Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1 (SI2OS1, IR2OS1)
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Treći, jun 2015.
Datum: 19.6.2015.

Treći 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 = _____% = _____/10

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) Ulaz/izlaz

U nekom sistemu svaki drajver blokovski orijentisanog uređaja („diska“) registruje sledeću strukturu (tabelu) koja sadrži pokazivače na funkcije koje implementiraju odgovarajuće operacije sa tim uređajem:

```
typedef ... Byte; // Unit of memory
typedef ... BlkNo; // Disk block number

typedef int (*DiskOperation)(BlkNo block, Byte* buffer);

struct DiskOperationsTable {
  int isValid;
  DiskOperation readBlock, writeBlock;
  DiskOperationsTable () : isValid(0), readBlock(0), writeBlock(0) {}
};
```

Sistem organizuje tabelu registrovanih drajvera za priključene uređaje kao niz ovih struktura, s tim da polje `isValid==1` označava da je dati element niza zauzet (validan, postavljen, odnosno disk je registrovan), a 0 da je ulaz slobodan (disk nije registrovan):

```
const int MaxNumOfDisks; // Maximal number of registered disk devices
DiskOperationsTable disks[MaxNumOfDisks];
```

Sistem preslikava simbolička imena dodeljena priključenim uređajima, u obliku slova abecede, brojevima ulaza u tabeli `disks` (u opsegu od 0 do `MaxNumOfDisks-1`).

a)(5) Realizovati funkcije:

```
int readBlock(int diskNo, BlkNo block, Byte* buffer);
int writeBlock(int diskNo, BlkNo block, Byte* buffer);
```

koje treba da pozovu odgovarajuću implementaciju operacije drajvera (polimorfno, dinamičkim vezivanjem) za zadati uređaj. (Ove funkcije poziva interna kernel nit kada opslužuje zahteve za operacijama sa diskovima, da bi inicijalizovala prenos na odgovarajućem uređaju.)

b)(5) Realizovati funkciju koja registruje operacije drajvera za dati disk:

```
int registerDriver(int diskNo, DiskOperation read, DiskOperation write);
```

U slučaju greške, sve ovde navedene funkcije vraćaju negativnu vrednost, a u slučaju uspeha vraćaju 0.

Rešenje:

2. (10 poena) Fajl sistem

Neki fajl sistem primenjuje deljene (*shared*) i ekskluzivne (*exclusive*) ključeve za pristup fajlu. Za operacije čitanja (*read, r*) i izvršavanja (*execute, x*) fajla potreban je deljeni ključ, a za operaciju upisa (*write, w*) ekskluzivni ključ.

Tokom operacije otvaranja fajla, operativni sistem proverava da li je operacija koju je zahtevao proces dozvoljena (u smislu ključa) i zaključava fajl odgovarajućim ključem ukoliko jeste dozvoljena. Ovo se obavlja u sledećoj funkciji `lock`:

```
int lock(FCB* file, unsigned int op);
const int OP_RD = 4;
const int OP_WR = 2;
const int OP_EX = 1;
```

Zahtevana operacija se identifikuje jednim od tri bita *rwx* u argumentu `op`, s tim što je uvek tačno jedan bit postavljen na 1. Za potrebe maskiranja tih bita definisane su konsante `OP_RD`, `OP_WR` i `OP_EX`. Funkcija treba da vrati 1 ako je operacija dozvoljena, a 0 ako nije.

Pre poziva ove funkcije, FCB traženog fajla je već učitan u memoriju i na njega ukazuje prvi argument. U toj strukturi, pored ostalog, postoje i celobrojna polja `sharedLock` i `exclLock` za deljeni i ekskluzivni ključ nad datim fajlom (1-zaključan, 0-otključan).

Realizovati funkciju `lock`.

Rešenje:

3. (10 poena) Fajl sistem

U implementaciji nekog fajl sistema evidencija slobodnih blokova na disku vodi se pomoću bit-vektora koji se kešira u memoriji u nizu `blocks` veličine `NumOfBlocks/BITS_IN_BYTE` (konstanta `NumOfBlocks` predstavlja broj blokova na disku). Svaki element ovog niza je veličine jednog bajta, a svaki bit odgovara jednom bloku na disku (1-zauzet, 0-slobodan). Pretpostaviti da je `NumOfBlocks` umnožak broja 8 (`BITS_IN_BYTE`).

```
typedef unsigned char byte;
const unsigned int BITS_IN_BYTE = 8;
const unsigned long NumOfBlocks = ...;
byte blocks[NumOfBlocks/BITS_IN_BYTE];
```

a)(3) Implementirati sledeće dve funkcije:

```
void blockToBit(unsigned long blkNo, unsigned long& bt, byte& mask);
void bitToBlk(unsigned long& blkNo, unsigned long bt, byte mask);
```

Funkcija `blockToBit` prima kao ulazni parametar redni broj bloka `blkNo` i na osnovu njega izračunava i upisuje u izlazne parametre broj bajta (`bt`) u bit-vektoru i jednu jedinu jedinicu u onaj razred parametra `msk` koji odgovara bitu u tom bajtu za dati blok. Funkcija `bitToBlock` radi obrnutu konverziju: za ulazni parametar koji je redni broj bajta u bit-vektoru (`bt`) i najniži razred u parametru `msk` koji je postavljen na 1, izračunava i upisuje u izlazni parametar `blk` redni broj bloka koji odgovara tom bajtu i bitu.

b)(7) Korišćenjem funkcija pod a), implementirati funkcije `allocateBlock` i `freeBlock`. Funkcija `allocateBlock` treba da pronađe prvi slobodan blok i označi ga zauzetim. Ako takav blok nađe, vraća broj tog bloka, a ako slobodnog bloka nema, vraća 0 (blok broj 0 je rezervisan i nikada se ne koristi u fajl sistemu). Kako bi se optimizovao pristup fajlovima, slobodan blok se traži u blizini bloka sa datim rednim brojem `startingFrom`, na sledeći način:

- slobodan blok treba tražiti počev od datog bloka naviše (i alocirati prvi slobodan na koji se naiđe), pri tom može (ali ne mora) da se alocira i bilo koji slobodan blok koji je u istom bajtu bit-vektora kao i dati blok, bez obzira da li je ispred ili iza datog bloka (i on se smatra dovoljno bliskim).
- ukoliko se na ovaj način ne pronađe, počinje se pretraga od prvog bloka (blok broj 0 je rezervisan) pa sve do zadatog bloka `startingFrom`.

Funkcija `freeBlock` označava dati blok slobodnim.

```
unsigned long allocateBlock (unsigned long startingFrom);
void freeBlock (unsigned long blk);
```

Rešenje: