2. (10 poena)

U nekom operativnom sistemu struktura PCB sadrži polje `open_files` koje je pokazivač na tabelu otovorenih fajlova tog procesa. U toj tabeli (zapravo nizu), svaki ulaz je struktura koja predstavlja deskriptor jednog otvorenog fajla. U toj strukturi postoji celobrojno polje `access` čija vrednost 1 označava da je dati proces otvorio dati fajl sa pravom na upis, a vrednost 0 označava da je dati proces otvorio dati fajl samo sa pravom na čitanje. Promenljiva tipa `FHANDLE` predstavlja indeks u tabeli otvorenih fajlova datog procesa u čijem je kontekstu otvaren fajl. Implementirati operaciju `check_access()` čiji je potpis dat, a koju poziva fajl podsistem u svakom sistemskom pozivu za pristup sadržaju fajla radi provere prava datog procesa na izvršavanje date operacije nad sadržajem fajla. Argument `p` je pokazivač na PCB procesa koji je izdao sistemski poziv, argument `f` je identifikator fajla, a argument `write` ima vrednost 0 ako sistemski poziv zahteva samo čitanje, odnosno vrednost 1 ako sistemski poziv zahteva upis u dati fajl.

```
int check_access (PCB* p, FHANDLE f, int write);
```

Rešenje:

```
int check_access (PCB *p, FHANDLE f, int write){
    if (write == 1) {
        return (p->open_files)[f].access == 1;
    } else {
        return 1;
    }
}
```

U G O
rwx rwx rwx
1 0

2. (10 poena) Fajl sistem

U nekom fajl sistemu primenjuje se zaštita pristupa fajlovima kao u sistemu Unix. U strukturi FCB postoje sledeća polja:

- `unsigned long int owner`: identifikator korisnika koji je vlasnik fajla;
- `unsigned long int group`: identifikator grupe korisnika kojoj je fajl pridružen;
- `unsigned int protection`: biti prava pristupa (relevantno je samo 9 najnižih bita, pri čemu najviša 3 su dodeljena vlasniku, naredna 3 grupi, a najniža 3 ostalima).

U strukturi UCB *(user control block)* koja predstavlja jednog registrovanog korisnika sistema takođe postoji polje `group` koje predstavlja identifikator grupe kojoj je taj korisnik pridružen. Prava pristupa za grupu kojoj je pridružen fajl odnose se na korisnike koji su pridruženi istoj toj grupi.

Na raspolaganju je i sledeća funkcija koja vraća pokazivač na odgovarajuću strukturu UCB za korisnika koji je identifikovan datim identifikatorom:

```
UCB* getUCB (unsigned long int uid);
```

Realizovati funkciju koja ispituje da li je tražena operacija dozvoljena datom korisniku za dati fajl. Operacija se identifikuje jednim od tri najniža bita *rwx*, sa značenjem kao u bitima prava pristupa za fajl, s tim što je uvek samo jedan bit postavljen na 1. Funkcija treba da vrati 1 ako je operacija dozvoljena, a 0 ako nije.

```
int isAllowed(FCB* file, unsigned long int uid, unsigned int op);
```

Rešenje:

```
int isAllowed(FCB *file, ulong uid, uint op){
    if (file->owner == uid){
        return ((file->protection) >> 6) & op > 0;
    }

    UCB *ucb = getUcb(uid);
    if (ucb->group == file->group){
        return ((file->protection) >> 3) & op > 0;
    }

    return (file->protection) & op > 0
}
```

2. (10 poena) Interfejs fajl sistema

U nekom fajl sistemu u sistemskom pozivu za otvaranje fajla proces navodi da li će fajl samo čitati ili ga i na bilo koji način menjati. U zavisnosti od toga, taj sistemski poziv zaključava fajl sa jednim od dve vrste ključa. Ako se fajl otvara samo za čitanje, fajl se zaključava deljenim ključem; ako se fajl otvara za izmenu, zaključava se ekskluzivnim ključem. Ukoliko poziv ne može da se izvrši zbog toga što ključ ne može da se dobije, poziv se otkazuje bez izmena u fajl sistemu i vraća se greška.

Procesi A, B, C i D izvršavaju sistemske pozive otvaranja i zatvaranja istog fajla u sledećem redosledu (neki proces izvršava poziv zatvaranja fajla samo ako ga je uspešno otvorio):

1) A: open(READ) ✓
2) B: open(WRITE) ✗
3) C: open(READ) ✓
4) A: close ✓
5) C: close ✓
6) D: open(WRITE) ✓

Koje od ovih operacija će se izvršiti uspešno, a koje neuspešno?
Precizno obrazložiti odgovor!

Odgovor:

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 konstante `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:

```
int lock(FCB *file, uint op) {
    if (op == OP_RD || op == OP_EX) {
        if (file->exclLock == 1) {
            return -1;
        } else {
            file->sharedLock += 1;
            return 0;
        }
    } else {
        if (file->exclLock == 1 || file->sharedLock > 0) {
            return -1;
        } else {
            file->exclLock = 1;
            return 0;
        }
    }
}
```

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*)

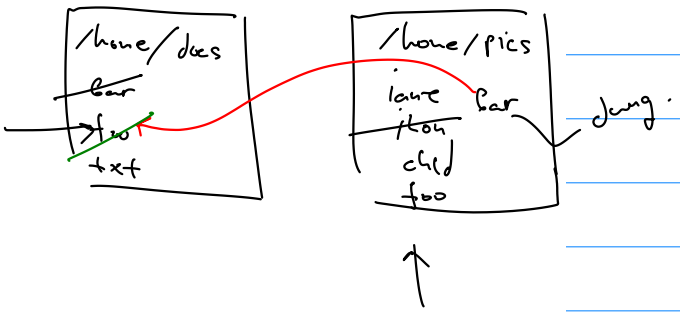
`ln <src> <dst>` za postojeći fajl sa zadatom stazom `<src>` kreira novu tvrdu vezu sa datom stazom `<dst>` (*link*)

`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: jane, chld, foo.

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: txt

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.

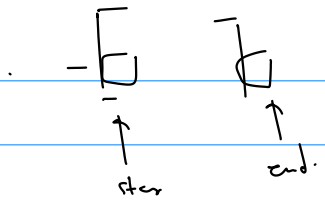
Šta radi sledeći program?

```
#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;
}
```



2. (10 poena) Fajl sistem

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.

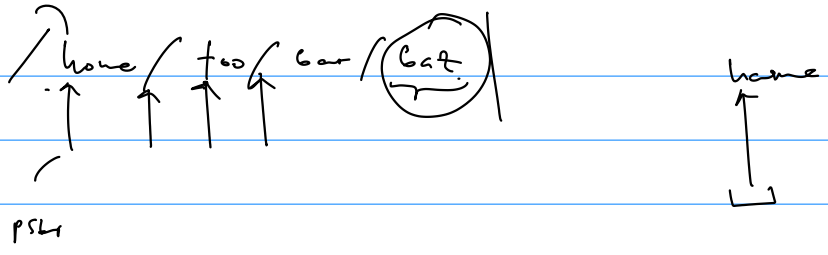
Za svaki proces se u polju `curDir` strukture PCB čuva pokazivač na čvor (tipa `Node*`) koji predstavlja tekući direktorijum datog procesa. Koreni direktorijum cele hijerarhije dostupan je kao statički pokazivač `Node::rootNode` tipa `Node*`.

Znak za razdvajanje (delimiter) u stazama, kao i znak za koreni direktorijum je kosa crta `'/'`. Implementirati sledeću funkciju koja se koristi u implementaciji ovog fajl sistema:

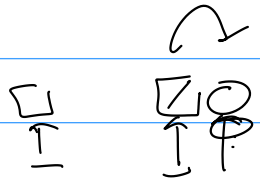
```
Node* Node::getNode (PCB* pcb, const char* path);
```

Ova funkcija vraća čvor određen stazom koja je zadataka drugim argumentom, pri čemu ta staza može biti zadata kao apsolutna (počinje znakom `'/'`), ili kao relativna (ne počinje znakom `'/'`) u odnosu na tekući direktorijum procesa čiji je PCB dat kao prvi argument.

Rešenje:



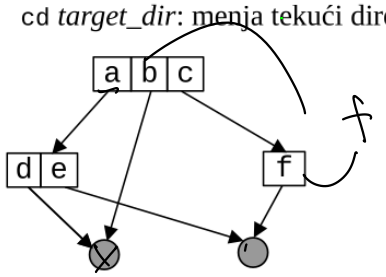
```
Node * Node::getNode(PCB *pcb, const char *path) {
    char *pStart = path;
    char *pEnd = nullptr;
    Node *cd;
    if (*path == '/') {
        cd = Node::root;
        pStart += 1;
    } else {
        cd = pcb->curDir;
    }
    while (1) {
        pEnd = pStart;
        while (*pEnd != '/' && *pEnd != '\0') pEnd += 1;
        cd = cd->getSubnode(pStart, pEnd);
        if (cd == 0) return 0;
        if (*pEnd == '\0') return cd;
        pStart = pEnd + 1;
    }
}
```



2. (10 poena) Fajl sistem

Ispisati sekvencu Unix komandi kojima korisnik, kroz CLI, može da napravi strukturu (pod)direktorijuma (na slici predstavljeni pravougaonicima) i fajlova (na slici predstavljeni kružićima) koja je prikazana na datoj slici, unutar tekućeg direktorijuma koji je inicijalno prazan; „koreni“ direktorijum prikazan na slici predstavlja taj tekući direktorijum (u njemu treba formirati ulaze *a*, *b*, i *c*). Fajlove praviti tako da im se sadržaj učitava sa konzole (unos znaka Ctrl-D označava kraj ulaznog toka na konzoli); ne pisati unos koji se unosi na konzoli ~~za sadržaj~~ fajla. Sve strelice na slici predstavljaju tvrde veze (*hard links*). Na raspolaganju su sledeće komande:

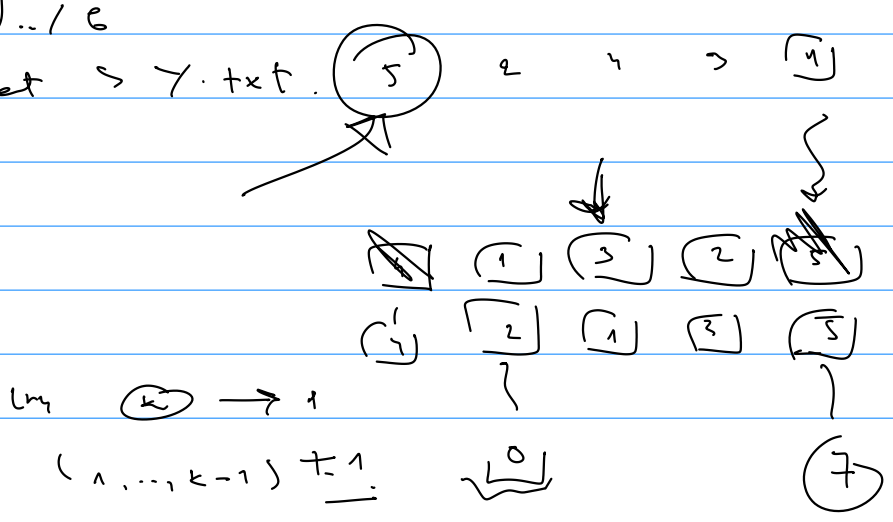
- `mkdir` *directory*: pravi navedeni direktorijum sa podrazumevanim pravima pristupa; pretpostaviti da su ta podrazumevana prava pristupa dovoljna da tekući korisnik uradi sve potrebne komande;
- `cat`: ukoliko nije naveden ulazni fajl, ovaj sistemski program sadržaj koji daje na svoj standardni izlaz uzima sa svog standardnog ulaza;
- `ln` *original_filename link_name*: pravi tvrdu vezu *link_name* ka fajlu *original_filename*;
- `cd` *target_dir*: menja tekući direktorijum na zadati.



Rešenje:

```
$ mkdir a
$ mkdir b
$ mkdir c
$ cd a
$ mkdir d
$ mkdir e
$ cd ../c
$ mkdir f
$ cd f
$ cat > Y.txt
$ cd ../b
$ cat > Y.txt
```

```
$ cd ../a
$ cd d
$ ln /e/X.txt X.txt
$ cd ../e
$ ln /c/f/Y.txt Y.txt
```



3. (10 poena) Fajl sistem

U implementaciji nekog fajl sistema keš struktura FCB svih otvorenih fajlova i korišćenih direktorijuma („globalna tabela otvorenih fajlova“) organizuje se u memoriji kao objekat klase `FCBCache`. U tom kešu jedan FCB čuva se u jednom ulazu predstavljenom strukturom `FCBEntry` čiji je deo definicije dat dole. Polje `id` čuva identifikator FCB-a koji je učitan u ovaj ulaz (npr. broj bloka na disku), polje `refCnt` je brojač referenci, a polje `fcb` sam sadržaj FCB-a.

```
struct FCBEntry {
  FCBID id;  // ID of the FCB stored in this entry
  unsigned long refCnt = 0; // Reference counter
  FCB fcb;  // FCB of the node
  ... uint lru_idx;
};
```

Kada neki deo kernela želi da koristi neki FCB, recimo kada proces otvara fajl, poziva se operacija `FCBCache::request(FCBID)` koja pronalazi ulaz u kome se već nalazi traženi FCB ili ga učitava ukoliko on nije u kešu (tabeli). Tom prilikom se `refCnt` tog ulaza inkrementira ili postavlja na 1 ukoliko je dati FCB tek učitan. Slično, kada taj korisnik više ne koristi taj FCB, poziva operaciju `FCBCache::release(FCBID)` koja „oslobađa“ taj FCB, ali on ostaje u kešu ukoliko ponovo bude potreban, dok se eventualno odatle ne izbaci. Keš funkcioniše tako što nekorišćeni ulazi (oni kod kojih je `refCnt==0`) ostaju u kšeu osim ukoliko se ne zahteva prostor za učitavanje novog FCB-a, a u kešu više nema slobodnih ulaza. Tada se kao „žrtva za izbacivanje“ bira onaj nekorišćeni ulaz koji je najdavnije bio korišćen – LRU (*least recently used*) politikom zamene.

Implementirati sledeće pomoćne nestatičke funkcije članice klase `FCBCache` koje se pozivaju iz implementacije navedenih operacija `FCBCache::request` i `FCBCache::relase`:

- `void FCBCache::updateLRUonRequest (FCBEntry* f)`: ova operacija poziva se iz operacije `FCBCache::request` koja je pronašla odgovarajući ulaz `f`; ona treba da po potrebi ažurira evidenciju potrebnu za LRU algoritam zamene; pre poziva ove operacije `f->refCnt` je već ažuriran (inkrementiran);
- `void FCBCache::updateLRUonRelease (FCBEntry* f)`: ova operacija poziva se iz operacije `FCBCache::release` koja oslobađa odgovarajući ulaz `f`; ona treba da po potrebi ažurira evidenciju potrebnu za LRU algoritam zamene; pre poziva ove operacije `f->refCnt` je već ažuriran (dekrementiran);
- `FCBEntry* FCBCache::getLRUVictim ()`: ova operacija poziva se iz operacije `FCBCache::request` po potrebi, ukoliko nema slobodnog mesta za učitavanje traženog FCB-a; ona treba da vrati ulaz koji je po LRU algoritmu odabran kao žrtva za zamenu i po potrebi ažurira evidenciju potrebnu za LRU algoritam zamene; ukoliko takvog nema, treba da vrati *null*.

Navesti proširenja klase `FCBCache` i strukture `FCBEntry` potrebnim članovima.

class FCBCache {
⋮
FCBEntry cash[cashSize];
⋮
}.

void FCBCache::updateLRUonRequest (FCBEntry* f) {
int lru_idx = f -> lru_idx;