B

U/I nizi nivo

C programeri u GNU/Linux imaju na raspolaganju dva skupa ULAZNO/IZLAZNIH funkcija. Standardna C biblioteka obezbedjuje U/I funkcije: printf, fopen, itd.[1] Linux-ovo jezgro samo za sebe obezbedjuje drugi skup U/I operacija koje funkcionisu na nizem nivou nego C funkcije.

Posto je ova knjiga namenjena za ljude koji vec znaju jezik C, mi pretpostavljamo da ste se vec sreli i da znate da koristite U/I funkcije C biblioteka.

Cesto postoje dobri razlozi za upotrebu Linux-ovih U/I funkcija nizeg nivoa. Mnoge od njih su sistemski pozivi jezgra[2] i one najvecim delom omogucavaju direktan pristup u osnovne mogucnosti sistema koji su raspolozivi za namenske programe. Ustvari standardne U/I rutine C biblioteka su implementirane u vrh nizeg nivoa U/I sistemskih poziva Linux-a. Koriscenje poslednjeg je uglavnom najefikasniji nacin da se izvedu ulazne i izlazne operacije- i ponekad je pogodniji.

Kroz ovu knjigu mi pretpostavljamo da ste upuceni sa pozivima opisanim u ovom dodatku. Mozete takodje biti upoznati sa njima zato sto su skoro isti kao i oni obezbedjenii u ostalim UNIX i kao UNIX operativnim sistemima(kao I Win32 platformama) Iako niste upuceni, ipak procitajte; videcete da je ostatak knjige mnogo laksi za razumevanje ako se prvo upoznate sa ovom materijom.

B.1 Citanje i pisanje podataka

Prva U/I funkcija sa kojom ste se susreli kada ste poceli da ucite jezik C je verovatno printf. Ona formatira tekstualni niz(string) i onda ga ispisuje na standardni izlaz. Uopstena verzija , fprintf, moze da ispise tekst u datoteku (stream) drugaciju nego standardni izlaz. Datoteka (stream) je predstavljena kao FILE* pokazivac. Kada otvorite datoteku sa fopen vi dobijate FILE* pokazivac. Kada zavrsite mozete je zatvoriti sa fclose. Pored fprint, mozete koristiti funkcije kao sto su fputc, fputs i fwrite koje upisuju podatke u datoteku (stream) ili fscanf, fgetc, fgets i fread koje ucitavaju podatke.

Sa Linux-ovim operacijama nizeg nivoa, koristimo postupak nazvan deskriptor datoteke umesto FILE* pokazivaca. Deskriptor datoteke je celobrojna vrednost koja upucuje na odredjenu datoteku u pojedinacnom procedu. Moze biti otvoren za citanje, za pisanje ili i za citanje i za pisanje. Deskriptor datoteke ne mora da upucuje na otvorenu datoteku; moze predstavljati vezu sa drugom sistemskom komponentom koja moze da prima i salje podatke. Na primer veza izmedju uredjaja je predstavljena pomocu deskriptora datoteke (pogledati poglavlje 6, “Uredjaji”), kao otvorena uticnica (pogledati poglavlje 5, “Interprocesna komunikacija”, sekciju 5.5, “Uticnice”) ili kao jedan kraj cevovoda(pipe) (videti sekciju 5.4,”Cevovodi(pipes)”).

Ukljucite zaglavlja <fcntl.h>, <sys/types.h>, <sys/stat.h>, i <unistd.h> ako koristite U/I

funkcije nizeg nivoa koje su opisane ovde.

B.1.1 Otvaranje datoteke

Da bi otvorili datoteku i naveli deskriptor datoteke da moze da pristupi toj datoteci koristite funkciju open. Ona uzima kao argumente ime putanje datoteke koja treba da se otvori, kao slovni niz, i indikator koji opisuje kako da je otvori. Mozete koristiti open da kreirate novi datoteku; ako koristite, onda dodajte argumente koji opisuju koja ce pristupna prava imati nova datoteka.

Ako je drugi argument O_RDONLY, datoteka je otvorena samo za citanje; ako budete pokusali da upisujete u nju pojavice se greska. Slicno O_WRONLY stvara da bude samo za upis. Naznacujuci O_RDWR deskriptor datoteke se navodi da bude upotrebljena i za upis i za citanje. Znajte da ne mogu sve datoteke datoteke biti otvorene na sva tri nacina. Na primer, dozvole nad datotekom mogu da zabrane odredjenom procesu da je otvori za citanje ili za upis; datoteka na uredjaju koji je samo za citanje kao CD-ROM ne moze biti otvorena za upis.

Mozete navesti dodatne funkcije koriscenjem bita operacija ili sa ovom vrednoscu sa jednim ili vise indikatora. Ovo su najcesce koriscene vrednosti:

- Navedite O_TRUNC da odbacite otvorenu datoteku, ako prethodno postoji. Podatci upisani u deskriptor datoteke ce zameniti prethodni sadrzaj datoteke.
- Navedite O_APPEND da dodate u postojecu datoteku. Podatci upisani u deskriptor datoteke ce biti dodati na kraj datoteke.

- Navedite O_CREAT da kreirate novu datoteku. Ako ime datoteke koje navedete ne postoji, nova datoteka ce biti kreirana, pod pretpostavkom da postoji direktorijum koji sadrzi datoteku i da proces ima prava da kreira datoteku u tom direktorijumu. Ako datoteka vec postoji, onda ce se samo otvoriti.
- Navedite O_EXCL sa O_CREAT da nasilno kreirate novu datoteku. Ako datoteka vec postoji, poziv open ce biti bezuspesan.

Ako pozovete open sa O_CREAT, dodajte treci dodatni argument zadajuci dozvole za novu datoteku. Pogledajte poglavlje 10,"Bezbednost", sekciju 10.3, "Pristupna prava sistemskih datoteka", za opis bitova ovlascenja i kako da ih upotrebljavate.

Na primer program u Listing B.1 kreira novu datoteku sa imenom zadatim u komandnoj liniji. Koristi O_EXCL indikator sa open, pa ako datoteka vec postoji, dolazi do greske. Novoj datoteci su data prava citanja i pisanja, vlasniku i grupi vlasnika, i pravo citanja ostalima. (ako je umask podesen na ne nulte vrednosti, aktuelne dozvole mogu biti restriktivnije).

Listing B.1 (create-file.c) Kreiranje nove datoteke

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
```

```
#include <unistd.h>

int main (int argc, char* argv[])
{
    /* Putanja gde ce se kreirati nova datoteka. */
    char* path = argv[1];
    /* Dozvole za novu datoteku. */
    mode_t mode = S_IRUSR | S_IWUSR | S_IRGRP | S_IWGRP | S_IROTH;

    /* Kreira datoteku. */
    int fd = open (path, O_WRONLY | O_EXCL | O_CREAT, mode);
    if (fd == -1) {
        /* Dolazi do greske. Ispisuje se poruka o gresci. */
        perror ("open");
        return 1;
    }

    return 0;
}
```

Evo ga program u akciji:

```
% ./create-file testfile
% ls -l testfile
-rw-rw-r--  1    samuel    users   0 Feb  1 22:47 testfile
% ./create-file testfile
open: File exists
```

Primetite da je velicina nove datoteke 0 zato sto program nije upisao nikakve podatke u nju.

B.1.2 Zatvaranje deskriptora datoteke

Kada zavrsite sa deskriptorom datoteke, zatvorite ga sa close. U nekim slucajevima kao program u Listing B.1, nije potrebno da se eksplicitno pozove close zato sto Linux zatvara sve otvorene deskriptore datoteka kada se proces prekine (to je kada se program zavrsi). Naravno, jednom kad se zatvori deskriptor datoteke, ne bi trebalo vise da ga koristimo.

Zatvaranje deskriptora datoteke moze prouzrokovati da Linux uradi odredjene akcije u zavisnosti od prirode deskriptora datoteke. Na primer, kada zatvorite deskriptor datoteke za mrezni prikljucak, Linux zatvara mreznu vezu izmedju dva kompjutera koji komuniciraju preko tog prikljucka.

Linux ogranicava broj otvorenih deskriptora datoteka da proces moze da se otvori na vrema. Otvoreni deskriptor datoteke koristi resurse jezgra, tako da je dobro da se zatvori deskriptor datoteke kada se zavrsi sa njim. Tipican limit 1024 deskriptora datoteka po procesu. Mozete podesiti taj limit sa specijalnim sistemskim pozivom setrlimit; videti sekciju 8.5,"getrlimit i setrlimit: Limiti resursa", za vise detalja.

B.1.3 Upisivanje podataka

Upisite podatke u deskriptor datoteke koristeci poziv write. Obezbedite deskriptoru datoteke, pokazivac na bafer podataka, i broj bajtova za upis. Deskriptor datoteke mora biti otvoren za upis.

Podatci upisani u datoteku ne moraju biti string; write kopira proizvoljne bajtove iz bafera u deskriptor datoteke.

Program u Listing B.2 dodaje tekuce vreme u datu datoteku na komandnu liniju. Ako datoteka ne postoji, kreirace se. Ovaj program takodje koristi time, localtime i asctime funkcije za dobijanje i formatiranje tekuceg vremena.

Listing B.2 (timestamp.c) Dodavanje vremenske oznake datoteci:

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <time.h>
#include <unistd.h>
/* Vraca string koji predstavlja tekuci datum i vreme. */
char* get_timestamp ()
{
  time_t now = time (NULL);
  return asctime (localtime (&now));
}
int main (int argc, char* argv[])
{
  /* Datoteka u koju treba da se doda vremenska oznaka. */
  char* filename = argv[1];
  /* Uzima tekucu vremensku oznaku */
  char* timestamp = get_timestamp ();
  /* Otvara datoteku za upis. Ako postoji, pristupa joj. */
      int fd = open (filename, O_WRONLY | O_CREAT | O_APPEND, 0666);
  /* Racuna duzinu timestamp stringa. */
  size_t length = strlen (timestamp);
  /* Upisuje timestamp u datoteku. */
  write (fd, timestamp, length);
  /* Sve gotovo. */
  close (fd);
  return 0;
}
```

Evo kako timestamp program radi:

```
% ./timestamp tsfile
% cat tsfile
Thu Feb  1 23:25:20 2001
% ./timestamp tsfile
% cat tsfile
Thu Feb  1 23:25:20 2001
Thu Feb  1 23:25:47 2001
```

Zapamtite da prvi put kad pozovemo timestamp, on kreira takozvain tsfile, dok kad drugi put

pozovemo taj program on mu samo dodaje.

–

B.1.4 Citanje podataka

Odgovarajuca komanda za citanje podataka je read. Kao i write, on pravi deskriptor datoteke, pokazivac ka baferu, i brojac. Brojac broji koliko je bajtova procitano iz deskriptora datoteke u bafer. Komada read vraca – 1 ako je greska u pitanju ili onoliko bajtova koliko je u stvari procitao. Taj broj moze biti manji od broja bajtova koji su zahtevani, na primer, ako nema vise bajtova u datoteci.

> Citanje DOS/Windows tekstualnih datoteka
> Nakon citanja ove knjige, sigurni smo da cete uzeti i napisati sve vase programe za GNU/Linux. Bilo kako bilo, vas program ce verovatno ponekad trebati da procita neki DOS ili Windows program. Bitno je primeiti jednu vaznu razliku u strukturi datoteka ove dve platforme.
>
> U GNU/Linux tekstualnim datotekama, svaki red je odvojen od sledeceg sa karakterom za novi red. Karakter za novi red je konstanta "\n", koja ima ASCII kode 10. U Windows-u, redovi su separatisani sa kombinacijom 2 karaktera: CR karakter (karakterakter je "\r", ciji je ASCII kod 13), koji je propracen sa karakterom za novi red.
>
> Neki GNU/Linux tekt editori prikazuju ^M na kraju svakog reda kad Windows textualne datoteke – to je CR karakter. Emacs prikazuju Windows tekstualne datoteke pravilno ali oznacava ih pokazivanjem (DOS) u statusnoj (mode) liniji na dnu bafera.Neki Windows editori, kao na primer Notepad, prikazuju sav text u GNU/Linux text datoteci u jednom redu jer ocekuje CR karakter na kraju svakog reda. Drugi programi koji su i za GNU/Linux i Windows i rade sa textom mogu da prijave dosta gresaka kada im se ubaci pogresan format texta.
>
> Ako vas program cita tekstualne datoteke napravljene sa strane Windows programa, verovatno cete zeleti da promenite sekvencu "/r/n" sa jednim novim redom. Slicno tako, ako vas program pise textualne datoteke koje trebaju da budu procitane sa strane Windows programa, zamenite karakter za novi red sa "/r/n" kombinacijom. Morate ovo uraditi bilo da koristite nizi nivo U/I funkcija koji su objasnjeni u ovom dodatku bilo da koristite standardne C U/I funkcije.

Listing B.4 prikazuje prostu demonstraciju funkcije read. Program ispisuje heksadecimalne ostatke sadrzaja datoteke preciziranog na comandnoj liniji. Svaki red prikazuje pomeraj u datoteci i sledecih 16 bajtova.

Listing B.4 (hexdump.c) Ispis heksadecimalnog ostatka datoteke

```
#include <fcntl.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
  unsigned char buffer[16];
  size_t offset = 0;
  size_t bytes_read;
```

```
    int i;

    /*Otvara datoteku za citanje*/
    int fd = open (argv[1], O_RDONLY);
    /*Cita iz datoteke, jedan po jedna karakter. Nastavlja dok citanje “ne postane kratko”,
      a to je, kada cita manje nego sto smo mi trazili. To pokazuje da smo stigli do kraja d
      atoteke.*/
    do {
            /*Citanje bafera. */
            bytes_read = read (fd, buffer, sizeof (buffer));
            /*Pisanje pomeraja u datoteku, pracen je sa samim bajtovima*/
            printf (“0x%06x : “, offset);
            for (i = 0; i < bytes_read; ++i)
                    printf (“%02x “, buffer[i]);
            printf (“\n”);
            /*Pamti nasu poziciju u datoteci */
            offset += bytes_read;
    }
    while (bytes_read == sizeof (buffer));
    /*Sve gotovo.*/
    return 0;
}
```

Evo ga hexdump u akciji. Prikazuje ispis ostataka izvrsne datoteke:

```
% ./hexdump hexdump
0x000000 : 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
0x000010 : 02 00 03 00 01 00 00 00 c0 83 04 08 34 00 00 00
0x000020 : e8 23 00 00 00 00 00 00 34 00 20 00 06 00 28 00
0x000030 : 1d 00 1a 00 06 00 00 00 34 00 00 00 34 80 04 08
...
```

Vas izlaz moze igledati drugacije, u zavisnoti od prevodioca koji koristite da prevedete hexdump i prevedenih(kompajliranih) indikatora (flags) za prevodjenje koje ste selektovali.

B.1.5 Kretanje po datoteci

Deskriptor datoteke pamti njegovu poziciju u datoteci. Dok citate iz njega ili pisete po njemu, njegova pozicija napreduje paralelno sa brojem bajtova koje citamo ili pisemo. Ponekad, kako bilo, moracete da se krecete po datoteci bez citanja ili pisanja podataka. Na primer, zelite da pisete po sredini datoteke bez menjanja sredine datoteke, ili zelite da se vratite na pocetak datoteke i ponovo da procitate nesto a da ne morate ponovo da otvarate datoteku i krecete iz pocetka.

Lseek funkcija vam omogucava da pozicionrate deskriptor datoteke u datoteci. Prosledite ga deskriptoru datoteke i dva dodatna argumenta odredjujuci novu poziciju.

- Ako je treci argument SEEK_SET, lseek tumaci drugi argument kao poziciju, u bajtovima, od pocetka datoteke.

- Ako je treci argument SEEK_CUR, lseek tumaci drugi argument kao pomeraj , koji moze biti pozitivan ili negativan, od trenutne pozicije.

- Ako je treci argument SEEK_END, lseek tumaci drugi argument kao pomeraj od kraja datoteke. Pozitivna vrednost pokazuje poziciju izvan kraja datoteke.

Funkcija lseek vraca novu poziciju, kao pomeraj od pocetka datoteke.
Vrsta pomeraja je off_t. Ako se pojavi greska, lseek vraca -1. lseek ne mozete koristit sa nekim vrstama deskriptora datoteka, kao sto je uticnica(soket) deskriptor datoteke.

Ako zelite da nadjete poziciju deskriptora datoteke u datoteci a da ga ne menjate, odredite pomeraj 0 od trenutne pozicije – na primer:

```
off_t position = lseek (file_descriptor, 0, SEEK_CUR);
```

Linux vam omogucava da lseek funkcijom pozicionirate deskriptor datoteke izvan datoteke. Normalno, ako je deskriptor datoteke pozicioniran na kraju datoteke i vi nesto upisujete u njega, Linux ce automatski da produzi datoteku kako bi napravio mesta za nove podatke. Ako pozicionirate deskriptor datoteke izvan datoteke i onda nesto upisujete u njega, Linux prvo prosiruje datoteku kako bi prilagodio rupu koju ste napravili sa lseek operacijom i onda upisuje na kraj datoteke. Ova praznina, kako bilo, ne zauzima mesto na disku; naprotiv, Linux samo pravi poruku koliko je to veliko. Ako kasnije pokusate da procitate do iz datoteke, vasem programu ce izgledati kao da je ta praznina popunjena sa 0 bajtovima.

Koriscenjem ovog ponasanja lseek, moguce je kreirati veoma velike datoteke koje ne zauzimaju skoro nista memorije na disku. Program lseek-huge u Listing B.5 radi to. On uzima kao komandnu liniju ime datoteke i velicinu trazene datoteke u MB. Program otvara novu datoteku, prosedje do kraja datoteke koristeci lseek, i onda upise jedinstveni 0 bajt pre nego sto ga zatvori.

Listing B.5 (lseek-huge.c) Kreiranje velike datoteke sa **lseek**.

```
#include <fcntl.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
    int zero = 0;
    const int megabyte = 1024 * 1024;

    char* filename = argv[1];
    size_t length = (size_t) atoi (argv[2]) * megabyte;

    /*Otvaranje nove datoteke */
    int fd = open (filename, O_WRONLY | O_CREAT | O_EXCL, 0666);
    /* Prelazimo na 1 bajt manje od mesta gde hocemo da bude kraj datoteke. */
```

```
        lseek (fd, length - 1, SEEK_SET);
        /* Pisemo jedinstven 0 bajt*/
        write (fd, &zero, 1);
        /*Sve gotovo.*/
        close (fd);

        return 0;
}
```

Koriscenjem lseek-huge, napravicemo datoteku od 1 GB(1024MB). Proverite slobodno mesto na disku pre i posle operacije.

```
% df -h .
Filesystem  Size  Used  Avail  Use%  Mounted on
/dev/hda5  2.9G  2.1G  655M 76%  /
% ./lseek-huge bigfile 1024
% ls -l bigfile
-rw-r-----  1  samuel  samuel  1073741824  Feb  5  16:29  bigfile
% df -h .
Filesystem  Size Used  Avail Use% Mounted on
/dev/hda5 2.9G 2.1G  655M 76% /
```

Neprimetna kolicina memorije na disku je potrosena, uprkos nenormalnoj velicini bigfile. Opet, iako otvorimo bigfile i citamo iz njega, on ce izgledati kao da je popunjen sa nulama u vrednosti od 1GB . Za primer, mozemo da testiramo njegov sadrzaj sa hexdump programom iz Listing B.4.

```
% ./hexdump bigfile | head -10
0x000000 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000010 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000020 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000030 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000040 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x000050 : 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
...
```

Ako ovo sami startujete, verovatno cete hteti da ga ubijete sa Ctrl+C, radije nego da gledate kako on ispisuje preko $2_{30}0$ bajtova.

Zapamtite da su ove magicne praznine u datotekama specijalne karakteristike ext2 sistem datoteka koji je tipicno koriscen za GNU/Linux diskove. Ako pokusate da koristite lseek-huge da kreirate datoteke na nekim drugim tipovima sistema datoteka, kao sto je fat ili vfat sistem datoteka koji su korisceni kao postavka za DOS i Windwos particije, otkricete da datoteka koju dobijete kao rezultat ustvari stvarno popunjava odredjenu kolicinu memorije.

Linux vam ne dozvoljava da premotavate sa lseek pre pocetka datoteke.

B.2 **stat**

Koriscenjem open i read, mozete videti sadrzaj datoteke. Ali sta je sa ostalim informacijama? Na primer, pozivanje ls -l prikazuje, za datoteke u aktivnom direktorijumu, informacije kao sto su velicina datoteke, poslednje promene u datoteci, dozvole, i vlasnika datoteke.

Stat funkcija nam daje te informacije o datoteci. Funkcija stat sa putanjom do datoteke koji vas zanima i pokazivac na promenljivu tipa struct stat. Ako stat odradi sve uspesno, onda vraca 0 i popunjava polja sa informacijama o datoteci; u suprotnom, ona vraca -1.

Ovo su najkoriscenija polja u struct stat:

- st_mode sadrzi pristupne dozvole te datoteke. Dozvole datoteka su objasnjene u Sekciji 10.3, "Prava pristupa za sistem datoteka"
- kao dodatak na pristupne dozvole, st_mode sifruje tip datoteke u visim bitovima. Pogledati tekst koji prati ovu listu za instrukcije za instrukcije desifrovanja informacije.
- st_uid i st_gid sadrze ID od korisnika i grupe, odnosno, kome pripada datoteka. Korisnicki i grupni ID-evi su opisani u Sekciji 10.1, "Korisnici i grupe"
- st_size sadrzi velicinu datoteke, u bajtovima.
- st_atime sadrzi vreme kada je poslednji put bilo pristupano toj datoteci (pisano ili citano)
- st_mtime sadrzi vreme kada je ta datoteka poslednji put bila promenjena.

Ovi makroi proveravaju vrednost st_mode da bi shvatili koju vrstu datoteke vas stat poziva. Makro ocenjuje da je to istina ako je datoteka odredjenog tipa.

S_ISBLK (mode) blokiranje uredjaja

S_ISCHR (mode) karakter uredjaj

S_ISDIR (mode) direktorijum

S_ISFIFO (mode) fifo (nazvana cev(pipe))

S_ISLNK (mode) simbolicki link

S_ISREG (mode) regularna datoteka

S_ISSOCK (mode) prikljucak

St_dev polje sadrzi broj glavnih i sporednih hardverskih uredjaja u kojima se ta datoteka nalazi. Brojevi uredjaja su objasnjeni u Poglavlju 6. Broj glavnih uredjaja je pomeren ulevo za 8 bitova; broj sporednih uredjaja obuhvata 8 najmanje znacajnih bitova. St_ino polje zauzima takozvani inode broj datoteke. On nalazi datoteku u sistemu datoteka.

Ako pozovete stat na simbolicki link, stat prati link i vi mozete dobiti informacije o datoteci na koji taj link pokazuje, ali ne i o njemu samom. Ovo podrazumeva da S_ISLINK nikada nece biti istinit (true) kao rezultat stat. Koristite lstat funkcije ako ne zelite da partite simbolicki link; ova funkcija dobija informacije o samom linku pre nego o njenom odredistu. Ako pozovete lstat u datoteci koja nije simbolicki link, onda je ona ista kao stat. Pozivanjem funkcije stat na prekinut link (link koji pokazuje na nepostojecu ili nepristupacno odrediste) kao rezultat se pojavi greska,

dok pozivanjem funkcije lstat ne prijavljuje gresku.

Ako vec imate otvorenu datoteku za citanje ili pisanje, pozovite fstat umesto stat. Ono uzima deskriptor datoteke kao njegov prvi argument umesto putanje.

B.3 Vektorsko citanje i pisanje

Write funkcija uzima kao argument pokazivac ka startu bafera podatka i duzinu tog bafera. On pise kontinualnu oblast memorije u deskriptor datoteke. Kako bilo, program ce ponekad trebati da zapise nekoliko stavki podatka, svaka se nalazi u razlicitim delovima memorije. Da bi koristili write program mora ili da kopira u mali deo memorije, koje jasno pravi neuspelo koriscenje CPU ciklusa i memorije, ili bi morao da napravi visestruki poziv ka write.

Za neke aplikacije, visestruki poziv ka write su neefikasni ili nepozeljni. Na primer, kada pisemo mreznom prikljucku, dve write funkcije mogu da pruzrokuju da dva paketa budu poslata kroz mrezu, s obzirom na cinjenicu da isti podatak moze biti poslat u jedinom paketu ako je moguc jedan poziv za write.

Writev funkcija omogucava vam da visestruki diskontinualne regione memorije upisete u deskriptor datoteke u jednoj operaciji. Ovo je nazvano vektorsko pisanje. Ono sto trebamo dodatno da uradimo prilikom koriscenja writev je to da moramo da podesimo strukturu podataka odredjivanjem starta i duzine svakog dela memorije. Ovaj strukturni podatak je niz elemenata strukture iovec. Svaki element specificira jedan deo memorije za pisanje; polja iov_base i iov_len odredjuju adresu starta dela memorije i njenu duzinu. Ako

unapred znate kolika oblast vam treba, mozete prosto da deklarisete red promenljivih strukture iovec; ako broj regija moze da varira, onda morate da dodelite dinamicki niz.

Funkcija writev daje deskriptoru datoteke da upisuje, niz structure iovec, i broj elemenata u nizu. Povratna vrednost je kompletan broj upisanih bajtova.

Program u Listing B.7 pise argumente sa komandne linije u datoteku koristeci jedan writev poziv. Prvi argument je ime datoteke; drugi i treci argument su napisani u datoteku sa tim imenom, svaki u jedan red. Program dodeljuje niz sa elementima structure iovec koji je duplo duzi od broja argumenata koja su napisana – za svaki argument on pise text svojih argumenata kao i karakter za novi red. Posto ne znamo unapred broj argumenata, niz je dodeljen koriscenjem malloc.

Listing B.7 (write-arg.c) Pisanje liste argumenata u datoteku koristeci **writev**

```
#include <fcntl.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>

int main (int argc, char* argv[])
{
        int fd;
        struct iovec* vec;
        struct iovec* vec_next;
```

```
    int i;
    /* Trebace nam “bafer” koji sadrzi karaktere za novi red. Koristite uobicajene char
       promenljive za to */
    char newline = ‘\n’;
    /* Prvi argument u komandnoj liniji je ime datoteke */
    char* filename = argv[1];
    /* Preskacemo prva 2 elementa sa liste argumenata. Element 0 je ime programa, a
       element 1 je ime datoteke */
    argc -= 2;
    argv += 2;

    /* Dodeljivanje niza elemenata iovec. Trebaju nam po dva za svaki element liste ar
       gumenata, jedan za sam text, a drugi za novi red */
    vec = (struct iovec*) malloc (2 * argc * sizeof (struct iovec));

    /* Petlja preko liste argumenata, pravljenje iovec upisa */
    vec_next = vec;
    for (i = 0; i < argc; ++i) {
     /*Prvi element je text samog argument*/
        vec_next->iov_base = argv[i];
        vec_next->iov_len = strlen (argv[i]);

        ++vec_next;
        /*Drugi red je jedan karakter za novi red. U redu je ako visestruki elementi iz ni
          za struct iovec pokazuju na isti deo memorije*/
        vec_next->iov_base = &newline;
        vec_next->iov_len = 1;
        ++vec_next;
    }

    /*Pisanje argumenata u datoteku*/
    fd = open (filename, O_WRONLY | O_CREAT);
    writev (fd, vec, 2 * argc);
    close (fd);

    free (vec);
    return 0;
}
```

Evo ga primer pokretanja write-args.

```
% ./write-args outputfile “first arg” “second arg” “third arg”
% cat outputfile
first arg
second arg
third arg
```

Linux snabdeva odgovarajucu funkciju readv koja u jednoj operaciji cita u visestruko diskontinualne regione memorije. Slicno kao writev, niz elemenata structure iovec odredjuje oblast memorije iz koje ce podaci biti procitani sa deskriptorom datoteke.

B.4 Veza sa standardnom C bibliotekom U/I funkcija

Pomenuli smo ranije da standardna C biblioteka U/I funkcija je implementirana na vrhu ovih U/I funkcija niskog nivoa. Ponekad je zgodno da se koriste standardne funkcije sa deskriptorima datoteke, ili da se koriste U/I funkcije niskog nivoa u standardnoj biblioteci FILE* datoteke(stream). GNU/Linux vam omogucuje da koristite oboje.

Ako ste otvorili datoteku koristeci fopen, mozete dobiti osnovni deskriptor datoteke koristeci fileno funkciju. On uzima argument FILE* i vraca deskriptor datoteke. Na primer, da otvorimo datoteku sa standardnom fopen funkcijom ali da upisujete u nju uz pomoc writev, mozete uz pomoc ovog koda:

```
FILE* stream = fopen (filename, “w”);
int file_descriptor = fileno (stream);
writev (file_descriptor, vector, vector_length);
```

Zapamtite da stream i file_descriptor odgovaraju istoj otvorenoj datoteci. Ako pozovete ovu liniju, vise ne mozete da upisijete u file_descriptor:

```
fclose (stream);
```

Slicno, Ako pozovete ovu liniju, vise ne mozete da upisujete u stream:

```
close (file_descriptor);
```

Da bi isli obrnuto, od deskriptora datoteke do datoteke (stream), koristi se fdopen funkcija. Ona spaja FILE* datoteku (stream) pokazivac sa odgovarajucim deskriptorom datoteke. Funkcija fdopen uzima argument deskriptora datoteke i string argument odredjujuci nacin u kom se kreira datoteku (stream). Sintaksa argumenta je ista kao i drugog argumenta fopen, i mora biti kompatibilan sa deskriptorom datoteke. Na primer, naznacite r za citanje deskriptora datoteke ili w za pisanje u deskriptor datoteke. Kao sa fileno, datoteka (stream) i deskriptor datoteke upucuju ka istoj otvarenoj datoteci, pa ako zatvorite jedan, sledeci necete moci da koristite.

B.5 Druge operacije sa datotekama

Par operacija nad datotekama i direktorijumima mogu da budu korisne:

- getcwd nam daje trenutni direktorijum u kome se radi. Ima 2 argumenta, char bafer i duzinu tog bafera. on kopira putanju trenutno koriscenog direktorijuma u bafer.

- chdir menja direktorijum u kojem se trenutno nalazite na putanju koju stavite kao njegov argument.

- mkdir kreira novi direktorijum. Njegov prvi argument je putanja gde ce se nalaziti taj novi direktorijum. Drugi argument su pristupne dozvole za tu novu datoteku. Interpretacija dozvola je ista kao i treci argument za otvaranje i modifikovane su uz pomoc procesa umask.

- rmdir brise direktorijum. Njegov argument je putanja do direktorijuma.

- unlink brise datoteku. Njegov argument je putanja do datoteke. Ova funkcija takodje moze biti koriscena za brisane objekata drugih sistema datoteka, kao imenovane cevovodi (pipes) (pogledati sekciju 5.4.5, “FIFOs”) ili uredjaje (Pogledati poglavlje 6).

U stvari, unlink ne mora da obrise sadrzaj datoteke. Kao sto njegovo ime govori, on raskida link datoteke i direktorijuma u kome se nalazi. Datoteka se vise ne ispisuje u tom direktorijumu kada ga izlistavano, ali ako bilo koji proces ima otvoren deskriptor datoteke koji ide prema datoteci, sadrzaj datoteke nije obrisan sa diska. Samo kada ni jedan process nema otvoren deskriptor datoteke onda je sadrzaj datoteke obrisan. Tako da, ako neki process otvori datoteku da bi ga citao ili pisao u nju i onda drugi proces uradi unlink nad tom datotekom i kreira novu datoteku sa istim imenom, prvi proces ce videti stari sadrzaj datoteke pre nego novi. (osim ako ne zatvori prethodno pa ga ponovo otvori).

- rename preimenuje ili premesta datoteku. Njegova dva argumenta su njegova stara putanja i nova putanja do datoteke. Ako su putanje ka razlicitim direktorijumima, rename premesta datoteku, sve dok su obe u istim sistemima datoteka. Mozete koristiti rename da pomerate direktorijume ili objekte drugih sistema datoteka takodje.

B.6 Citanje sadrzaja direktorijuma

GNU/Linux obezbedjuje funkcije za citanje sadrzaja direktorijuma. Iako, one nisu direktno vezane za U/I funkcije nizeg nivoa koje su su opisane u ovom dodatku, Mi ih objasnjavamo jer su cesto korisne u raznim aplikativnim programima.

Da bi procitali sadrzaj direktorijuma, pratite ove korake:

1. Pozovite opendir funkciju, dodajuci putanju direktorijuma koji zelimo da ispitamo. Funkcija opendir vraca DIR* identifikator, koji cete da koristite da pristupite sadrzaju direktorijuma. Ako se pojavi neka greska, funkcija vraca NULL.

2. Pozovite readdir ponavljajuci, prosledjuci DIR* identifikator koji ste dobili sa opendir. Svaki put kada pozovete readdir, on vraca pokazivac na struct dirent primer koji odgovara sledecem direktorijumu. Kada stignete do kraja sadrzaja tog direktorijuma, readdir vraca NULL vrednost.

   Struct dirent koju dobijete natrag od readdir ima polje d_name, koje sadrzi ime prisupljenog direktorijuma.

3. Pozovite closedir, prosledjuci DIR* identifikator, da bi zavrsili operaciju listanja.

Ukljucite <sys/types.h> i <dirent.h> ako koristite ove funkcije u svom programu.

Zapamtite to da ako zelite sadrzaj direktorijuma da bude slozem u odredjenom redosledu, da to morate sami da uradite.

Program u Listing B.8 ispisuje sadrzaj direktorijuma. Direktorijum moze biti zadat u komandnoj liniji, ali ako nije zadat, program koristi trenutni direktorijum kom je pristupljeno. Za svaki pristup direktorijumu, on prikazuje tip pristupa i njegovu putanju. Funkcija get_file_type koristi lstat da odredi tip sistema datoteka.

Listing B.8 (listdir.c) Ispis liste direktorijuma

```
#include <assert.h>
#include <dirent.h>
#include <stdio.h>
#include <string.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>

/*Vraca string koji opisuje tip sistema datoteka putanje*/

const char* get_file_type (const char* path)
{
    struct stat st;
    lstat (path, &st);

    if (S_ISLNK (st.st_mode))
        return “symbolic link”;
    else if (S_ISDIR (st.st_mode))
        return “directory”;
    else if (S_ISCHR (st.st_mode))
        return “character device”;
    else if (S_ISBLK (st.st_mode))
        return “block device”;
    else if (S_ISFIFO (st.st_mode))
        return “fifo”;
    else if (S_ISSOCK (st.st_mode))
        return “socket”;
    else if (S_ISREG (st.st_mode))
        return “regular file”;
    else
        /* Neocekivano. Svaki pristup bi trebalo da bude jedan tip od gore navedenih*/
        assert (0);
}

int main (int argc, char* argv[])
{
  char* dir_path;
  DIR* dir;
  struct dirent* entry;
  char entry_path[PATH_MAX + 1];
  size_t path_len;
```

```
  if (argc >= 2)
   /*Ako je direktorijum zadat na komandnoj liniji, koristite ga.*/
          dir_path = argv[1];
  else
   /*U suprotnom koristi trenutni direktorijum*/
          dir_path = ".";
  /*Kopira putanju direktorijuma u entry_path*/
  strncpy (entry_path, dir_path, sizeof (entry_path));
  path_len = strlen (dir_path);
  /*Ako putanja direktorijuma se ne zavrsava sa kosom crtom, dodaje je.*/
  if (entry_path[path_len - 1] != '/') {
          entry_path[path_len] = '/';
          entry_path[path_len + 1] = '\0';
          ++path_len;
  }

  /*Startovanje operacije listanja direktorijuma zadatog u komandnoj liniji.*/
  dir = opendir (dir_path);
  /*Petlja preko svih pristupljenih direktorijuma*/

  while ((entry = readdir (dir)) != NULL) {
          const char* type;
           /* Pravi putanju do pristupljenog direktorijuma dodavanjem imena cemu  treba
              da pristupi na ime putanje*/
          strncpy (entry_path + path_len, entry->d_name,
                   sizeof (entry_path) - path_len);
   /*Odredjivanje tipa pristupa*/
          type = get_file_type (entry_path);
   /*Ispisivanje tipa i pristupljene putanje*/
          printf ("%-18s: %s\n", type, entry_path);
  }
  /*Sve gotovo.*/
  closedir (dir);
  return 0;
}
```

Evo prvih nekoliko redova od ispisivanja liste /dev direktorijuma. (Vas ispis moze da se razlikuje po necemu)

```
% ./listdir /dev
directory : /dev/.
directory : /dev/..
socket : /dev/log
character device : /dev/null
regular file : /dev/MAKEDEV
fifo : /dev/initctl
character device : /dev/agpgart
...
```

Da bi proverili ovo, mozete koristiti komandu ls u istom direktorijumu. Navedite –u infikator da bi nalozili ls da ne sortira, i specificirajte –a indikator da bi uzrokovali da trenutni direktorijum (.) i roditeljski direktorijum (..) budu uracunati.

```
% ls -lUa /dev
total 124
drwxr-xr-x 7 root root 36864 Feb 1 15:14 .
drwxr-xr-x 22 root root 4096 Oct 11 16:39 ..
srw-rw-rw- 1 root root 0 Dec 18 01:31 log
crw-rw-rw- 1 root root 1, 3 May 5 1998 null
-rwxr-xr-x 1 root root 26689 Mar 2 2000 MAKEDEV
prw------- 1 root root 0 Dec 11 18:37 initctl
crw-rw-r-- 1 root root 10, 175 Feb 3 2000 agpgart
...
```

Prvi karakter svakog reda pokazuje koji je tip pristupa.

8. Linux Sistemski pozivi

Do sada predstavili smo razne funkcije koje vaš program može pozvati da bi odradio sistemski-orijentisane funkcije, kao što je parsiranje opcija iz komandne linije, manipulacija procesima i mapiranje memorije. Ako pogledate ispod “haube”, videcete da ove funkcije spadaju u dve kategorije, u zavisnosti od toga kako su implementirane:

Obicna funkcija - library function, nalazi se u biblioteci, izvan vašeg programa. Vecina ovakvih funkcija koje smo do sada predstavili su u standardnoj C biblioteci - libc. Na primer getopt_long and mkstemp. Poziv funkcije iz biblioteke je kao i svaki drugi poziv funkcije. Argumenti se smeštaju u procesorske registre ili na stack, a izvršavanje se premešta na pocetak samog koda funkcije, koji se obicno nalazi u ucitanoj biblioteci.

Sistemski poziv je implementiran u kernelu Linux -a. Kada program napravi sistemski poziv, argumenti se pakuju i predaju kernelu, koji preuzima izvršavanje programa sve dok se poziv ne završi. Sistemski poziv nije obican funkcijski poziv, tako da je potrebna specijalna procedura za transfer kontrola kernel -u. Medutim, GNU C biblioteka (tj. implementacija standardne C biblioteke koje dolazi uz GNU/Linux sisteme) obuhvata Linux sistemske pozive sa funkcijama tako da bi one bile lakše pozvane. I/O Funkcije niskog nivoa kao što su open and read su primeri sistemskih poziva na Linux -u.

Set Linux sistemskih poziva formira najosnovniji interfejs izmedu programa i Linux kernela.Svaki poziv predstavlja osnovnu operaciju ili mogucnost. Neki sistemski pozivi su veoma mocni i mogu imati velikog uticaja na sistem. Na primer, neki sistemski pozivi vam

omogucavaju da ugasite (shut down) Linux ili da alocirate sistemske resurse i sprecite druge korisnike da im pristupe. Ovi pozivi imaju restrikciju da samo mogu da ih pozovu procesi koje je stvorio korisnik sa Superuser privilegijama (tj. programi koje je startovao root).
Treba znati da funkcija može pozvati jednu ili više drugih funkcija ili sistemskih poziva u zavisnosti od toga kako su implementirane.
Linux trenutno pruža oko 200 raznovrsnih sistemskih poziva. Lista sistemskih poziva za vašu verziju Linux kernela je u /usr/include/asm/unistd.h. Neki od njih, operativni sistem koristi za internu upotrebu, a drugi se koriste samo pri implementaciji specijalnih funkcija. U ovom poglavlju predstavljacemo selekciju sistemskoh poziva koji su najkorisniji.
Vecina ovih sistemskih poziva su deklarisani u <unistd.h>.

8.1 Upotreba strace

Pre nego što pocnemo da diskutujemo o sistemskim pozivima bice korisno da predstavimo komandu sa kojom možete nauciti više o sistemskim pozivima i debagovati ih. Komanda strace prati izvršavanje nekog programa, listajuci sve sistemske pozive koje program napravi i sve signale koje prima.
Kako bi pratili sve sistemske pozive i signale potrebno je jednostavno pozvati strace, navodenjem ime programa sa svojim argumentima u komandnoj liniji. Na primer, u koliko želimo da pratimo sistemske pozive, koje poziva hostname[1] komanda, koristimo sledecu komandu:

```
% strace hostnam% hostname
```

Ova komanda proizvodi nekoliko ekrana izlaznog teksta. Svaka linija se odnosi na samo jedan sistemsji poziv. Za svaki poziv ispisuje se ime tog sistemskog poziva, zatim njegovi argumenti (ili njuhove skracenice ukoliko su predugacki) i njihova povratna vrednost. Gde je to moguce strace prikazuje simbolicka imena umesto numerickih vrednosti za argumente i povratne vrednosti, kao i polja struktura koje su predate pokazivacem u sistemski poziv. Napominjemo da strace ne može da prikaže pozive obicnih funkcija.
U izlazu strace hostname, prva linija predstavlja execve sistemski poziv koji u stvari poziva hostname program[2]:

```
execve("/bin/hostname", ["hostname"], [/* 49 vars */]) = 0
```

Prvi argument je ime programa koji se pokrece; drugi je njegova lista argumenata koja se sastoji od samo jednog elementa; i treci je lista okruženja, koju strace izostavlja. Sledecih 30 redova su deo mehanizma koji ucitava standardnu C biblioteku.

Pri kraju se nalaze sistemski pozivi koji ustvari pomažu da program završi posao. Uname je sistemski poziv koji se koristi da bi se pribavio hostname iz kernela.

uname({sys=″Linux″, node=″myhostname″, ...}) = 0

Primetite da strace obeležava polja (sys i node) u strukturi argumenata. Ovu strukturu popunjava sistemski poziv - Linux popunjava sys polje sa imenom operativnog sistema, a node polje kao hostname. Uname sistemski poziv se detaljnije objašnjava u Odeljku 8.15. “uname”.

!!

1 Komanda hostname poyvana bez parametara jednostavno štampa hostname racunara na standardni iylay

2 Ulinux-u, exec familija funkcija se implementira putem execve sistemskog poyiva!

Na kraju, write sistemski poziv proizvodi izlaz. Prisetite se da file descriptor 1 predstavlja standardni izlaz. Treci argument je broj karaktera koji treba ispisati, a povratna vrednost predstavlja broj karaktera koji je u stvari ispisan.

write(1, “myhostnamwrite(myhostname\n″, 11) =11

Kada pozovete strace izlaz može biti teško citljiv, zbog toga što se izlaz hostname programa može lako pomešati sa izlazom strace. Ako program koji pratite proizvodi mnogo izlaza, ponekad je bolje preusmeriti izlaz u neki fajl. Da bi ste ovo postigli koristite opciju -o filename.

Da bi se razumeo celokupan izlaz strace -a, potrebno je detaljno poznavanje Linux kernela, što prosecnom programeru i nije od presudne važnosti. Medutim, razumevanje je korisno za debagovanje težih problema ili za razumevanje kako odredeni programi funkcionišu.

8.2 access: Provera prava pristupa fajli

access sistemski poziv odreduje da li neki proces ima pravo pristupa fajli. Moguce je odrediti bilo koju kombinaciju read, write i execute dozvola, takode je moguce odrediti i da li fajl postoji. Sistemski poziv access prima 2 argumenta. Prvi je put do fajle koja se proverava, a drugi je u zavisnosti od dozvole: R_OK, W_OK, i X_OK. Povratna vrednost je 0 ako proces ima sve specificirane dozvole. Ako fajl postoji ali pozivajuci proces nema sve specificirane dozvole, access vraca -1 i setuje errno na EACCES (ili EROFS, ako je dozvola upisa (write) tražena na read-only file sistemu).

Ako je drugi argument F_OK, access jednostavno proverava da li fajl postoji, i ako postoji povratna vrednost je 0, a ako ne postoji povratna vrednost je -1 i errno je setovan na ENOENT. Erno je setovan na EACCES ukoliko je direktorijum, naveden u putu, nedostupan.

Program koji je prikazan u Listingu 8.1 koristi access da bi proverio da li fajl postoji i da bi odredio read i write dozvole. Na komandnoj liniji se unosi ime fajle.

Listing 8.1 (check-access.c) Provera prava pristupa fajle

```
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
int main (int argc, char* argv[])
{
char* path = argv[1];
int rval;

/* Check file existence. */
rval = access (path, F_OK);
if (rval == 0)
printf ("%s exists\n", path);
else {
if (errno == ENOENT)
printf ("%s does not exist\n", path);
else if (errno == EACCES)
printf ("%s is not accessible\n", path);
return 0;
}
/* Check read access. */
rval = access (path, R_OK);
if (rval == 0)
printf ("%s is readable\n", path);
else
printf ("%s is not readable (access denied)\n", path);
/* Check write access. */
rval = access (path, W_OK);
if (rval == 0)
printf ("%s is writable\n", path);
else if (errno == EACCES)
printf ("%s is not writable (access denied)\n", path);
else if (errno == EROFS)
printf ("%s is not writable (read-only filesystem)\n", path);
return 0;
}
```

Na primer, ako želite da proverite prava pristupa za fajl koji se zove README na CD-ROM -u, uradicete to na sledeci nacin:

```
% ./check-access /mnt/cdrom/README
```

```
/mnt/cdrom/README exists
/mnt/cdrom/README is readable
/mnt/cdrom/README is not writable (read-only filesystem)
```

8.3 fcntl: Zakljucavanje i druge operacije nad fajlovima

Sistemski poziv fcntl je tacka pristupa za nekoliko naprednih operacija nad file deskriptorima. Prvi argument fcntl je deskriptor za otvaranje fajlova, a drugi je vrednost koja prikazuje kakva operacija treba da bude izvršena. Za neke operacije fcntl uzima dodatni argument. Ovde cemo objasniti jednu od najkorisnijih operacija fcntl - zakljucavanje fajlova (file locking). Za ostale možete pogledati fcntl man strane za više informacija.

Sistemski poziv fcntl omogucava programu da zakljuca faljl za citanje ili pisanje, slicno mutex zakljucavanju o cemu smo diskutovali u Poglavlju 5. “Komunikacija izmedu procesa”. Read lock se postavlja na deskriptor fajle koja se može citati, a write lock se postavlja na deskriptor fajle u koju je moguc upis. Više procesa može imati read lock na istoj fajli u istom trenutku, a samo jedan proces može imati write lock, takode jedan fajl se ne može zakljucati i za upis i za citanje. Napominjemo da zakljucana fajla ne sprecava druge procese da je otvore, procitaju ili promene njen sadržaj, osim ako oni takode ne traže zakljucavanje sa fcntl.

Da bi se fajl zakljucao, prvo treba napraviti i izjednaciti sa 0 sva polja promenljive tipa struct flock. Setujte l_type polje strukture na F_RDLCK za read lock ili na F_WRLCK za write lock. Zatim pozovite fcntl, predajuci fajl deskriptor za fajl, F_SETLCKW kod operacije, i pokazivac na struct flock promenljivu. Ako je neki drugi proces zakljucao fajl da sprecava ponovno zakljucavanje, fcntl se blokira sve dok se fajl ne odkljuca.

U Listingu 8.2 program otvara fajl za citanje, cije je ime uneto u komandnoj liniji, a zatim postavlja write lock. Program ceka od korisnika da pritisne Enter, i tada odkljucava fajl i zatvara ga.

Listing 8.2 (lock-file.c) Postavljanje Write Lock-a sa fcntl

```
#include <fcntl.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
int main (int argc, char* argv[])
{
char* file = argv[1];
```

```
int fd;
struct flock lock;
printf (“opening %s\n”, file);

/* Open a file descriptor to the file. */
fd = open (file, O_WRONLY);
printf (“locking\n”);
/* Initialize the flock structure. */
memset (&lock, 0, sizeof(lock));
lock.l_type = F_WRLCK;
/* Place a write lock on the file. */
fcntl (fd, F_SETLKW, &lock);
printf (“locked; hit Enter to unlock... “);
/* Wait for the user to hit Enter. */
getchar ();
printf (“unlocking\n”);
/* Release the lock. */
lock.l_type = F_UNLCK;
fcntl (fd, F_SETLKW, &lock);
close (fd);
return 0;
}
```

Kompajlirajte i startujte program na test fajli /tmp/test-file na sledeci nacin:

```
% cc -o lock-file lock-file.c /*kompajliranje*/
% touch /tmp/test-file /*pravljenje test fajle*/
% ./lock-file /tmp/test-file /*startovanje programa*/
opening /tmp/test-file
locking
locked; hit Enter to unlock...
```

Sada probajte da iz drugog prozora startujete program na istoj fajli:

```
% ./lock-file /tmp/test-file
opening /tmp/test-file
locking
```

Dakle primeticemo da je druga instanca blokirana dok pokušava da zakljuca fajl. Vratite se na prvi prozor i pritisnite Enter, odziv ce biti:

```
unlocking
```

Program koji radi u drugom prozoru u istom trenutku zakljucava fajl. Ako želite da se fcntl ne blokira ako sistemski poziv ne može da zakljuca fajl, koristite K_SETLK umesto F_SETLKW. Na taj nacin, cim fajl ne može da se zakljuca, fcntl vraca -1 istog trenutka.

Linux pruža još jednu mogucnost implementacije zakljucavanja

fajlova sa flock pozivom. fcntl verzija ima glavnu prednost: radi sa

fajlovima na NTFS[3] file sistemima (ukoliko je NTFS server relativno novijeg datuma i korektno konfigurisan). Tako da, ukoliko imate pristup na 2 mašine koje mount -uju isti file sistem preko NTFS, možete da ponovite predhodni primer koristeci 2 razlicite mašine. Pustite lock-file na jednoj mašini, specificirajuci fajl na NTFS fajl sistemu, a zatim ucinite isto na drugoj mašini specificirajuci isti fajl. NTFS nastavlja drugi program kada se fajl otkljuca preko prvog programa.

8.11 readlink: Citanje sadržaja simbolickih linkova

Sistemski poziv readlink pribavlja put do objekta na koji pokazuje simbolicki link[6]. Uzima tri argumenta: put do simbolickog linka, bafer koji prima put do fajle na koju pokazuje link i dužinu tog bafera. Readlink ne stavlja null karakter ('\0') koji oznacava kraj stringa, u ovom slucaju bafera. Ali umesto toga vraca broj karaktera koji upisuje u bafer, tako da je upis oznake kraja bafera krajnje jednostavan.
Ako je prvi argument pokazuje na fajl koji nije simbolicki link, readlink postavlja vrednost errno na EINVAL i vraca -1.
Manji program u Listingu 8.9 prikazuje put do fajle na koju pokazuje simbolicki link (simbolicki link se unosi preko komandne linije).
Listing 8.9 (print-symlink.c) Prikaz puta do fajle na koju pokazuje simbolicki link

```
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
int main (int argc, char* argv[])
{
char target_path[256];
```

[6] Simbolicki link je precica ka objektu u sistemu datoteka. Simbolicki link dopušta linkove na direktorijume, nepostojece datoteke i datoteke koje se nalaye na drugom sistemu datoteka. Zauzima prostor na disku i jedno mesto u i-node tabeli.

```
char* link_path = argv[1];
/* Attempt to read the target of the symbolic link. */
int len = readlink (link_path, target_path, sizeof (target_path));
if (len == -1) {
/* The call failed. */
if (errno == EINVAL)
/* It's not a symbolic link; report that. */
```

```
fprintf (stderr, “%s is not a symbolic link\n”, link_path);
else
/* Some other problem occurred; print the generic message. */
perror (“readlink”);
return 1;
}
else {
/* NUL-terminate the target path. */
target_path[len] = ‘\0’;
/* Print it. */
printf (“%s\n”, target_path);
return 0;
}
}
```

Evo primera kako možete napraviti simbolicki link i iskoristiti program print-symlink da ga procitate:

```
% ln -s /usr/bin/wc my_link /*pravljenje simbolickog linka*/
% ./print-symlink my_link /*pozivanje programa*/
/usr/bin/wc /*izlaz iz programa*/
```

8.12 sendfile: Brzi prenos podataka

Sistemski poziv sendfile pruža efikasan mehanizam za kopiranje podataka iz jednog file deskriptora u drugi. Fajl deskriptori mogu biti otvoreni za fajlove na disku, soketima ili drugim uredajima. Procedura kopiranja jednog fajl deskriptora u drugi je sledeca: program alocira bafer fiksne velicine, kopira deo podataka iz jednog deskriptora u bufer, upisuje sadržaj bafera u drugi deskriptor i ponavlja ovu proceduru sve dok se svi podaci ne kopiraju. Ovaj postupak je ne efikasan u pogledu prostora, jer je potrebna dodatna memorija za bafer, i vremena, jer se vrši kopiranje podataka u taj dodatni bafer.

Korišcenjem sendfile, posredni bafer može biti eliminisan. Ovaj poziv prima sledece argumente: fajl deskriptor u koji se upisuje, deskriptor iz koga se cita, pokazivac na ofset promenljivu i broj bajtva za prenos. Ofset promenljiva sadrži ofset fajle iz koje se kopira, tj. mesto u fajli odakle se zapocinje kopiranje (0 predstavlja pocetak fajle). Ova promenljiva menja sadržaj na vrednost pozicije u fajli, nakon izvršenog transfera. Povratna vrednost je broj izkopiranih bajtova. Ukljucite biblioteku <sys/sendfile.h> ako u programu koristite sendfile. Program u Listingu 8.10 je jednostavan ali vrlo efikasno implementira kopiranje fajla. Kada se pozove, zajedno sa dva imena fajli na komandnoj liniji, on kopira sadržaj prve fajle u drugu. Program koristi

fstat za odredivanje velicine izvorišne fajle (u bajtovima).
Listing 8.10 (copy.c) Kopiranje fajlova korišcenjem sendfile

```
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/sendfile.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <unistd.h>
int main (int argc, char* argv[])
{
int read_fd;
int write_fd;
struct stat stat_buf;
off_t offset = 0;
/* Open the input file. */
read_fd = open (argv[1], O_RDONLY);
/* Stat the input file to obtain its size. */
fstat (read_fd, &stat_buf);
/* Open the output file for writing, with the same permissions as the
source file. */
write_fd = open (argv[2], O_WRONLY | O_CREAT,
stat_buf.st_mode);
/* Blast the bytes from one file to the other. */
sendfile (write_fd, read_fd, &offset, stat_buf.st_size);
/* Close up. */
close (read_fd);
close (write_fd);

return 0;
}
```

Poziv sendfile može se koristiti na mnogim mestima da bi kopiranje bilo efikasnije. Dobar primer bio bi na Web serveru ili drugom mrežnom servisu, koji šalje sadržaj fajle preko mreže klijentskom programu, na sledeci nacin: Serverski program otvara fajl na lokalnom disku, da bi uzeo sadržaj i upisuje te podatke u mrežni soket. Ovu operaciju znacajno ubrzavamo korišcenjem sendfile. Treba preuzeti i neke druge korake da bi transfer bio što brži, kao što je podešavanje parametara soketa.