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 2016.
Datum: 8.5.2016.

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 generiše prekid procesoru. U statusnom registru najniži bit 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.

Potrebno je napisati kod operacije `transfer()`, zajedno sa odgovarajućom prekidnom rutinom `ioInterrupt()` za prekid od kontrolera uređaja.

```
void transfer (IORequest* request);
interrupt void ioInterrupt ();
```

Rešenje:

2. (10 poena)

U školskom jezgru promena konteksta implementirana je korišćenjem date funkcije `yield()`, u sistemskom pozivu `dispatch()` i na svim ostalim mestima na sličan način kao što je dato.

Potrebno je implementirati sistemski poziv (statičku operaciju) `Thread::wait()` kojim pozivajuća nit čeka (suspenduje se ako je potrebno) dok se ne završe sve niti-deca koje je ova pozivajuća nit do tada kreirala. Za te potrebe treba implementirati i sledeće nestatičke funkcije-članice:

- `void Thread::created(Thread* parent)`: poziva je jezgro interno za datu novokreiranu nit (`this`), kada je ta nit kreirana, sa argumentom `parent` koji ukazuje na roditeljsku nit u čijem kontekstu je ova nova nit-dete kreirana;
- `void Thread::completed()`: poziva je jezgro za datu nit (`this`), kada se ta nit završila.

Ukoliko proširujete klasu `Thread` novim članovima, precizno navedite kako.

```
void yield (jmp_buf old, jmp_buf new) {
  if (setjmp(old)==0) longjmp(new,1);
}

void Thread::dispatch () {
  lock();
  jmp_buf old = Thread::running->context;
  Scheduler::put(Thread::running);
  Thread::running = Scheduler::get();
  jmp_buf new = Thread::running->context;
  yield(old,new);
  unlock();
}
```

Rešenje:

3. (10 poena)

U sistemu UNIX i sličnim sistemima sistemski poziv

```
int execvp(const char * filename, const char * const args[]);
```

radi isto što i poziv `execlp`, s tim što drugi argument predstavlja niz pokazivača na nizove znakova (*null-terminated strings*) koji predstavljaju listu argumenata poziva programa koji treba izvršiti. Po konvenciji, prvi argument pokazuje na naziv fajla samog programa koji se izvršava. Ovaj niz pokazivača mora da se završi elementom sa vrednošću NULL. Ovaj poziv vraća 0 u slučaju uspeha, a vrednost manju od 0 u slučaju greške.

Korišćenjem ovog sistemskog poziva, kao i sistemskog poziva `fork()`, realizovati sledeću funkciju:

```
int multiexec (int number, const char* filename, const char* const args[]);
```

Ova funkcija treba da kreira zadati broj (`number`) procesa-dece, i svaki od tih procesa-dece treba da izvršava isti program zadat u fajlu sa imenom `filename`, sa samo po jednim argumentom. Argumenti tih procesa-dece zadati su redom u prvih `number` elemenata niza `args`. Ova funkcija treba da vrati broj uspešno kreiranih procesa-dece (bez obzira na to da li su ti procesi uspešno izvršili `execvp`) i da odmah vrati kontrolu pozivaocu, ne čekajući da se ti procesi završe.

Rešenje: