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 2013.
Datum: 14.6.2013.

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

Neki sistem organizuje keš blokova sa diska na sledeći način. Keš čuva najviše `CACHESIZE` blokova veličine `BLKSIZE` u nizu `diskCache`. Svaki ulaz *i* u nizu `diskCacheMap` sadrži broj bloka na disku koji se nalazi u ulazu *i* keša. Vrednost 0 u nekom ulazu niza `diskCacheMap` označava da je taj ulaz prazan (u njega nije učitan blok). Data je sledeća implementacija keša:

```
typedef ... Byte; // Unit of memory
typedef ... BlkNo; // Disk block number
const int BLKSIZE = ...; // Disk block size in Bytes
const int CACHESIZE = ...; // Disk cache size in blocks

Byte diskCache [CACHESIZE][BLKSIZE]; // Disk cache
BlkNo diskCacheMap [CACHESIZE]; // The contents of disk cache. 0 for empty

Byte* getDiskBlock (BlkNo blk) {
  // Search for the requested block in the cache and return it if found:
  int hash = blk%CACHESIZE;
  int cursor = hash;
  for (int i=0; i<CACHESIZE; i++) {
    cursor = (hash+i)%CACHESIZE;
    if (diskCacheMap[cursor]==blk) return diskCache[cursor];
    if (diskCacheMap[cursor]==0) break;
  }
  // Not found.
  if (diskCacheMap[cursor]!=0) cursor=hash; // Cache full
  // If there is a block to evict, write it to the disk:
  if (diskCacheMap[cursor]!=0)
    diskWrite(diskCacheMap[cursor],diskCache[cursor]);
  // Load the requested block:
  diskCacheMap[cursor] = blk;
  diskRead(blk,diskCache[cursor]);
  return diskCache[cursor];
}
```

Operacije `diskRead` i `diskWrite` vrše sinhrono čitanje, odnosno upis datog bloka sa diska.

Funkcija `getDiskBlock` vraća pokazivač na deo memorije u kome se nalazi učitan traženi blok diska, pri čemu se taj blok učitava u keš (uz prethodno snimanje eventualno izbačenog bloka) ukoliko taj blok već nije u kešu. Ovu funkciju koristi ostatak sistema za pristup blokovima diska. Ova operacija implementira keš kao heš (*hash*) tabelu, s tim da koliziju rešava otvorenim adresiranjem.

Modifikovati funkciju `getDiskBlock` tako da se svaki blok na disku uvek smešta u isti ulaz u kešu, određen istom navedenom heš funkcijom.

Rešenje:

2. (10 poena) Interfejs fajl sistema

U nekom interfejsu fajl sistema[1] definisano je nekoliko funkcija za rad sa direktorijumima. Dat je izvod iz dokumentacije jedne od njih:

```
struct dirent * readdir (DIR *dirstream);
```

This function reads the next entry from the directory `dirstream`*... If there are no more entries in the directory or an error is detected,* `readdir` *returns a null pointer.*

Prevod: Ova funkcija čita sledeći ulaz u direktorijumu `dirstream`... Ako više nema ulaza u direktorijumu ili je došlo do greške, `readdir` vraća *null* pokazivač.

Napomena: simbol . označava tekući direktorijum.

Šta radi sledeći program?

```
#include <stddef.h>
#include <stdio.h>
#include <sys/types.h>
#include <dirent.h>

int main (void)
{
  DIR *dp;
  struct dirent *ep;

  dp = opendir ("./");
  if (dp != NULL)
    {
      while (ep = readdir (dp))
        puts (ep->d_name);
      closedir (dp);
    }
  else
    puts ("Couldn't open the directory.");

  return 0;
}
```

Odgovor:

[1] Radi se o standardnom POSIX API.

3. (10 poena) Implementacija fajl sistema

U implementaciji nekog fajl sistema definisani su celobrojni tip `Byte` koji predstavlja bajt, kao i celobrojni tip `BlkNo` koji predstavlja broj logičkog bloka sadržaja fajla (numeracija počev od 0). U strukturi FCB celobrojno polje `cur` predstavlja kurzor za čitanje i upis (u bajtovima, počev od 0), a polje `size` stvarnu veličinu u bajtovima. Konstanta `BLKSIZE` predstavlja veličinu bloka u bajtovima. Na raspolaganju je funkcija koja učitava logički blok datog fajla i vraća pokazivač na taj učitani blok u kešu (vraća 0 u slučaju greške):

```
Byte* readFileBlock (FCB* file, BlkNo blockNo);
```

Realizovati funkciju `fread()` koja za dati fajl učitava `n` bajtova u dati bafer, počev od kurzora, i vraća broj stvarno učitanih bajtova, a kurzor pomera na kraj učitane sekvence. U slučaju prekoračenja veličine sadržaja fajla ili druge greške treba vratiti broj stvarno učitanih bajtova:

```
int fread (FCB* file, Byte* buffer, int n);
```

Rešenje: