Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (SI3OS2, IR3OS2)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Treći, januar 2014.
Datum: 21.1.2014.

Treći kolokvijum iz Operativnih sistema 2

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 = _____%

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) Sistemski pozivi

U nekom operativnom sistemu sistemski poziv se vrši softverskim prekidom. Svakoj grupi srodnih sistemskih usluga odgovara jedan softverski prekid. Unutar date grupe, sistemsku uslugu određuje vrednost u registru `R1`. Svaki sistemski poziv u registru `R2` očekuje adresu strukture podataka u kojoj su parametri sistemskog poziva, zavisni od konkretne usluge. Status sistemskog poziva vraća se kroz registar `R0` (0-uspešno, <0 – kod greške).
Grupa usluga koje se pozivaju softverskim prekidom 11h vezane su za kreiranje procesa. Sistemski poziv broj 31h u toj grupi je zahtev za kreiranjem procesa nad zadatim programom. Svoje parametre ova usluga očekuje u sledećoj strukturi:

```
struct create_process_struct {
  char* program_file;    // Null-terminated string with program file path
  char** args;           // Program arguments (array of strings)
};
```

Pretpostaviti da je procesor 32-bitni, dvoadresni, RISC, sa *load-store* arhitekturom. Ima registarski fajl sa 32 registra (`R0-R31`). Svi registri i adrese su 32-bitni. U asemblerskom kodu unutar C koda datog kompajlera, može se upotrebljavati identifikator statičke promenljive koji se prevodi u memorijsko direktno adresiranje te promenljive. Povratne vrednosti funkcija se prenose kroz registar `R0`, ukoliko je veličina odgovarajuća.
Interpreter komandne linije (CLI) ovog opreativnog sistema realizovan je kao sistemski program koji učitava jednu liniju sa konzole, zatim pokreće proces nad komandom čiji je naziv dat u učitanoj komandnoj liniji, čeka da se taj proces završi i prelazi na učitavanje sledeće komandne linije. Skup sistemskih komandi ovog inerpretera implementiran je kao skup izvršnih programa koji se nalaze u unapred definisanom direktorijumu. Punu putanju do ovog direktorijuma sadrži globalna promenljiva `CLI_path`.
Na jeziku C/C++ napisati bibliotečnu funkciju `execute_command()` koja je deo API ovog operativnog sistema i koju poziva interpreter komandne linije:

```
extern char* CLI_path;
int execute_command(char* command_name, char** args);
```

Prvi argument je naziv komande, odnosno ime programa, a drugi argument predstavlja paremetre poziva te komande. U slučaju greške ova funkcija treba da prosledi kod greške. Na raspolaganju je funkcija koja vrši konkatenaciju dva niza karaktera:

```
char* strcat(const char* str1, const char* str2); // str1 and str2 are
null-terminated strings
```

Rešenje:

2. (10 poena) Operativni sistem Linux

Napisati *shell script* koji obrađuje fajl u kom se nalaze informacije kojim korisničkim grupama pripadaju korisnici. Fajl koji treba obraditi se zadaje kao prvi argument pri pozivu, korisničko ime korisnika za koga je potrebno ažurirati informacije se zadaje kao drugi argument pri pozivu i korisničke grupe se zadaju kao ostali argumenti pri pozivu. Ukoliko se skripta ne pozove sa dovoljnim brojem argumenata ispisati odgovarajuću poruku i prekinuti izvršavanje. Smatrati da fajl zadat kao prvi argument postoji i da je dozvoljen upis u njega. Fajl sadrži, u svakoj liniji posebno, informaciju o tome koji korisnik pripada kojim korisničkim grupama u sledećem formatu: na početku linije je korisničko ime, zatim slede korisničke grupe odvojene znakom razmaka. Primer jedne linije je:

```
pera users pera sys admin
```

Skripta treba da ažurira informacije za korisnika čije je korisničko ime zadato kao argument tako što će za korisnika dodati korisničke grupe koje su zadati kao argumenti. Ako korisnik ne postoji u fajlu, treba da se doda linija za njega. Za jednog korisnika u fajlu ne sme da se pojavi jedna korisnička grupa više od jedanput. Primer poziva skripta i rezultujuća linija za gore dati primer su:

```
./script.sh groups.txt pera users root audio
pera users pera sys admin root audio
```

Rešenje:

3. (10 poena) Operativni sistem Linux

Na jeziku C/C++, koristeći mehanizam prosleđivanja poruka operativnog sistema Linux, projektovati konkurentni klijent-server sistem. Klijenti su procesi koji ciklično obavljaju svoje aktivnosti. Pre nego što u jednom ciklusu neki klijent započne svoju aktivnost, dužan je da od servera traži dozvolu u obliku "žetona" (engl. *token*). Kada dobije žeton, klijent započinje aktivnost. Po završetku aktivnosti, klijent vraća žeton serveru. U nastavku je data funkcija `client` koja kao argument prima jedinstveni broj (identifikator) klijenta. Svaki klijent pri slanju zahteva identifikuje se ovim brojem. Navedena je i struktura poruka koje se razmenjuju, kao i značenje vrednosti svakog polja.

```
#define MESSAGE_Q_KEY 1

struct requestMsg {
 long mtype;  // tip poruke – identifikator klijenta
 char msg[1]; // operacija - vrednost: 1 - request token, 2 - release token
};

void client(int id) {
 int requestMsgQueueId = msgget(MESSAGE_Q_KEY, IPC_CREAT | 0666);
 int responseMsgQueueId = msgget(MESSAGE_Q_KEY + 1, IPC_CREAT | 0666);
 size_t len = sizeof(char);

 while (1) {
  //request token
  struct requestMsg msg;
  msg.mtype = id;
  msg.msg[0] = (char) 1;
  msgsnd(requestMsgQueueId, &msg, len, 0);
  msgrcv(responseMsgQueueId, &msg, len, id, 0);

  //execute activity
  sleep(1);

  //release token
  msg.mtype = id;
  msg.msg[0] = (char) 2;
  msgsnd(requestMsgQueueId, &msg, len, 0);

  //wait next round
  sleep(1);
 }
}
```

Napisati kod procesa-servera koji na početku nad funkcijom `client` kreira *N* procesa-klijenta, a zatim prima zahteve i vodi računa da u jednom trenutku ne može biti izdato više od *M* žetona: ukoliko klijent traži žeton, a ne može da ga dobije jer je već izdato *M* žetona, klijent se blokira. Parametri *M* i *N* zadaju se kao argumenti prilikom poziva programa. Nije potrebno proveravati uspešnost izvršavanja operacije nad sandučićima (*message queue*).

Rešenje: