Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Treći, jun 2024.
Datum: 9. 6. 2024.

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 = _____% = _____/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

Korišćenjem biblitečnih funkcija *popen*, *dup2* i *fileno* implementirati funkciju *redirect* čiji je potpis dat i koja pokreće proces dete nad programom u fajlu sa putanjom zadatom argumentom `exe`, a onda preusmerava standardni izlaz pozivajućeg (svog) procesa na standardni ulaz tog procesa deteta; u slučaju neuspeha vraća -1, u slučaju uspeha vraća 0. Biblitečna funkcija *fileno* vraća celobrojni deskriptor fajla (*fd*) koji odgovara zadatom znakovnom toku `stream`.

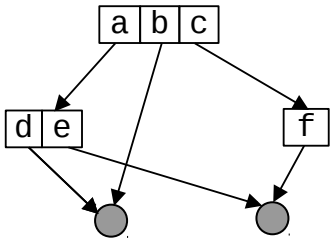
```
int fileno (FILE* stream);
int redirect (const char* exe);
```

Rešenje:

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:

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
  ...
};
```

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.

Rešenje:

Rešenja zadataka za treći kolokvijum iz Operativnih sistema 1
Jun 2024.

1. (10 poena)

```
int redirect (const char* exe) {
  FILE* out = popen(exe,"w");
  if (!out) return -1;
  int fd = fileno(out);
  dup2(fd,1);
  return 0;
}
```

2. (10 poena)

```
mkdir a
cat >b
mkdir c
ln b a/d
cat >a/e
ln a/e c/f
```

3. (10 poena)

U strukturu `FCBEntry` treba dodati sledeće članove za ulančavanje u dvostruko ulančanu LRU listu ulaza koji se ne koriste:

```
FCBEntry *lruPrev = 0, *lruNext = 0;  // For the LRU double-linked list
```

U klasu `FCBCache` treba dodati sledeće članove koji predstavljaju glavu i rep LRU liste nekorišćenih ulaza; lista je uređena po hronologiji korišćenja:

```
FCBEntry *lruHead = 0, *lruTail = 0; // LRU double-linked list
```

Pomoćna nestatička funkcija članica koja izbacuje dati ulaz iz LRU liste (ako je on u listi):

```
inline void FCBCache::removeFromLRU (FCBEntry* f) {
  // If not in the LRU list, return:
  if (!f->lruPrev && !f->lruTail && this->lruHead!=f) return;
  // Else, remove it from the LRU list:
  if (f->lruNext) f->lruNext->lruPrev = f->lruPrev;
  else this->lruTail = f->lruPrev;
  if (f->lruPrev) f->lruPrev->lruNext = f->lruNext;
  else this->lruHead = f->lruNext;
  f->lruNext = f->lruTail = 0;
}

inline void FCBCache::updateLRUonRequest (FCBEntry* f) {
  if (f && f->refCnt==1) this->removeFromLRU(f);
}
```

```
inline void FCBCache::updateLRUonRelease (FCBEntry* f) {
  if (f && f->refCnt==0) {  // Insert it at the head of the LRU list
    f->lruNext = this->head;
    f->lruPrev = 0;
    if (this->lruHead) this->lruHead->lruPrev = f;
    else this->lruTail = f;
    this->lruHead = f;
  }
}

FCBEntry* getLRUVictim () {
  FCBEntry* f = this->lruTail;  // Get it from the tail of the LRU list
  if (f) this->removeFromLRU(f);
  return f;
}
```

Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Treći, jun 2023.
Datum: 11. 6. 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 = _____% = _____/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 ulazno-izlaznom podsistemu nekog operativnog sistema zahtevi za operacijama sa diskom predstavljaju se instancama strukture `DiskRequest`. Jedan zahtev odnosi se na prenos `blockCount` susednih blokova počev od bloka broj `startBlockNo`, a prenos se vrši sa baferom `buffer` i u smeru koji zadaje polje `dir`.

```
struct DiskRequest {
  uint32  blockCount;
  uint32  startBlockNo;
  uint32* buffer;
  enum Dir : uint32 {in=0, out=1};
  Dir dir;
  ... // Other details irrelevant here
};
```

Za svaki fizički uređaj formira se poseban red `RequestQueue` ovakvih zahteva. Svaki takav red opslužuje po jedna interna nit jezgra koja izvršava funkciju `diskDriver` prikazanu dole. Ova funkcija uzima jedan po jedan zahtev iz reda i svaki zahtev zadaje drajveru uređaja `dd` pozivom njegove operacije `startTransfer`. Potom se blokira na semaforu `semComplete` koji drajver uređaja treba da signalizira kada zahtev bude opslužen, a onda dalje obaveštava podnosioca zahteva (izostavljeni detalji).

```
void diskDriver (RequestQueue* rque, IBlockDeviceDriver* dd) {
  while (true) {
    DiskRequest* req = rque->getRequest();// Blocks until a request arrives
    dd->startTransfer(req);
    semComplete->wait();
    ... // Notify the request initiator
  }
}
```

Interfejs drajvera uređaja `IBlockDeviceDriver` dat je dole. Kada se drajver instalira, jezgro poziva njegovu operaciju `init` za inicijalizaciju samog drajvera, zadajući semafor koji treba signalizirati nakon svakog opsluženog zahteva (`semComplete` pomenut gore).

```
class IBlockDeviceDriver {
public:
  virtual int  init (Semaphore* complete) = 0;
  virtual void startTransfer (DiskRequest*) = 0;
};
```

Tokom inicijalizacije u svojoj operaciji `init` drajver može da poziva sledeće operacije jezgra:

- `uint32* requestDMAChannel()`: zahteva od jezgra zauzimanje jednog slobodnog DMA kontrolera (kanala) za svoje potrebe; jezgro dodeljuje takav kontroler na upotrebu drajveru, ukoliko ga ima, i vraća adresu početka regiona memorijski mapiranih upravljačkih registara dodeljenog DMA kontrolera; ukoliko slobodnog DMA kontrolera nema, vraća *null*;
- `uint32 requestIVTEntry(void (*transferComplete)(void*),void* ptr))`: od jezgra traži zauzeće i inicijalizaciju jednog slobodnog ulaza u IVT; ukoliko ne uspe, vraća negativnu vrednost; ukoliko uspe, inicijalizuje taj ulaz u IVT adresom funkcije zadate prvim parametrom i vraća taj broj ulaza; jezgro obezbeđuje da se prekidna rutina poziva sa parametrom `ptr` (prekidna rutina na koju je usmeren jedan ulaz u IVT poziva se uvek sa istim tim parametrom).

Ukoliko inicijalizacija ne uspe, funkcija `init` treba da vrati negativnu vrednost, 0 za uspeh.
Svaki DMA kontroler obavlja prenos sa samo jednim diskom i ima dva 32-bitna upravljačka registra u svom regionu memorijski mapiranih registara. Prvi registar nalazi se na pomeraju nula od početka regiona, a prilikom inicijalizacije DMA kontrolera u ovaj registar potrebno je

upisati broj ulaza u IVT koji će ovaj DMA kontroler koristiti prilikom prekida koji generiše kada završi svaku zadatu operaciju. Drugi registar nalazi se na pomeraju 4 (adresibilna jedinica je bajt, adrese su 32-bitne) od početka regiona, a jedna operacija prenosa DMA kontroleru se zadaje sukcesivnim upisima u ovaj isti upravljački registar na sledeći način (nije potrebno čekati na dozvolu za sledeći upis, ovi upisi mogu da idu odmah jedan iza drugog): najpre je u ovaj registar potrebno upisati adresu bafera, zatim broj prvog bloka na disku, zatim broj susednih blokova koje treba preneti i konačno smer prenosa (0 ili 1). Ovaj poslednji upis ujedno i pokreće prenos. Po završetku prenosa DMA kontroler generiše prekid.

Realizovati u potpunosti klasu `DiskDeviceDriver` koja implementira interfejs `IBlockDeviceDriver`, a koja realizuje drajver za prenos sa disk uređajem korišćenjem DMA kontrolera. Jedan objekat ove klase treba da bude zadužen za jedan fizički disk uređaj.

Rešenje:

2. (10 poena) Fajl sistem

U sistemima nalik sistemu Unix cevovod (*pipe*) se formira sistemskim pozivom *pipe*:

```
#include <unistd.h>
int pipe (int pipefd[2]);
```

Ovaj sistemski poziv pravi cevovod i u `pipefd[0]` upisuje deskriptor fajla (*file descriptor*) odredišne strane otvorenog cevovoda (strana za čitanje), a u `pipefd[1]` upisuje desktriptor fajla izvorišne strane tog cevovoda (strana za upis). U slučaju greške, ovaj poziv vraća negativnu vrednost, a u slučaju uspeha vraća 0. Dati su i sistemski pozivi za čitanje i upis:

```
ssize_t read  (int fd, void *buf, size_t count);
ssize_t write (int fd, void *buf, size_t count);
```

Ovi pozivi prenose najviše `count` znakova sa fajlom koga identifikuje `fd` iz bafera ili u bafer na koga ukazuje `buf`, a vraćaju stvarno prenesen broj znakova.

Korišćenjem ovih, kao i drugih potrebnih sistemskih poziva i bibliotečnih funkcija (*fork*, *exit*, *close*, *putchar*, *fprintf*), napisati program koji, kada se nad njim pokrene proces, formira jedan cevovod kojim komunicira sa procesom detetom koga pokreće. Nakon toga, proces roditelj kroz cevovod procesu detetu šalje niz znakova dat svojim argumentom komandne linije `argv[1]` i završen znakom `'\0'`, a proces dete dobijeni niz znakova ispisuje na svoj standardni izlaz. Nakon što uradi navedeno, svaki od procesa se gasi. Greške obraditi ispisom na `stderr` i gašenjem procesa.

Rešenje:

3. (10 poena) Fajl sistem

U implementaciji nekog FAT fajl sistema ceo FAT keširan je u memoriji u nizu `fat`:

```
extern uint32 fat[];
extern uint32 freeHead, freeCount;
```

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 ulančani u jednostruku listu čija glava je u promenljivoj `freeHead`, dok ukupan broj slobodnih blokova čuva promenljiva `freeCount`. Implementirati sledeću funkciju koja treba da obriše sadržaj datog fajla:

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

Rešenje:

Rešenja zadataka za
treći kolokvijum iz Operativnih sistema 1
Jun 2023.

1. (10 poena)

```
class DiskDeviceDriver : public IBlockDeviceDriver {
public:
  virtual int  init (Semaphore* complete);
  virtual void startTransfer (DiskRequest*);
private:
  uint32* dmaCtrlRegion;
  Semaphore* semComplete;
  friend void transferComplete (void*);
};

int DiskDeviceDriver::init (Semaphore* sc) {
  semComplete = sc;
  dmaCtrlRegion = requestDMAChannel();
  if (!dmaCtrlRegion) return -1;
  uint32 ivte = requestIVTEntry(transferComplete,this);
  if (ivte<0) return -1;
  *(dmaCtrlRegion+0) = ivte;
  return 0;
}

void DiskDeviceDriver::startTransfer (DiskRequest* req) {
  *(dmaCtrlRegion+1) = req->buffer;
  *(dmaCtrlRegion+1) = req->startBlockNo;
  *(dmaCtrlRegion+1) = req->blockCount;
  *(dmaCtrlRegion+1) = req->dir;
}

interrupt void transferComplete (void* obj) {
  ((DiskDeviceDriver*)obj)->semComplete->signal();
}
```

2. (10 poena)

```
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>

#define handle_error(msg) \
  do { fprintf(stderr,"Error: %s\n",msg); exit(-1); } while(0)

int main (int argc, const char* argv[]) {
  if (argc!=2) handle_error("Missing argument.");

  int pipefd[2];
  if (pipe(pipefd)<0) handle_error("Cannot create a pipe.");

  pid_t pid = fork();
  if (pid<0) handle_error("Cannot create a child process.");

  if (pid>0) {

    write(pipefd[1],argv[1],strlen(argv[1]));

  } else {

    char c;
    while (read(pipefd[0],&c,1)>0)
      putchar(c);

  }

  close(pipefd[0]);
  close(pipefd[1]);
  exit(0);
}
```

3. (10 poena)

```
void truncate (FCB* fcb) {
  if (fcb==0) return;
  if (!fcb->head) return;  // File already empty

  uint32 last = fcb->head, cnt = 1;
  while (fat[last])
    last = fat[last], cnt++;
  fat[last] = freeHead;
  freeHead = fcb->head;
  freeCount += cnt;
  fcb->head = 0;
  fcb->size = 0;
```

Elektrotehnički fakultet u Beogradu
Katedra za računarsku tehniku i informatiku

Predmet: Operativni sistemi 1
Nastavnik: prof. dr Dragan Milićev
Odsek: Softversko inženjerstvo, Računarska tehnika i informatika
Kolokvijum: Treći, jun 2022.
Datum: 12. 6. 2022.

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

Date su deklaracije pokazivača preko kojih se može pristupiti registrima kontrolera tastature preko koga stižu znakovi otkucani na tastaturi:

```
typedef volatile unsigned REG;
REG* ioStatus =...;     // status register
char* ioData =...;      // data register
```

Kontroler tastature poseduje interni memorijski modul koji služi za prihvatanje jednog ili više znakova otkucanih na tastaturi (hardverski bafer). Kada se ovom baferu pojave znakovi (jedan ili više), kontroler generiše prekid. Tada se iz registra za podatke mogu čitati pristigli znakovi sukcesivnim operacijama čitanja, jedan po jedan, sve dok je bit spremnosti (*ready*) u razredu 0 statusnog registra postavljen na 1. Naredni nalet pristiglih znakova će ponovo generisati prekid.

Za smeštanje znakova učitanih sa tastature kernel koristi svoj (softverski) ograničeni bafer veličine 256 znakova. U ovaj bafer upisuju se znakovi učitani sa kontrolera tastature sve dok u njemu ima mesta; znakovi koji ne mogu da stanu u bafer se jednostavno odbacuju. Iz ovog bafera znakove uzimaju različiti uporedni tokovi kontrole (procesi) pozivom operacije `getc`; ukoliko u baferu nema znakova, pozivajući tok kontrole treba da se suspenduje dok znakova ne bude. Sinhronizacija se može obavljati semaforima čiji je interfejs isti kao u školskom jezgru. Pretpostaviti sledeće:

- prekidna rutina izvršava se međusobno isključivo sa operacijama na semaforu;
- operacija `signal` na semaforu može se pozivati iz prekidne rutine, jer ona ne radi nikakvu promenu konteksta;
- procesor maskira prekide pri obradi prekida; prekid sa kontrolera tastature se može eksplicitno maskirati pozivom `kbint_mask()`, a demaskirati pozivom `kbint_unmask()`.

Implementirati opisani podsistem: bafer kernela (operaciju `getc` i sve druge potrebne operacije) i prekidnu rutinu kontrolera tastature.

Rešenje:

2. (10 poena) Fajl sistem

Neprazno binarno stablo čiji su čvorovi tipa `Node` zapisano je u binarni fajl na sledeći način (prikazan je pojednostavljen kod bez obrade grešaka): za svaki čvor najpre je zapisan njegov sadržaj tipa `NodeData`, zatim jedan `int` indikator koji kaže da li taj čvor ima svoje levo podstablo, zatim jedan `int` indikator koji kaže da li taj čvor ima svoje desno podstablo, a onda isto tako redom celo levo podstablo ako ga ima, pa celo desno podstablo ako ga ima. Korišćenjem istog POSIX API za fajlove napisati kod funkcije `readTree` za učitavanje i izgradnju stabla iz takvog fajla. Prvi parametar je staza do fajla, a drugi parametar je adresa pokazivača u koji treba upisati adresu korenog čvora formiranog i učitanog stabla. Obraditi greške vraćanjem negativne vrednosti. POSIX API funkcija

```
ssize_t read(int fd, void *buf, size_t count);
```

vraća broj stvarno učitanih bajtova (može biti manji od zahtevanog ako se stigne do kraja fajla) ili negativnu vrednost u slučaju greške.

```
#include <fcntl.h>

struct Node {
  NodeData data;
  Node *left, *right;
};

void writeSubtree (int fd, Node* node) {
  write(fd,node->data,sizeof(NodeData));
  int ind = (node->left!=0);
  write(fd,ind,sizeof(int));
  ind = (node->right!=0);
  write(fd,ind,sizeof(int));

  if (node->left) writeSubtree(fd, node->left);
  if (node->right) writeSubtree(fd, node->right);
}

void writeTree (const char* pathname, Node* root) {
 int fd = open(*pathname,O_WRONLY|O_CREATE|O_TRUNC);
 int r = writeSubtree(fd,root);
 close(fd);
}

int readTree (const char* pathname, Node** root);
```

Rešenje:

3. (10 poena) Fajl sistem

Sistemski pozivi za pristup sadržaju binarnog fajla uobičajeno omogućavaju čitanje ili upis *size* susednih bajtova počev od pozicije *offset* u odnosu na početak sadržaja fajla (prvi bajt je na poziciji 0). Da bi ostvario ovakav pristup, neki kernel koristi klasu `FLogicalAccess` čiji je interfejs dat dole. Ova klasa koristi se tako što se za svaki ovakav pristup instancira objekat ove klase, inicijalizuje se pozivom operacije `reset` sa zadatim parametrima, a onda se iterira sve dok operacija `end` ne vrati 1. U svakoj iteraciji se pristupa odgovarajućim bajtovima jednog logičkog bloka fajla, tako što se može dobiti sledeća informacija:

- `getBlock`: vraća redni broj logičkog bloka u kom se pristupa u toj iteraciji;
- `getRelOffset`: vraća redni broj bajta u tekućem bloku počev od kog se pristupa;
- `getRelSize`: broj bajtova u tekućem bloku kojima se pristupa u toj iteraciji.

```
class FLogicalAccess {
public:

 FLogicalAccess ();
 void reset (size_t offset, size_t size);
 int  end() const;
 void next();

 size_t getBlock() const;
 size_t getRelOffset() const;
 size_t getRelSize() const;
};
```

Primer upotrebe ove klase je sledeći:

```
FLogicalAccess fla;
for (fla.reset(offset,size); !fla.end(); fla.next()) {
  // Access fla.getRelSize() bytes
  // starting from the offset fla.getRelOffset()
  // in the logical block fla.getBlock()
}
```

Na primer, ako je veličina bloka 8 bajtova, a inicijalno je zadato: *offset* = 11 i *size* = 15, onda će u prvoj iteraciji biti: `getBlock()`=1, `getRelOffset()`=3, `getRelSize()`=5, u drugoj će ove vrednosti redom biti 2, 0, 8, a u trećoj 3, 0, 2; posle toga će se izaći iz petlje zbog `end()`=1.

Implementirati u potpunosti klasu `FLogicalAccess`. Veličina bloka je `BLK_SIZE`.

Rešenje:

Rešenja zadataka za
treći kolokvijum iz Operativnih sistema 1
Jun 2022.

1. (10 poena)

```
class KeyboardBuffer {
public:
  KeyboardBuffer() : head(0),tail(0),count(0),mutex(1),itemAvailable(0) {}
  char getc ();
  void putc (char c);

private:
  static const size_t KB_SIZE = 256;
  char buffer[KB_SIZE];
  size_t head, tail, count;
  Semaphore mutex, itemAvailable;
};

static KeyboardBuffer* instance () {
  static KeyboardBuffer _instance;
  return &_instance;
}

char KeyboardBuffer::getc () {
  itemAvailable.wait();
  mutex.wait();
  kbint_mask();
  char c = buffer[head];
  head = (head+1)%KB_SIZE;
  count--;
  kbint_unmask();
  mutex.signal();
  return c;
}

void KeyboardBuffer::putc (char c) {
  if (count<KB_SIZE) {
    buffer[tail] = c;
    tail = (tail+1)%KB_SIZE;
    count++;
    itemAvailable.signal();
  }
}

interrupt void keyb_int () {
  while (*ioStatus & 1)
    KeyboardBuffer::instance()->putc(*ioData);
}
```

2. (10 poena)

```
#include <fcntl.h>

int readSubtree (int fd, Node** node) {
  static NodeData data;
  static int left, right;
```

```
  ssize_t cnt = read(fd,&data,sizeof(NodeData));
  if (cnt<sizeof(NodeData)) return -2;
  cnt = read(fd,&left,sizeof(int));
  if (cnt<sizeof(int)) return -2;
  cnt = read(fd,&right,sizeof(int));
  if (cnt<sizeof(int)) return -2;

  *node = new Node();
  if (*node == 0) return -3;
  (*node)->data = data;
  (*node)->left = (*node)->right = 0;

  int r = 0;
  if (left) {
    r = readSubtree(fd,&((*node)->left));
    if (r<0) return r;
  }
  if (right) {
    r = readSubtree(fd,&((*node)->right));
    if (r<0) return r;
  }
  return 0;
}

int readTree (const char* pathname, Node** root) {
 int fd = open(*pathname,O_RDONLY);
 if (fd<0) return -1;

 int r = readSubtree(fd,root);

 close(fd);
 return r;
}
```

3. (10 poena)

```
class FLogicalAccess {
public:

  FLogicalAccess () { reset(0,0); }
  void reset (size_t offset, size_t size);
  int  end() const { return isEnd; }
  void next();

  size_t getBlock() const { return blk; }
  size_t getRelOffset() const { return relOfs; }
  size_t getRelSize() const { return relSz; }

private:
  size_t size;
  size_t blk, relOfs, relSz;
  int isEnd;
};

inline void FLogicalAccess::reset (size_t offset, size_t sz) {
    blk = offset/BLK_SIZE;
    relOfs = offset%BLK_SIZE;
    relSz = (sz>BLK_SIZE-relOfs)?(BLK_SIZE-relOfs):sz;
    isEnd = (sz==0);
    size -= relSz;
}
```

```
inline void FLogicalAccess::next () {
  if (size>0) {
    blk++;
    relOfs = 0;
    relSz = (size>BLK_SIZE)?BLK_SIZE:size;
    isEnd = 0;
    size -= relSz;
  }
  else reset(0,0);
```

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:

Treći kolokvijum iz Operativnih sistema 1
Avgust 2023.

1. (10 poena)

U klasu `BlockIOCache` treba uvesti privatne podatke članove `numOfBlocks` i `hand`, oba tipa `int` i inicijalizovana na 0.

```
int BlockIOCache::getFreeEntry () {
  if (numOfBlocks<CACHESIZE) return numOfBlocks++;
  int oldHand = hand;
  do {
    if (entries[hand].refCounter == 0) {
      ioWrite(dev,entries[hand].blkNo,entries[hand].buf);
      int ret = hand;
      hand = (hand+1)%CACHESIZE;
      return ret;
    }
    hand = (hand+1)%CACHESIZE;
  } while (hand!=oldHand);
  return -1; // Cannot find a block to evict
}
```

2. (10 poena)

```
cat > a/doc
ln a/doc b/hdoc
ln -s b/hdoc c/sdoc
rm a/doc
```

Komanda `cat b/hdoc` će ispisati uneti sadržaj učitan sa tastature i upisan u fajl u prvoj komandi.
Nakon komande `rm b/hdoc`, komanda `cat c/sdoc` će ispisati grešku (fajl ne postoji).

3. (10 poena)

```
Node* Node::getNodeAbs (const char* pStart, const char* pEnd) {
  const char *pE = pEnd;
  while (pE>pStart) {
    Node* node = DentryCache::getNode(pStart,pE);
    if (node) {
      if (pE<pEnd) node = Node::getNodeRel(pE+1,pEnd,node);
      if (node) DentryCache::store(pStart,pEnd,node);
      return node;
    }
    do { pE--; } while (pE>pStart && *pE!=delimiter);
  }
  Node* node = Node::getNodeRel(pStart+1,pEnd,Node::root);
  if (node) DentryCache::store(pStart,pEnd,node);
  return node;
}
```