Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1
Nastavnik: prof. dr Dragan Milićev
Odsek: Računarska tehnika i informatika, Softversko inženjerstvo
Kolokvijum: Treći, avgust 2023.
Datum: 27. 8. 2023.

Treći kolokvijum iz Operativnih sistema 1

Kandidat: _____

Broj indeksa: _____ *E-mail:*_____

Kolokvijum traje 90 minuta. Dozvoljeno je korišćenje literature.

Zadatak 1 _____/10 *Zadatak 3* _____/10
Zadatak 2 _____/10

Ukupno: _____/30 = _____% = _____/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)

U nekom sistemu implementira se keš blokova sa blokovskih uređaja 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 je kapaciteta `CACHESIZE` blokova veličine `BLKSIZE`. Keš je interno organizovan kao heš mapa `map` sa `MAPSIZE` ulaza. Svaki ulaz niza `map` sadrži glavu liste keširanih blokova koji se preslikavaju u taj ulaz. Funkcija `hash` je heš funkcija koja preslikava broj bloka u ulaz u nizu `map`. Glava liste, kao i pokazivač na sledeći element u listi čuvaju se kao indeksi elementa niza `entries` koji sadrži keširane blokove; vrednost -1 označava kraj (*null*). Svaki element niza `entries` je struktura tipa `CacheEntry` u kojoj je polje `blkNo` broj bloka koji je keširan u tom elementu, polje `next` ukazuje na sledeći element liste, a polje `buf` je sadržaj samog bloka.

Na početku složene operacije sa uređajem, kod koji koristi keš najpre zahteva da potrebni blok bude učitan pozivom funkcije `getBlock` koja vraća pokazivač na niz bajtova u baferu – učitanom bloku. Pošto više ovakvih složenih operacija može biti pokrenuto uporedo, blok iz keša može biti izbačen (zamenjen drugim) samo ako ga više niko ne koristi, što se realizuje brojanjem referenci u polju `refCounter` strukture `CacheEntry`. Prikazana je implementacija funkcije `getBlock` koja treba da obezbedi da je traženi blok u kešu, odnosno učita ga ako nije.

Potrebno je implementirati pomoćnu funkciju `getFreeEntry` koja treba da vrati indeks slobodnog ulaza u nizu `entries` u koji se može učitati traženi blok u keš. Inicijalno je keš prazan i svi ulazi u njemu su slobodni. Ova funkcija treba redom da zauzima elemente niza `entries`, sve dok ima slobodnih. Kada slobodnih ulaza više nema, ona treba da izbaci blok (snimi ga na disk) u prvom ulazu koji je na redu po FIFO redosledu (najstariji učitan), ali samo pod uslovom da se blok u tom ulazu ne koristi (tj. njegov `refCounter` je nula). Ako to nije zadovoljeno, treba da proba sa sledećim i tako u krug. Ako nema mesta u kešu jer nijedan blok ne može da se izbaci, treba vratiti -1. Ukoliko je potrebno dodati ili izmeniti članove ove klase, precizno navesti kako to treba uraditi. Na raspolaganju je i funkcija koja učitava blok, odnosno upisuje blok na dati uređaj:

```
void ioRead (int device, BlkNo blk, Byte* buffer);
void ioWrite(int device, BlkNo blk, Byte* buffer);

typedef unsigned char Byte; // Unit of memory
typedef long BlkNo; // Device block number
const unsigned BLKSIZE = ...; // Block size in Bytes

class BlockIOCache {
public:
  BlockIOCache (int device);
  Byte* getBlock (BlkNo blk);
  ...
protected:
  static int hash (BlkNo);
  int getFreeBlock ();
private:
  static const unsigned CACHESIZE = ...; // Cache size in blocks
  static const unsigned MAPSIZE = ...; // Hash map size in entries

  struct CacheEntry { BlkNo blkNo; int next; int refCounter; Byte buf[BLKSIZE]; };

  int dev;
  int map[MAPSIZE];  // Hash map
  CacheEntry entries[CACHESIZE]; // Cache
  ...
};
```

```
Byte* BlockIOCache::getBlock (BlkNo blk) {
  // Find the requested block in the cache and return it if present:
  int entry = hash(blk);
  for (int i=map[entry]; i!=-1; i=entries[i].next)
    if (entries[i].blkNo==blk) {
      entries[i].refCounter++;
      return entries[i].buf;
    }
  // The block is not in the cache, find a free slot to load it:
  int free = getFreeEntry();
  if (free==-1) return 0;  // Error: cannot find space
  // Load the requested block:
  entries[free].blkNo = blk;
  entries[free].refCounter = 1;
  entries[free].next = map[entry];
  map[entry] = free;
  ioRead(dev,blk,entries[free].buf);
  return entries[free].buf;
}
```

Rešenje:

2. (10 poena)

Dat je podsetnik na neke osnovne Unix komande:

- `cat`: iz svakog fajla koji je naveden kao argument ove komande, redom kojim su oni navedeni, učitava znakove i ispisuje ih na standardni izlaz; ukoliko nema argumenata, znakove učitava sa standardnog ulaza (dok ne naiđe na znak EOF koji se na konzoli signalizira pritiskom tastera Ctrl+D).
- `ln` *`src_file dst_file`*: pravi novu tvrdu vezu (ulaz u direktorijumu) za postojeći fajl *src_file* u odredištu definisanom sa *dst_file*; opcija -s radi isto to, samo što pravi meku (simboličku) vezu.
- `rm`: briše fajl ili direktorijum ili vezu zadat kao argument.

Izvršavaju se sledeće komande po datom redosledu; napisati te komande ili odgovoriti na postavljeno pitanje:

1. U postojećem poddirektorijumu `a` tekućeg direktorijuma napraviti nov tekstualni fajl `doc` čiji će sadržaj biti učitan sa konzole:

_____

2. U postojećem poddirektorijumu `b` tekućeg direktorijuma napraviti tvrdu vezu pod nazivom `hdoc` na fajl napravljen u koraku 1:

_____

3. U postojećem poddirektorijumu `c` tekućeg direktorijuma napraviti meku vezu pod nazivom `sdoc` na fajl napravljen u koraku 2:

_____

4. Obrisati `doc` u poddirektorijumu a:

_____

5. Šta će nakon ovoga ispisati komanda `cat b/hdoc`?

_____

6. Nakon izdate komande `rm b/hdoc`, šta će ispisati komanda `cat c/sdoc`?

_____

3. (10 poena)

U implementaciji nekog fajl sistema svaki čvor u hijerarhijskoj strukturi direktorijuma i fajlova predstavljen je objektom klase `Node`. Operacija te klase:

```
Node* Node::getSubnode(const char* pStart, const char* pEnd);
```

vraća podčvor datog čvora `this` koji ima simboličko ime zadato nizom znakova koji počinje znakom na koga ukazuje `pStart`, a završava znakom ispred znaka na koga ukazuje `pEnd` (`pEnd` može ukazivati na `'\0'` ili `'/'`). Ukoliko dati čvor `this` nije direktorijum, ili u njemu ne postoji podčvor sa datim simboličkim imenom, ova funkcija vraća 0. Koreni direktorijum cele hijerarhije dostupan je kao statički pokazivač `Node::root` tipa `Node*`. Znak za razdvajanje (`delimiter`) u stazama, kao i znak za koreni dorektorijum je kosa crta `'/'`.

Data je implementacija funkcije `getNodeRel` koja za (sintaksno ispravnu) relativnu putanju datu nizom znakova koji počinje znakom na koga ukazuje `pStart` i završava se znakom ispred znaka na koji ukazuje `pEnd` vraća čvor definisan tom putanjom u odnosu na dati čvor.

Korišćenjem ove funkcije, implementirati funkciju `Node::getNodeAbs` koja za apsolutnu putanju (garantovano je sintaksno ispravna i počinje znakom `'/'`) vraća čvor određen tom putanjom, uz korišćenje keša preslikavanja apsolutnih putanja u čvorove `DentryCache`. Data funkcija `DentryCache::getNode` vraća ranije zapamćen čvor za apsolutnu putanju datu nizom znakova zadatog sa `pStart` i `pEnd`. Ukoliko traženi čvor postoji, treba ažurirati keš unosom ulaza koji preslikava datu apsolutnu putanju u dati čvor pozivom funkcije `DentryCache::storeNode`.

```
static const char delimiter = '/';

Node* Node::getNodeRel (const char* pStart, const char* pEnd, Node* node) {
  while (node && pStart<pEnd) {
    const char* pE = pStart+1;
    while (pE<pEnd && *pE!=delimiter) pE++;
    node = node->getSubnode(pStart,pE);
    pStart = (pE<pEnd)?(pE+1):pE;
  };
  return node;
}

Node* DentryCache::getNode(const char* pStart, const char* pEnd);
void DentryCache::storeNode(const char* pStart, const char* pEnd, Node*);

Node* Node::getNodeAbs (const char* pStart, const char* pEnd);
```

Rešenje: