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 2017.
Datum: 11.6.2017.

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 implementira se keš blokova sa blokovskih uređaja („diskova“) kodom koji je dat u nastavku. Za svaki uređaj sa datim identifikatorom pravi se jedan objekat klase `BlockIOCache`, inicijalizovan tim identifikatorom, koji predstavlja keš blokova sa tog uređaja. Keš čuva najviše `CACHESIZE` blokova veličine `BLKSIZE` u nizu `cache`. Svaki ulaz *i* u nizu `cacheMap` sadrži broj bloka na disku koji se nalazi u ulazu *i* keša. Flegovi u ulazu *i* niza `flags` ukazuju na to da je u ulazu *i* validan sadržaj (da ulaz nije prazan, odnosno već je bio korišćen i u njega je učitan sadržaj nekog bloka) – fleg `F_VALID`, odnosno da je sadržaj „zaprljan“, tj. menjan i treba ga upisati na disk – fleg `F_DIRTY`.

Kod koji koristi ovaj keš koristi ga na sledeći način za neku složenu operaciju sa uređajem:

```
Byte* buffer = diskCache->getBlock(blk);
...
buffer[...] = ...;
...
diskCache->writeBlock(buffer);
...
diskCache->releaseBlock(buffer);
```

Na početku ovakve složene operacije sa uređajem, ovaj kod najpre traži da keš obezbedi da je potrebni blok učitan pozivom funkcije `getDiskBlock` koja vraća pokazivač na niz bajtova u baferu – učitanom bloku. Potom ovaj kod može da čita ili upisuje u sadržaj tog bafera. Ako je upisivao, mora da pozove funkciju `writeBlock`, kako bi keš znao da dati blok treba da upiše na uređaj. Kada završi celu operaciju, kod poziva funkciju `releaseBlock`, kako bi kešu stavio do znanja da on više ne koristi taj blok. Pošto više ovakvih složenih operacija može biti ugnježđeno, blok iz keša može biti izbačen (zamenjen drugim) samo ako ga više niko ne koristi, što se realizuje brojanjem referenci u nizu `refCounter`.

Implementirati funkcije `getBlock` i `releaseBlock`. Funkcija `getBlock` treba da obezbedi da je traženi blok u kešu, odnosno učita ga ako nije; ako ga treba učitati, to može biti urađeno u bilo koji ulaz keša koji sadrži blok koji nije validan ili nije više korišćen; ako takvog nema, treba vratiti 0 (greška, nema mesta u kešu). Na raspolaganju su sledeće dve funkcije koje vrše učitavanje, odnosno upis (respektivno) bloka na dati uređaj:

```
void ioRead(int device, BlkNo blk, Byte* buffer);
void ioWrite(int device, BlkNo blk, Byte* buffer);
typedef char Byte; // Unit of memory
typedef long long BlkNo; // Device block number

class BlockIOCache {
public:
  BlockIOCache (int device);

  Byte* getBlock (BlkNo blk);
  void  writeBlock (Byte* buffer);
  void  releaseBlock (Byte* buffer);

private:
  static const unsigned BLKSIZE = ...; // Block size in Bytes
  static const unsigned CACHESIZE = ...; // Cache size in blocks
  static const short F_VALID = 0x01; // Valid bit mask
  static const short F_DIRTY = 0x02; // Dirty bit mask

  int dev;
  Byte cache [CACHESIZE][BLKSIZE]; // Cache
  BlkNo cacheMap [CACHESIZE]; // Contents of the cache (block numbers)
  unisgned refCounter [CACHESIZE]; // Reference counters
  short flags [CACHESIZE]; // Valid, Dirty flags
};
```

```
BlockIOCache::BlockIOCache (int device) : dev(device) {
  for (int i=0; i<CACHESIZE; i++)
    this->cacheMap[i] = this->refCounter[i] = this->flags[i] = 0;
}

void BlockIOCache::writeBlock (Byte* buffer) {
  int i = (buffer-this->cache[0])/BLKSIZE;
  if (i<0 || i>=CACHESIZE) return; // Exception
  this->flags[i] |= F_DIRTY;
}
```

Rešenje:

2. (10 poena) Fajl sistem

Sistemi bazirani na sistemu Unix podržavaju strukture direktorijuma tipa acikličnog usmerenog grafa (DAG) pomoću dve vrste referenci na fajl kao objekat u fajl sistemu:

- *soft* (*symbolic*) *link*: „meka (ili simbolička) veza“ predstavlja fajl posebnog tipa čiji sadržaj čuva proizvoljnu (apsolutnu ili relativnu) putanju do nekog drugog fajla, poput „prečice“ (*shortcut*); svaku operaciju (komandu) nad ovakvom vezom sistem preusmerava na referencirani fajl, osim komande `rm` za brisanje – brisanje simboličke veze briše samu vezu, ne i referencirani fajl; ukoliko referencirani fajl nestane ili se premesti, veza ostaje neažurna, „viseća“;
- *hard link*: „tvrda veza“ predstavlja jedan ulaz u direktorijumu koji referencira određeni fajl kao objekat (tj. njegov *inode*); na jedan fajl može ukazivati više tvrdih veza; komanda `rm` uklanja tvrdu vezu, a sam fajl se implicitno briše iz sistema kada nestane poslednja tvrda veza na njega.

Iz komandne linije mogu se izvršiti sledeće sistemske komande:

`ls` — prikazuje sadržaj tekućeg direktorijuma (*list*)

`cd <dir>` — menja tekući direktorijum (*change directory*)

`ln <src> <dst>` — za postojeći fajl sa zadatom stazom `<src>` kreira novu tvrdu vezu sa datom stazom `<dst>` (*link*)

`ln -s <src> <dst>` — za postojeći fajl sa zadatom stazom `<src>` kreira novu meku vezu sa datom stazom `<dst>`

`rm <file>` — briše ulaz sa zadatim imenom iz tekućeg direktorijuma; ukoliko je to poslednja tvrda veza na fajl, briše se i sam fajl.

Sve staze mogu biti apsolutne ili relativne. Zabeležena je sledeća sesija jednog korisnika:

```
> cd /home/docs
> ls
bar  foo  txt
> cd /home/pics
> ls
jane  john  chld
> ln /home/docs/bar  foo
> rm john
> cd /home/docs
> rm bar
> cd /home/pics
> ls
```

a)(5) Napisati izlaz poslednje komande.

Odgovor: _____

b)(5) Nakon prikazane sekvence zadate su sledeće komande:

```
> ln -s  /home/docs/foo  bar
> rm /home/docs/foo
> rm foo
> cd /home/docs
> ls
```

Napisati izlaz poslednje komande.

Odgovor: _____

3. (10 poena) Fajl sistem

U implementaciji nekog FAT fajl sistema cela FAT keširana je u memoriji u strukturi:

```
extern unsigned long fat[];
```

Za ulančavanje se kao *null* vrednost u ulazu u FAT koristi 0 (blok broj 0 je rezervisan). U FCB fajla polje `head` sadrži redni broj prvog bloka sa sadržajem fajla i polje `size` koje sadrži veličinu sadržaja fajla u bajtovima. Slobodni blokovi su označeni ulazima u FAT sa vrednošću -1. Implementirati sledeću funkciju koja treba da obriše sadržaj fajla:

```
void truncate (FCB* fcb);
```

Rešenje: