Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 2 (IR3OS2, SI3OS2)
Nastavnik: prof. dr Dragan Milićev
Školska godina: 2012/2013. (Zadatak važi počev od januarskog roka 2013.)

Projekat za domaći rad
- Projektni zadatak –

Verzija dokumenta: 1.2

Važne napomene: Pre čitanja ovog teksta, **obavezno** pročitati opšta pravila predmeta i pravila vezana za izradu domaćih zadataka! Pročitati potom ovaj tekst **u celini i pažljivo**, pre započinjanja realizacije ili traženja pomoći. Ukoliko u zadatku nešto nije dovoljno precizno definisano ili su postavljeni kontradiktorni zahtevi, student treba da uvede razumne pretpostavke, da ih temeljno obrazloži i da nastavi da izgrađuje preostali deo svog rešenja na temeljima uvedenih pretpostavki. Zahtevi su namerno nedovoljno detaljni, jer se od studenata očekuje kreativnost i profesionalni pristup u rešavanju praktičnih problema!

Uvod

Cilj ovog zadatka jeste implementacija uprošćenog školskog fajl sistema. Fajl sistem je namenjen za particije na standardnim hard diskovima. Za pristup particijama na hard diskovima obezbeđena je apstrakcija u vidu klase `Partition` koja pruža sve potrebne operacije za pristup podacima na disku. Fajl sistem treba da obezbedi montiranje zadate particije. Svakoj particiji se pri montiranju dodeljuje prvo slobodno veliko slovo engleskog alfabeta kojim će particija dalje biti identifikovana. Time je broj particija koje istovremeno mogu biti montirane u sistemu ograničen na 26. Potrebno je obezbediti mogućnost montiranja particije sa postojećim fajl sistemom, kao i montiranje prazne particije i njeno kasnije formatiranje na školski fajl sistem.

Opšti zahtevi

Odnos projekta i korisničke aplikacije

Tražene podsisteme treba realizovati na jeziku C++. Korisnička aplikacija, koja će sadržati test primere, treba povezati sa prevedenim kodom projekta u jedinstven konzolni program (.exe). Druga biblioteka, `partition.lib`, sadržaće apstrakciju particije namenjenu za pristup particijama na hard diskovima.

Odnos projekta i operativnog sistema domaćina

Projekat se realizuje pod operativnim sistemom Windows XP, koji je u ovom slučaju operativni sistem domaćin. Izradom projekta se ni na koji način ne sme ugroziti ispravno funkcionisanje operativnog sistema domaćina. Svaki eventualni problem koji se pojavi po pokretanju projekta biće smatran kao greška pri izradi projekta. Po završetku rada, okruženje je neophodno ostaviti u neizmenjenom stanju u odnosu na trenutak pre pokretanja projekta, osim onih delova koji se namerno menjaju samim test primerom koji treba da proveri ispravnost projekta. Svi resursi u sistemu koji će biti korišćeni pri izradi projekta moraju biti korišćeni kroz odgovarajući API operativnog sistema domaćina koji je za to namenjen. Deo koda koji je obezbeđen u okviru postavke projekta je pažljivo napisan, i ukoliko se koristi u skladu sa uputstvom za rad, ne može prouzrokovati nikakve probleme i greške pri izvršavanju.

Zadatak: Školski fajl sistem

Uvod

Za trajno čuvanje podataka fajl sistema koji će biti implementiran u okviru ovog projekta biće korišćeni hard diskovi. Svaki korišćeni disk će biti particionisan. Radi pojednostavljenja rešenja, obezbeđen je interfejs za pristup pojedinačnoj particiji. U sastavu interfejsa koji je obezbeđen nalaze se sledeće operacije:

- kreiranje objekta koji će predstavljati zadatu particiju;
- dobijanje opštih informacija o particiji;
- čitanje sadržaja zadatog klastera na zadato mesto u memoriji;
- upis sadržaja sa zadatog mesta iz memorije u zadati klaster particije.

Fajl sistem treba da obezbedi interfejs za montiranje i demontiranje particija i za rad sa fajlovima. Na svakoj particiji postoji samo jedan direktorijum, koreni direktorijum (*root*), i u njemu su smešteni svi fajlovi. Interakcija sa uređajem na kojem su podaci smešteni vrši se kroz prethodno opisanu apstrakciju.

Za pristup fajl sistemu korisniku treba obezbediti poseban interfejs kroz koji će videti fajl sistem i u njemu postojeće fajlove. Taj interfejs treba obezbediti u vidu dve klase.

Prva klasa treba da apstrahuje fajl sistem na korisničkom nivou. Ta klasa treba da podrži sve opšte operacije fajl sistema:

- dohvatanje opštih informacija o fajl sistemu (korisnik zadaje oznaku particije za koju želi informacije);
- montiranje i demontiranje particije u fajl sistem, tj. dodeljivanje slova particiji;
- formatiranje zadate montirane particije;
- ispitivanje postojanja fajla;
- izbegavanje mrtvog blokiranja pomoću bankarevog algoritma;
- brisanje i otvaranje zadatog fajla;

Druga klasa treba da apstrahuje fajl na korisničkom nivou. Jedan objekat ovog tipa, za nit u čijem je kontekstu kreiran, predstavlja jedan otvoren fajl iz fajl sistema. Jedini način kreiranja objekta koji predstavlja fajl je pozivom operacije otvaranja fajla koja će biti obezbeđena u okviru fajl sistema. Operacije koje ova klasa treba da obezbedi su:

- čitanje i upis niza bajtova zadate veličine od tekuće pozicije;
- dohvatanje, provera i izmena tekuće pozicije u fajlu; svaka nit treba da ima svoju tekuću poziciju;
- brisanje dela fajla od tekuće pozicije do kraja;
- zatvaranje fajla uništavanjem objekta koji predstavlja otvoren fajl, čime se fajl oslobađa za korišćenje od strane drugih niti.

Ove dve klase su na vrhu slojevite strukture fajl sistema i jedine su koje će krajnji korisnik direktno koristiti pri radu sa fajl sistemom. Na dnu hijerarhije se nalazi data apstrakcija particije. Studentima se ostavlja da osmisle i implementiraju nedostajući deo hijerarhije fajl sistema poštujući sva ograničenja navedena u ovoj postavci.

Particija

Opis datog interfejsa za pristup particiji

Za potrebe trajnog čuvanja podataka fajl sistema, studentima je data na raspolaganje klasa koja predstavlja jednu particiju. Operacije koje ova klasa u svom interfejsu obezbeđuje su (opisi su poređani po istom redosledu kao i metode u klasi):

- kreiranje objekta particije; pri kreiranju se zadaje naziv konfiguracionog fajla koji sadrži sve ostale informacije potrebne za formiranje ovog objekta;
- dohvatanje informacije o broju klastera na particiji;
- čitanje jednog (i samo jednog) klastera; zadaje se redni broj klastera (klasteri su numerisani počev od 0); pročitani podaci se smeštaju na zadato mesto u memoriji; operacija vraća: 0 – neuspeh ili 1 – uspeh; na pozivaocu leži odgovornost da u memoriji obezbedi slobodan prostor dovoljne veličine za smeštanje traženih podataka;
- upis jednog (i samo jednog) klastera; zadaje se redni broj klastera za upis (klasteri su numerisani počev od 0); podaci za upis se čitaju sa zadatog mesta u memoriji; vraća: 0 – neuspeh ili 1 – uspeh;

Sve date operacije su *thread-safe*, što znači da se potpuno bezbedno mogu pozivati iz konkurentnih niti. Operacije čitanja i upisa su blokirajuće. Sa ovim operacijama moguće je pristupiti samo onim klasterima koji u potpunosti pripadaju particiji. Radi pojednostavljenja, za veličinu klastera je uzeto 2kB i taj parametar nije moguće menjati.

Particija na jeziku C++

Particija je realizovana klasom `Partition` čija se potpuna definicija nalazi u zaglavlju `part.h`. Interfejs ove klase definisan je u sledećem fajlu:

```
// File: part.h

typedef unsigned long ClusterNo;
const unsigned long ClusterSize = 2048;


class Partition {
public:
      Partition(char *);

      virtual ClusterNo getNumOfClusters() const;

      virtual int readCluster(ClusterNo, char *buffer);
      virtual int writeCluster(ClusterNo, const char *buffer);

      virtual ~Partition();
private:
      PartitionImpl * myImpl;

};
```

Fajl sistem

Opis zadatka

Potrebno je realizovati školski fajl sistem sa ulančanom indeksnom alokacijom klastera za fajlove. U nastavku su data ograničenja koja određuju način smeštanja fajlova na disku i

format ulaza korenog direktorijuma. Particija na kojoj je smešten ovakav fajl sistem ima sledeće sekcije:

- Početni (nulti) klaster - Na početku klastera nalaze se dva pokazivača širine 32 bita, prvi koji ukoliko je potrebno sadrži broj narednog klastera sa ulazima korenog direktorijuma, u suprotnom sadrži vrednost 0; drugi predstavlja broj prvog klastera u listi slobodnih klastera. U nastavku ovog klastera nalaze se ulazi korenog direktorijuma. Ukoliko je potrebno klasteri korenog direktorijuma ulančavaju se u dvostruko ulančanu listu pomoću ista dva pokazivača širine 32 bita koji se nalaze na početku.
- Oblast za indekse i podatke – ovde se nalaze klasteri sa indeksima i podacima; klaster je osnovna i nedeljiva jedinica podataka za koju se na disku vodi evidencija;

Radi poboljšanja performansi svaki fajl se sastoji od jednostruko ulančane liste indeksnih klastera koji ukazuju na klastere sa podacima. Na početku indeksnog klastera nalazi se pokazivač širine 32 bita koji sadrži broj sledećeg indeksnog klastera u listi. Ostatak indeksnog klastera su ulazi (takođe širine 32 bita) koji predstavljaju pokazivače na klastere sa podacima. Ukoliko je neki ulaz u indeksnom klasteru prazan ili neki pokazivač predstavlja kraj liste, u njemu je zapisana vrednost 0 (klaster 0 ne može se pojaviti kao klaster koji sadrži podatke ili indekse nekog fajla). Klasteri za podatke sadrže isključivo podatke. Takođe, za ulančavanje slobodnih klastera koristi se jednostruko ulančana lista.

Koreni direktorijum na disku zapisuje se na sledeći način: nakon 2 pokazivača specijalne namene, nalazi se niz ulaza koji opisuju sadržaj direktorijuma. Svaki ulaz u direktorijumu zauzima 20 bajtova. Polja koja se nalaze u svakom ulazu i bajtovi koje zauzimaju su:

- 0x00, 8 bajtova – naziv ulaza; maksimalno osam znakova; dopunjeno razmacima sa kraja; posebne vrednosti prvog bajta:
  - 0x00 – slobodan ulaz za dalje korišćenje;
- 0x08, 3 bajta – ekstenzija; maksimalno tri karaktera; dopunjeno razmacima sa kraja;
- 0x0b, 1 bajt – ne koristi se, upisati 0;
- 0x0c, 4 bajta – broj indeksnog klastera koji predstavlja početak datog fajla; za prazne fajlove na ovu poziciju upisati 0;
- 0x10, 4 bajta – veličina fajla u bajtovima;

U klaster treba smestiti samo ceo broj ulaza. Drugim rečima, jedan ulaz nikad se ne smešta jednim delom u jedan klaster i drugim delom u drugi klaster (razmisliti da li je ova situacija moguća). Brojevni podaci u ulazima korenog direktorijuma, kao i pokazivači unutar klastera, smeštaju se po *little endian* formatu (niži bajt je smešten na nižoj adresi). Slobodni ulazi posle kojih više nema zauzetih ulaza imaju vrednost nula u svih 20 bajtova.

Opis funkcionalnosti fajl sistema

Potrebno je realizovati klasu `FS` sa sledećim funkcionalnostima (opisi su poređani po istom redosledu kao i metode u klasi):

- montiranje particije u fajl sistem, tj. dodjeljivanje slova particiji; od ovog trenutka particija se može koristiti sve do trenutka demontiranja; vraća: slovo koje je dodeljeno particiji ili '0' ako je već montirano maksimalnih 26 particija;
- demontiranje particije iz fajl sistema; particija se zadaje prosleđivanjem slova koje je dodeljeno particiji. Nit koja demontira particiju blokira se sve dok se ne zatvore svi fajlovi sa date particije i odjavi njihovo korišćenje, a dalje otvaranje fajlova nije dozvoljeno.
- formatiranje particije zadate dodeljenim slovom; potrebno je inicijalizovati pokazivač na slobodne klastere, kao i *root* direktorijum. Nit koja formatira particiju blokira se

sve dok se ne zatvore svi fajlova sa date particije i odjavi njihovo korišćenje, a dalje otvaranje fajlova nije dozvoljeno.

- ispitivanje postojanja zadatog fajla; vraća: 0-ne postoji ili 1-postoji.
- čitanje sadržaja korenog direktorijuma; radi pojednostavljenja čita se niz uzastopnih validnih ulaza u korenom direktorijumu maksimalne dužine 64 počevši od zadatog ulaza - *i*; vraća: ceo broj sa značenjem: 0-neuspeh, 1-64 broj pročitanih ulaza i to su svi ulazi u direktorijumu od zadate pozicije *i*, 65-uspeh, niz je pun i nije pročitan ostatak ulaza u direktorijumu;
- najava, odnosno odjava korišćenja nekog fajla; vraća: 0-neuspeh ili 1-uspeh; radi izbegavanja mrtvog blokiranja pomoću bankarevog algoritma, fajl sistem vodi evidenciju o fajlovima koje svaka nit koristi; korisnik je obavezan da na početku najavi sve fajlove koje će koristiti, kao i da odjavi one koje više neće koristiti; prilikom odjavljivanja korišćenja nekog fajla potrebno je proveriti da li je taj fajl zatvoren; nakon odjavljivanja nije dozvoljeno ponovno otvaranje nekog fajla.
- otvaranje fajla sa zadatim nazivom (zadaje se apsolutnom putanjom); vraća: pokazivač na objekat otvorenog fajla ili *null* ako operacija otvaranja nije uspela usled greške; potrebno je proveriti da li nit koja poziva ovu operaciju je prethodno najavila korišćenje ovog fajla; u slučaju da fajl ne postoji kreira se novi, u suprotnom otvara se postojeći; fajl se otvara nedeljivo, za upis i čitanje; kursor se postavlja na početak; s obzirom da pristup na nivou fajla treba da podržava princip izbegavanja mrtvog blokiranja pomoću bankarevog algoritma, ukoliko fajl trenutno ne može biti otvoren (npr. sistem će preći u nebezbedno stanje), operacija treba da blokira pozivajuću nit dok se ne steknu svi uslovi za bezbedno otvaranje fajla;
- brisanje zadatog fajla (zadaje se apsolutnom putanjom), pod uslovom da je fajl zatvoren i da su sve ostale niti odjavile korišćenje datog fajla; vraća: 0-neuspeh ili 1-uspeh; Izbrisani fajl automatski se briše i iz liste najvljenih fajlova za korišćenje niti koja poziva ovu operaciju; Radi poboljšanja performansi prilikom brisanja vrši se kompakcija niza zauzetih ulaza tako što se u izbrisani ulaz prepisuje poslednji zauzeti ulaz. Takođe, potrebno je dealocirati poslednji klaster korenog direktorijuma ukoliko više nema ni jednog zauzetog ulaza u njemu.

Putanje u ovom fajl sistemu se označavaju kao u operativnom sistemu Windows. Koreni direktorijum particije kojoj je dodeljeno slovo 'X' označen je sa "X:\". Putanja je uvek apsolutna.

Fajl sistem na jeziku C++

Potrebno je realizovati klasu `FS` čija je puna deklaracija data u zaglavlju "fs.h" i koja izgleda ovako:

```
// File: fs.h

typedef unsigned long BytesCnt;
typedef unsigned long EntryNum;

const unsigned long ENTRYCNT=64;
const unsigned int FNAMELEN=8;
const unsigned int FEXTLEN=3;

struct Entry {
  char name[FNAMELEN];
  char ext[FEXTLEN];
  char reserved;
  unsigned long firstIndexCluster;
```

```
  unsigned long size;
};

typedef Entry Directory[ENTRYCNT];

class KernelFS;
class Partition;
class File;

class FS {
public:

 ~FS ();

  static char mount(Partition* partition); //montira particiju
                                     // vraca dodeljeno slovo
                                     // ili 0 u slucaju neuspeha

  static char unmount(char part);  //demontira particiju oznacenu datim
               // slovom vraca 0 u slucaju neuspeha ili 1 u slucaju uspeha

  static char format(char part); // formatira particiju sa datim slovom;
              // vraca 0 u slucaju neuspeha ili 1 u slucaju uspeha

  static char readRootDir(char part, EntryNum n, Directory &d);
         //prvim argumentom se zadaje particija, drugim redni broj
         //validnog ulaza od kog se počinje čitanje,
         //treci argument je adresa na kojoj se smesta procitani niz ulaza

  static char doesExist(char* fname); //argument je naziv fajla zadat
                                      //apsolutnom putanjom

  static char declare(char* fname, int mode);
      //prvi argument je naziv fajla zadat apsolutnom putanjom,
      //drugi mod:
      // 1 - najava koriscenja, 0 - odjava koriscenja zadatog fajla

  static File* open(char* fname);
  static char deleteFile(char* fname);

protected:
  FS ();
  static KernelFS *myImpl;
};
```

Opis funkcionalnosti fajla

Operacije koje treba obezbediti za rad sa fajlovima su (opisi su poređani po istom redosledu kao i metode u klasi):

- upis na tekuću poziciju u fajl zadatog broja bajtova sa zadate memorijske adrese; vraća: 0-neuspeh ili 1-uspeh; fajl se proširuje upisom na kraj fajla, ukoliko već nije dostignuta maksimalna veličina; po upisu, tekuća pozicija se pomera na kraj upisanog sadržaja; u slučaju upisa unutar postojećeg sadržaja fajla podaci se prepisuju preko postojećih (*overwrite*);
- čitanje sa tekuće pozicije iz fajla zadatog broja bajtova na zadatu adresu u memoriji; po čitanju, tekuća pozicija se pomera na kraj pročitanog sadržaja; na pozivaocu je odgovornost da obezbedi dovoljan prostor za smeštanje ovog niza znakova; vraća: 0-neuspeh (npr. ako je fajl zatvoren ili je pred početak čitanja tekuća pozicija bila na

kraju fajla) ili i > 0 - broj pročitanih bajtova (može se desiti da je pročitano manje nego što je traženo jer se stiglo do kraja fajla);

- pomeranje tekuće pozicije u fajlu (binarni fajl, pozicija se računa po rednom broju bajta u fajlu, počinje se od 0); zadaje se nova apsolutna pozicija; vraća: 0-neuspeh ili 1-uspeh;
- dohvatanje tekuće pozicije u fajlu;
- provera da li je tekuća pozicija u fajlu kraj tog fajla; vraća: 0-nije, 1-greška i 2-jeste;
- dohvatanje trenutne veličine fajla u bajtovima;
- brisanje dela fajla od tekuće pozicije do kraja; vraća: 0-neuspeh ili 1-uspeh;
- zatvaranje fajla, čime se fajl oslobađa za korišćenje od strane drugih niti; korisnik je obavezan da zatvori svaki otvoreni fajl; nakon zatvaranja fajla, sve ostale operacije su neuspešne. Kada se oslobodi fajl koji je neka nit pokušala da otvori i zbog toga bila suspendovana, ispituje se ponovo da li bi tada dati zahtev odveo sistem u nebezbedno stanje primenom bankarevog algoritma. Ako bi, nit se i dalje ostavlja suspendovanom. Ako ne bi, data nit se deblokira, a njen zahtev zadovoljava otvaranjem datog fajla. Raspoređivanje pristupa deblokiranih niti na nivou fajla treba da podržava princip FCFS (*First Come – First Served*).

Za svaki fajl je dozvoljeno alocirati samo onoliko klastera kolika je trenutna stvarna potreba u skladu sa trenutnom veličinom fajla. Prilikom brisanja fajla neophodno je osloboditi sve klastere koji ne predstavljaju stvarnu potrebu.

