sys_call_proc`, koji nema argumente, realizuje obradu sistemskog poziva, kao i raspoređivanje, tako što, nakon obrade samog sisteskog poziva, postavlja pokazivač `running` da ukazuje na PCB odabranog novog tekućeg procesa.

Na asembleru datog procesora napisati kod prekidne rutine `sys_call` za sistemske pozive softverskim prekidom, s tim da ona, zbog efikasnijeg rada, ne treba da vrši promenu konteksta (čuvanje i restauraciju registara) ako za tim nema potrebe, odnosno ukoliko procedura `sys_call_proc` nije promenila pokazivač `running`.

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 nastavku je data donekle izmenjena i pojednostavljena specifikacija sistemskog poziva iz standardnog *POSIX thread* API (*Pthreads*):

`int pthread_create(pthread_t *thread, void *(*routine)(void *), void *arg)`: kreira novu nit nad funkcijom na koju ukazuje `routine`, dostavljajući joj argument `arg`; identifikator novokreirane niti smešta u loakciju na koju ukazuje `thread`; vraća negativnu vrednost u slučaju greške, a 0 u slučaju uspeha.

Dat je potpis funkcije `fun`, pri čemu su `A`, `B`, `C` i `D` neki tipovi:

```
extern D fun (A a, B b, C c);
```

Potrebno je realizovati potprogram `fun_async` čiji će poziv izvršiti asinhroni poziv funkcije `fun`, tj. inicirati izvršavanje funkcije `fun` u posebnoj niti i potom odmah vratiti kontrolu pozivaocu. Po završetku funkcije `fun`, ta nit treba da pozove povratnu funkciju na koju ukazuje argument `cb` (engl. *callback*) i da joj dostavi vraćenu vrednost iz funkcije `fun`. Ignorisati greške.

```
typedef void (*CallbackD)(D);
void fun_async (A a, B b, C c, CallbackD cb);
```

Rešenje: