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 2012.
Datum: 10.1.2012.

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 sotfverski 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 31h vezane su za upravljanje memorijom. Sistemski poziv broj 21h u toj grupi je zahtev za alokacijom dela virtuelnog prostora pozivajućeg procesa. Svoje parametre ova usluga očekuje u sledećoj strukturi:

```
struct vm_area_desc {
  int page; // VM area starting page
  int size; // VM area size in pages
};
```

Proceror je 32-bitni, dvoadresni, RISC, sa *load-store* arhitekturom. Ima registarski fajl sa 32 registra. Svi registri i adrese su 32-bitni. U asemblerskom kodu untar 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.
Implementirati bibliotečnu C funkciju koja poziva opisanu uslugu:

```
int vm_alloc(int starting_page, int size_in_pages);
```

Rešenje:

2. (10 poena) Operativni sistem Linux

Napisati *shell script* koji generiše e-mail adrese svih studenata koji su predali projektni zadatak. Direktorijum u kome se nalaze projektni zadaci zadat je kao prvi argument pri pozivu, a datoteka u koju se upisuju e-mail adrese zadata je kao drugi argument. Ime datoteke u kojoj se nalazi projektni zadatak svakog studenta zapisana je u formatu:

ime_prezime_brIndeksa.zip – gde znak '_' predstavlja separator između polja.

Potrebno je generisati e-mail adresu u sledećem formatu:

<prvo slovo prezimena><prvo slovo imena><broj indeksa>d@student.etf.bg.ac.rs

Pretpostaviti da su imena fajlova u zadatom direktorijumu korektno formatirana. Ukoliko nije prosleđen odgovarajući broj parametara, ispisati poruku o grešci. Takođe, ako prvi parametar nije direktorijum ili drugi parametar ne predstavlja datoteku u koju se može izvršiti upis, ispisati odgovarajuću poruku o grešci. Na kraju izvršavanja skripta potrebno je ostati u polaznom direktorijumu. Primer imena datoteke u kojoj se nalazi projektni zadatak:

```
pera_zikic_110034.zip
```

Potrebno je generisati e-mail adresu:

```
zp110034d@student.etf.bg.ac.rs
```

Rešenje:

3. (10 poena) Operativni sistem Linux

Na jeziku C/C++, koristeći mehanizam prosleđivanja poruka operativnog sistema Linux, dati rešenje problema filozofa koji večeraju (*dining philosophers*), pri čemu je data funkcija `philosopher` koja kao argument prima jedinstveni broj (identifikator) filozofa. Svaki filozof pri slanju zahteva identifikuje se ovim brojem. Takođe, navedena je struktura poruka koje se razmenjuju, kao i značenje vrednosti svakog polja.

```
#define MESSAGE_Q_KEY 1

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

void philosopher(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 forks
  struct requestMsg msg;
  msg.mtype = id;
  msg.msg[0] = (char) 1;
  msgsnd(requestMsgQueueId, &msg, len, 0);
  msgrcv(responseMsgQueueId, &msg, len, id, 0);

  //eat
  sleep(1);

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

  //think
  sleep(1);
 }
}
```

Napisati implementaciju centralnog procesa koji na početku nad funkcijom `philosopher` kreira potreban broj filozofa predstavljenih procesima, a zatim u vidu konobara (*waiter*) komunicira sa filozofima i obezbeđuje njihovu sinhronizaciju. Nije potrebno proveravati uspešnost izvršavanja operacije nad sandučićima (*message queue*).

Rešenje: