Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1 (SI2OS1)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo
Kolokvijum: Prvi, Mart 2012.
Datum: 25.3.2012.

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	*Zadatak 4*	_____/10

Ukupno: _____/40 = _____% = _____/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 dva izlazna uređaja:

```
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
```

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`).

Potrebno je napisati kod koji vrši prenos po jednog bloka podataka na svaki od dva uređaja uporedo, na prvi uređaj tehnikom prozivanja (*polling*), a na drugi korišćenjem prekida. Transfer se obavlja pozivom sledeće funkcije iz koje se vraća kada su oba prenosa završena:

```
void transfer (unsigned* blk1, int count1, unsigned* blk2, int count2);
```

Rešenje:

2. (10 poena)

Neki računar podržava straničnu organizaciju virtuelne memorije, pri čemu je virtuelni adresni prostor veličine 16GB, adresibilna jedinica je 32-bitna reč, a fizički adresni prostor je veličine 1GB. Stranica je veličine 1KB. U deskriptoru stranice u jednom ulazu u tabeli preslikavanja stranica (PMT) najviši bit ukazuje na to da li je stranica u memoriji ili ne, a najniži biti predstavljaju broj okvira u fizičkoj memoriji u koji se stranica preslikava. Deskriptor stranice ne sadrži druge informacije.

a)(5) Prikazati logičku strukturu virtuelne i fizičke adrese i označiti širinu svakog polja. Ako je početak PMT nekog procesa na fizičkoj adresi FF00h, na kojoj adresi je deskriptor stranice ABCh tog procesa?

b)(5) Na jeziku C napisati kod funkcije

```
void setPageDescr(unsigned* pmtp, unsigned page, unsigned frame);
```

koja u tabelu na čiji početak ukazuje dati pokazivač `pmtp`, za stranicu sa datim brojem `page`, upisuje deskriptor tako da se ta stranica preslikava u okvir sa datim brojem `frame`.

Rešenje:

3. (10 poena)

Neki C kompajler ne koristi uobičajeni pristup alokaciji lokalnih promenljivih i argumenata funkcija, pa ih ne smešta na kontrolni procesorski stek (onaj na kome procesor implicitno čuva povratnu adresu prilikom skoka u potprogram mašinskom instrukcijom `call`). Umesto toga, ovaj kompajler koristi sledeću tehniku za alokaciju i pristup lokalnim promenljivim i argumentima funkcija.

Za svaku lokalnu promenljivu i argument funkcije definisanu u programu, kompajler statički (u vreme prevođenja) alocira jedan (i samo jedan) deo memorije za smeštanje jedne instance te promenljive/argumenta. Pristup toj promenljivoj ili argumentu u kodu funkcije je onda rešen statičkim vezivanjem, direktnim memorijskim adresiranjem te jedne i uvek iste lokacije koja je poznata u vreme prevođenja. Da bi se podržali rekurzivni pozivi, svakoj takvoj promenljivoj i argumentu pridružena je jedna (i samo jedna) globalna struktura podataka koju kompajler organizuje. Ta struktura podataka implementira LIFO protokol; drugim rečima, svakoj lokalnoj promenljivoj i argumentu pridružen je jedan LIFO stek na kome se čuvaju prethodne vrednosti te promenljive/argumenta u slučaju ugnježdenih i rekurzivnih poziva. Prilikom poziva funkcije, tekuća vrednost iz statičke lokacije za svaku lokalnu promenljivu/argument se smešta na vrh njegovog sopstvenog steka, sa koga se restaurira prilikom izlaska iz pozvane funkcije. Ovo radi kod koga generiše kompajler na odgovarajućim mestima.

a)(5) Na asembleru napisati prevod koji bi napravio ovaj kompajler za sledeću rekurzivnu C funkciju:

```
int f (int n) {
  if (n==0) return 0;
  else return f(n-1)+1;
}
```

Kod koji generiše kompajler za smeštanje vrednosti lokalne promenljive/argumenta koji je alociran na adresi simbolički označenoj sa `X` predstaviti sledećim makroima (ti makroi biće zamenjeni odgovarajućim kodom koji održava pomenuti stek pridružen promenljivoj `X` i koji ima određeni isti oblik parametrizovan adresom `X`, a koji ovde nije od interesa):

```
push(X)
pop(X)
```

b)(5) Objasniti šta je problem koji se mora prevazići ako se za ovaj kompajler želi napraviti višenitno jezgro poput školskog jezgra i kako se taj problem može prevazići u implementaciji jezgra, a bez izmene kompajlera.

Rešenje:

4. (10 poena)

Dat je sledeći kod koji koristi školsko jezgro.

```
class TreeNode {
public:
  TreeNode* getLeftChild();
  TreeNode* getRightChild();
  void  process();
  ...
};

class TreeVisitor : public Thread {
public:
  TreeVisitor (TreeNode* root) : myRoot(root) {}
protected:
  virtual void run();
  void visit(TreeNode*);
private:
  TreeNode* myRoot;
};

void TreeVisitor::run () {
  visit(myRoot);
}

void TreeVisitor::visit (TreeNode* node) {
  if (node==0) return;
  TreeNode* rn = node->getRightChild();
  if (rn) (new TreeVisitor(rn))->start();
  node->process();
  visit(node->getLeftChild());
}

void userMain () {
  TreeNode* root = ...;
  Thread* thr = new TreeVisitor(root);
  thr->start();
}
```

Napisati C kod koji radi isto, odnosno obilazi dato stablo rekurzivnim kreiranjem niti koje obilaze podstabla, samo korišćenjem nekog sistema u kome se niti kreiraju i odmah pokreću kao parametrizovani pozivi funkcije sledećim sistemskim pozivom:

```
void create_thread (void(*thread_body)(void*), void* param);
```

Klasa `TreeNode` implementirana je sledećim C kodom:

```
struct TreeNode;
TreeNode* getLeftChild (TreeNode*);
TreeNode* getRightChild (TreeNode*);
void process (TreeNode*);
```

Rešenje: