INTERPROCESNA KOMUNIKACIJA

POGLAVLJE 3, ”PROCESI”, RAZMOTRILO JE NASTAJANJE PROCESA, i prikazalo je kako jedan proces može dobiti izlaz (exit) status od procesa deteta. To je najednostavnija forma komunikacije između dva procesa, ali je bez bez sumnje i najmoćnija. Mehanizam iz 3. Poglavlja ne omogućava nijedan vid komunikaciju roditelja i deteta izuzev preko argumenata komandne linije i varijabli okoline, niti omogućava bilo koji način komunikacije deteta sa roditeljem izuzev preko detetovog izlaz statusa. Nijedan od ovih mehanizama ne pruža način ili sredstvo za komunikaciju sa procesom detetom dok je ono u stanju izvršavanjanja, niti ovi mehanizmi dozvoljavaju komunikaciju sa procesima izvan odnosa roditelj-dete.

Ovo poglavlje opisuje sredstva za interprocesnu komunikaciju koja zaobilaze ova ograničenja. Predstavićemo različite načine komunikacije između roditelja i deteta, između ”nepovezanih” procesa, pa čak i između procesa na različitim mašinama.

Interprocesna komunikacija (IPC) je prenos podataka između procesa. Na primer, Web pretraživač može tražiti Web stranicu sa Web servera, koji zatim šalje HTML podatke. Ovaj prenos podataka obično koristi sokete u vidu telefonske konekcije. U drugom primeru, želite da izlistate imena fajlova u direktorijumu koristeći komandu ls | lpr. Šel stvara ls proces i poseban lpr proces, povezujući ova dva procesa pomoću *pajpa* (eng. pipe) predstavljenog simbolom " | " . Pajp dozvoljava jednosmernu komunikaciju između dva srodna procesa. Proces ls upisuje podatke u pajp, a proces lpr ih izčitva iz pajpa.

U ovom poglavlju, razmotrićemo pet tipova interprocesne komunikacije:

- Deljiva memorija (eng. shared memory) dozvoljava procesima da komuniciraju prostim čitanjem i upisivanjem u određenu memorijsku lokaciju.
- Mapirana memorija je slična deljivoj memoriji, izuzev što je povezana sa fajlom u fajl sistemu.
- Pajpovi dozvoljavaju sekvencijalnu komunikaciju jednog procesa sa srodnim procesom.
- FIFO su slični pajpovima, izuzev što i nepovezani procesi mogu komunicirati jer je pajpu dodeljeno ime u fajl sistemu.
- Soketi podržavaju komunikaciju između nepovezanih procesa čak i na različitim računarima.

Ovi tipovi IPK se razlikuju po sledećim kriterijumima:

- Da li ograničavaju komunikaciju na srodne procese (procese sa zajedničkim pretkom), na nepovezane procese koji dele isti fajl sistem, ili na bilo koji računar povezan na mrežu.
- Da li je komunikacioni proces ograničen samo na upisivanje podataka ili samo na čitanje podataka.
- Broju procesa kojima je dozvoljeno da komuniciraju.
- Da li su komunikacioni procesi sinhronizovani pomoću IPK - na primer proces čitanja je u stanju čekanja sve dok podaci ne postanu dostupni za čitanje.

U ovom poglavlju ćemo zanemariti diskusiju o IPK dozvoli za komunikaciju samo određeni broj puta, kao što je komunikacija preko izlaz statusa deteta.

5.1 Deljiva memorija

Jedan od najednostavnijih metoda interprocesne komunikacije je upotreba deljive memorije. Deljiva memorija omogućava da dva ili više procesa pristupe istoj memoriji ako pozovu malloc i ako svi vraćeni pokazivači ukazuju na istu memoriju. Kada jedan proces promeni memoriju, svi ostali procesi vide promenu.

5.1.1 Brza lokalna komunikacija

Deljiva memorija je najbrža forma interprocesne komunikacije zato što svi procesi dele isti deo memorije. Pristup ovoj deljivoj memoriji je isto toliko brz kao i pristup procesnoj nedeljivoj memoriji, i ne zahteva sistemski poziv ili pristup kernelu. Takođe se izbegava nepotrebno kopiranje podataka.Ozbirom da kernel ne sinhornizuje pristupe deljivoj memoriji, potrebno je da obezbedite svoju sopstvenu sinhronizaciju. Na primer, proces nebi trebao čitati iz memorije dok se ne završi upis podataka u nju, niti bi dva procesa trebala da upisuju u istu memorijsku lokaciju u isto vreme. Česta strategija za izbegavanje ovih stanja gužve je upotreba semafora, koja je osmotrena u sledećem odeljku. Naši ilustrativni programi, ipak, prikazuju kako samo jedan proces pristupa memoriji da bi se fokusirali na mehanizam deljive memorije i kako bi izbegli da iskomplikujemo kod primera sa sinhronizacionom logikom.

5.1.2 Memorijski model

Da bi koristio deljivi memorijski segment, jedan proces mora alocirati segment. Zatim svaki proces koji želi da pristupi segmentu mora pridobiti segment. Nakon završetka korišćenja segmenta, svaki proces mora odbaciti segment.

Razumevanje Linux memorijskog modela pomaže da se objasni alokacija i pridobijanje procesa. Pod Linux-om, svaka procesna virtuelna memorija je podeljena na stranice. Svaki proces mora voditi računa o mapiranju od svojih memorijskih adresa do ovih virtualnih memorijskih stranica, koje sadržavaju aktuelne podatke. Iako svaki proces poseduje sopstvene adrese, višestruko procesno mapiranje može ukazivati na istu stranicu, dozvoljavajući deljenje memorije. Memorijske stranice se daljne osmatraju u Odeljku 8.8, "Mlock familija: Zaključavanje fizičke memorije," u Poglavlju br. 8, "Linux sistemski pozivi."

Alokacija novog deljivog memorijskog segmenta prouzrokuje kreiranje virtualnih memorijskih stranica. Obzirom da svi procesi žele da pristupe istom deljivom segmentu, samo bi jedan proces trebao da alocira novi deljivi segment. Alociranjem postojećeg segmenta ne kreiraju se nove stranice, već se vraća identifikator za već postojeće stranice. Da bi dozvolio procesu da koristi deljivi memorijski segment, proces ga pridružuje, što dodaje mapirani unos iz njegove virtualne

memorije u segmentove deljive stranice. Kada završimo sa segmentom, ovi mapirani unosi se uklanjaju. Kada više nema procesa koji bi koristili deljivi memorijski segment, tačno jedan proces mora delocirati virtualne memorijske stranice.
Svi deljivi memorijski segmenti su alocirani kao višestruki integrali veličine sistemske strane, koja je broj bajtova u stranici memorije. Na Linux sistemima, veličina stranice iznosi 4KB, ali vi morate odrediti ovu vrednost pozivajući funkciju getpagesize.

5.1.3 Alokacija

Proces alocira deljivi memorijski segment korišćenjem shmget ("SHared Memory GET"). Njegov prvi parametar je celobrojni ključ koji određuje koji segment se kreira. Nesrodni procesi mogu pristupiti istom deljivom segmentu pomoću specifiranja iste vrednosti ključa. Na nesreću, drugi procesi mogu takođe odabrati isti fiksni ključ, a što može voditi do konflikta. Upotrebom konstante IPC_PRIVATE kao vrednosti ključa garantuje da će se kreirati potpuno novi memorijski segment.

Njegov drugi parametar definiše broj bajtova u segmentu. Obzirom da su segmenti alocirani korišćenjem stranica, broj stvarno alociranih bajtova je zaokružen na višestruki integral veličine stranice.

Treći parametar je pametan bit ili vrednost flega koji određuje opcije shmget-a. Vrednosti flega uključuju sledeće:

- IPC_CREAT—Ovaj fleg ukazuje na potrebu za kreiranjem novog segmenta. Ovo omogućava kreiranje novog segmenta tokom određivanja vrednosti ključa.
- IPC_EXCL—Ovaj fleg koji se uvek koristi sa IPC_CREAT, prouzrokuje neuspeh u izvršenju shmget-a u slučaju da je određen ključ segmenta koji već postoji. Zbog toga, on uređuje da pozvani proces ima ekskluzivan segment. U slučaju da ovaj fleg nije dat i koristi se ključ već postojećeg segmenta, shmget vraća postojeći segment umesto da kreira novi.
- Mode flags—Ova vrednost je sačinjena od 9 bitova koji ukazuju na prava dodeljena vlasniku, grupi, i svetu (drugi) za kontrolu pristupa segmentu. Izvršni bitovi su zanemareni. Jednostavan način da se odrede prava je korišćenje konstanti definisanih u <sys/stat.h> i dokumentovanih u Odeljku 2 u stat man page. [1] Na primer, S_IRUSR i S_IWUSR određuju prava čitanja i upisivanja za vlasnika deljivog memorijskog segmenta, a S_IROTH i S_IWOTH IWUSR određuju prava čitanja i upisivanja za druge.

Na primer, ovakvo pozivanje shmget-a kreira novi deljivi memorijski segment (ili pristup već postojećem u slučaju da je shm_key već upotrebljen) koji je čitljiv i upisiv za vlasnika, ali ne i za druge korisnike.

```
int segment_id = shmget (shm_key, getpagesize (),
                         IPC_CREAT | S_IRUSR | S_IWUSER);
```

U slučaju da poziv uspe, shmget vraća naziv segmenta. Ako deljivi memorijski segment već postoji, vrši se verifikacija prava pristupa i proverava se da li je segment označen za uništenje.

5.1.4 Attachment and Detachment (pridruživanje i odbacivanje)

Da bi deljivi memorijski segment učinio dostupnim, proces mora koristiti shmat, "SHared Memory ATtach." Pokrenuti ga zajedno sa identifikatorom deljivog memorijskog segmenta dobijenim pomoću shmget. Drugi argument je pokazivač koji određuje gde u vašem procesnom adresnom prostoru želite da mapirate deljivu memoriju; ako odrediti vrednost NULL, Linux će odabrati slobodnu adresu. Treći argument je fleg, koji može uključivati sledeće:

- SHM_RND ukazuje da adresa koja je određena za drugi parametar, treba da bude zaokružena na sadržilac veličine stranice. Ako ne postavite ovaj fleg, potrebno je lično podesiti drugi argument za shmat.
- SHM_RDONLY ukazuje da će segment biti samo čitljiv, ne i upisiv.

Ako poziv uspe, on vraća adresu pridruženog deljivog segmenta. Deca kreirana pozivom forka nasleđuju pridružene deljive segmente; ako žele ona mogu odbaciti deljive memorijske segmente.
Kada ste završili sa deljivim memorijskim segmentom, segment je potrebno odbaciti koristeći shmdt ("SHared Memory DeTach"). Koristiti adresu dobijenu pomoću shmat. Ako je segment oslobođen i ako je ovo bio poslednji proces koji ga je koristio, on se uklanja. Poziv izlaza (exit) ili bilo kog člana exec familije automatski odbacuje segment.

5.1.5 Kontrola i Oslobađanje Deljive Memorije

Poziv shmctl ("SHared Memory ConTroL" – kontrola deljive memorije) vraća informaciju o deljivom memorijskom segmentu i može ga modifikovati. Prvi parametar je identifikator deljivog memorijskog segmenta.
Da bi dobili informaciju o deljivom memorijskom segmentu potrebno je koristiti IPC_STAT kao drugi argument i pokazivač na strukturu shmid_ds.
Za uklanjanje segmenta koristi se IPC_RMID kao drugi argument i NULL kao treći argument. Segment je uklonjen kada ga i poslednji proces koji ga je pridružio konačno odbaci.

Svaki deljivi memorijski segment bi trebalo eksplicitno osloboditi koristeći shmctl kada se završi sa njim, da bi se izbeglo narušavanje limita opsega sistema u pogledu ukupnog broja deljivih memorijskih segmenata. Upotrebom komandi exit i exec odbacuju se memorijski segmenti ali se ne oslobađaju .

Pogledajte shmctl man stranicu za opis drugih operacija koje se mogu obaviti na deljivim memorijskim segmentima.

5.1.6 PRIMER PROGRAMA

Program u Listingu 5.1 ilustruje upotrebu deljive memorije.

Listing 5.1 *(shm.c)* **Vežba Deljive Memorije**

```
#include <stdio.h> #include
<sys/shm.h> #include <sys/stat.h>

int main () {
 int segment_id;
 char* shared_memory;
 struct shmid_ds shmbuffer;
```

```
int segment_size;
const int shared_segment_size = 0x6400;

/* Alocira deljivi memorijski segment. */
segment_id = shmget (IPC_PRIVATE, shared_segment_size,
          IPC_CREAT | IPC_EXCL | S_IRUSR | S_IWUSR);

/* Pridružuje deljivi memorijski segment.  */
shared_memory =  (char*)  shmat  (segment_id, 0, 0);
printf ("deljiva memorija pridružena na adresi %p\n",
shared_memory);
/* Određuje veličinu segmenta. */
shmctl (segment_id, IPC_STAT, &shmbuffer);
segment_size = shmbuffer.shm_segsz;
printf ("veličina segmenta: %d\n", segment_size);
/* Upisuje string u deljivi memorijski segment. */
sprintf (shared_memory, "Hello, world.");
/* Odbacivanje deljivog memorijskog segmenta. */
shmdt (shared_memory);

/* Ponovno pridruživanje deljivog memorijskog segmenta na drugoj adresi.
*/
shared_memory = (char*) shmat (segment_id, (void*)
0x5000000, 0);
printf ("deljiva memorija ponovo pridružena na adresi %p\n",
shared_memory);
/* Ispisivanje stringa iz deljive memorije. */
printf ("%s\n", shared_memory);
/* Odbacivanje deljivog memorijskog segmenta. */
shmdt (shared_memory);

/* Oslobađanje (dealokacija) deljivog memorijskog
segmenta. */

 shmctl (segment_id, IPC_RMID, 0);

return 0; }
```