Fajl na jeziku C++

Fajl treba realizovati kao klasu čije metode pružaju korisniku usluge koje fajl sistem treba da obezbedi za rad nad fajlovima. Kompletna deklaracija klase `File` je zadata u zaglavlju `file.h`, i izgleda ovako:

```
// File: file.h
class KernelFile;
class File {
public:
  char write (BytesCnt, char* buffer);
  BytesCnt read (BytesCnt, char* buffer);
  char seek (BytesCnt);
  BytesCnt filePos();
  char eof ();
  BytesCnt getFileSize ();
  char truncate ();
  ~File();                                  //zatvaranje fajla

private:
  friend class FS;
  friend class KernelFS;
  File ();  //objekat fajla se može kreirati samo otvaranjem
  KernelFile *myImpl;
};
```

Klasa `KernelFile` i podatak-član `myImpl` imaju isti smisao kao i kod niti, semafora i događaja iz projekta iz OS1, tj. klasa `File` predstavlja samo omotač (*wrapper*) oko klase `KernelFile` i enkapsulira pozive svih njenih metoda.

Testovi

Javni testovi

Javni test-program služi da pomogne studentima da istestiraju svoj projekat. Ovi testovi neće obavezno pokriti sve funkcionalnosti koje projekat treba da ima, ali će istestirati većinu tih funkcionalnosti. Da bi se projekat uopšte odbranio, neophodno je da projekat sa javnim testom radi u potpunosti ispravno. Studentima se preporučuje da pored javnog testa naprave i svoje testove koji će im pomoći da što bolje istestiraju svoj projekat.

Tajni testovi

Tajni testovi detaljnije testiraju sve zahtevane funkcionalnosti u različitim regularnim i neregularnim situacijama (greške u pozivu ili radu uređaja), i nisu unapred dostupni studentima.

Testovi performansi

Testovi performansi mere vreme izvršavanja pojedinačnih operacija. Ovi testovi nisu obavezni, i mogu, ali ne moraju, doneti dodatne bodove u prvom ispitnom roku posle nastave za do 10 najboljih radova odbranjenih pre roka.

Zaključak

Potrebno je realizovati opisane podsisteme prema datim zahtevima na jeziku C++. Kao integrisano okruženje za razvoj programa (engl. *integrated development environment*, IDE) zahteva se Microsoft Visual studio 2008 radi kompatibilnosti sa testovima. (Testiranje se vrši pod operativnim sistemom Windows XP.)

Pravila za predaju projekta

Projekat se predaje isključivo kao jedna zip arhiva. Sadržaj arhive podeliti u dva foldera: *src* i *h*. U prvom folderu (*src*) treba da budu smešteni svi *.cpp* fajlovi koji su rezultat izrade projekta, a u drugom folderu (*h*) treba da budu svi *.h* fajlovi koji su rezultat izrade projekta. Opisani sadržaj ujedno treba da bude i jedini sadržaj arhive (arhiva ne sme sadržati ni izvršne fajlove, ni biblioteke, ni *.cpp* i *.h* fajlove koji predstavljaju bilo kakve testove, niti bilo šta što iznad nije opisano). Projekat je moguće upload-ovati više puta, ali do trenutka koji će preko e-mail liste biti objavljen za svaki ispitni rok i koji će uvek biti pre ispita. Na serveru uvek ostaje samo poslednja predata verzija i ona će se koristiti na odbrani. Za izlazak na ispit neophodno je predati projekat (prijava ispita i kolokvijumi su takođe preduslovi za izlazak na ispit).

Sajt za predaju projekta je http://rti.etf.rs/rti/os/os2/index.php

Nepoštovanje pravila za predaju projekta povlači negativne poene.

Zapisnik revizija

Ovaj zapisnik sadrži spisak izmena i dopuna ovog dokumenta po verzijama.

Verzija 1.2

Strana	Izmena
6.	Dodato pojašnjenje: Ukoliko je potrebno klasteri korenog direktorijuma ulančavaju se u dvostruko ulančanu listu pomoću ista dva pokazivača širine 32 bita koji se nalaze na početku.
6.	Dodato pojašnjenje, proširene sledeće rečenice: Nit koja formatira/demontira particiju blokira se sve dok se ne zatvore svi fajlovi sa date particije i odjavi njihovo korišćenje, a dalje otvaranje fajlova nije dozvoljeno.
7.	Operacija najave, odnosno odjave korišćenja nekog fajla vraća: 0-neuspeh ili 1-uspeh;