5.3 Mapirana memorija

Mapirana memorija dozvoljava različitim procesima da komuniciraju preko deljenog fajla. Premda možda gledate na mapiranu memoriju kao na upotrebu deljivog memorijskog segmenta sa imenom, morate biti svesni da postoje tehničke razlike. Mapirana memorija može se koristiti za interprocesnu komunikaciju ili kao jednostavan način za pristup sadržaju fajla.

Mapirana memorija formira vezu između fajla i procesne memorije. Linux deli fajl na delove veličine stranice i onda ih kopira u stranice virtualne memorije kako bi ih učinio dostupnim u

procesnom adresnom prostoru. Zato, procesi mogu čitati sadržaj fajla pomoću običnog memorijskog pristupa. Takođe, on može menjati sadržaj fajla upisujući u memoriju. Ovo omogućava brz pristup fajlovima.

Na mapiranu memoriju možete gledati kao na zaduženju bafera da drži čitav sadržaj fajla, i onda na čitanje fajla u baferu i (ako je bafer menjan) na kraju upisivanje bafera nazad u fajl. Linux rukuje operacijama čitanja i upisivanja za vas.
Pored interprocesne komunikacije postoje i druge upotrebe mapirane memorije. Neke od njih su obrađene u Odeljku 5.3.5, "Druge upotrebe mapirane memorije."

5.3.1 Mapiranje običnog fajla

Da bi se mapirao običan fajl u procesnu memoriju, koristite poziv mmap ("Memory MAPped," izgovara se "em-map"). Prvi argument je adresa na koju želite da Linux mapira fajl u vašem procesnom adresnom prostoru; Vrednost NULL dozvoljava Linuxu da odabere dostupnu početnu adresu. Drugi argument je dužina mape u bajtima. Treći argument određuje zaštitu mapiranog adresnog opsega. Ova zaštita se sastoji od pametnog bita "ili" PROT_READ, PROT_WRITE, i PROT_EXEC, koje odgovaraju pravima čitaj, piši, i izvrši respektivno. Četvrti argument je vrednost flega koja određuje dodatne opcije. Peti argument je fajl deskriptor koji opisuje fajl koji treba da se mapira. Poslednji argument je ofset od početka fajla odakle počinje mapiranje. Odgovarajućim izborom početnog ofseta i odgovarajuće dužine, moguće je da mapirate ceo ili deo fajla u memoriji. Vrednost flega je pametan bit "ili" tj. neka od ovih konstanti:

- MAP_FIXED—Ako postavimo ovaj fleg, Linux koristi zatraženu adresu da mapira fajl radije nego da je tretira kao nagoveštaj. Ova adresa mora biti stranično podešena.
- MAP_PRIVATE—Upisi u memorijski opseg se ne vraćaju u pridruženi fajl, nego u privatnu kopiju fajla. Nijedan drugi proces ne vidi ove upise. Ovaj mod ne bi trebalo koristiti sa MAP SHARED.
- MAP_SHARED—Upisi se odmah odražavaju na osnovni fajl, a ne na upise u bafer. Koristite ovaj mod kad koristite mapiranu memoriju za IPK. Ovaj mod ne bi trebalo koristiti sa MAP_PRIVATE.

Ako poziv uspe, on vraća pokazivač na početak memorije. Pri neuspehu, vraća MAP_FAILED.

Kada ste završili sa mapiranjem memorije, oslobodite je koristeći munmap. Pokrenite komandu navodeći početnu adresu i dužinu mapiranog memorijskog prostora. Linux automatski demapira mapirani prostor nakon završetka procesa.

5.3.2 Primer programa

Da bi ilustrovali korišćenje mapiranog memorijskog prostora osmotrimo dva programa za čitanje i upis u fajl. Prvi program, Listing 5.5, generiše slučajne brojeve i upisuje ih u mapiran memorijski fajl. Drugi program. Listing 5.6, čita brojeve, štampa ih, i zamenjuje ih u mapiranom memorijskom fajlu sa njihovom duplom vrednošću. Oba koriste argumente komandne linije fajla koji se mapira.

Listing 5.5 *(mmap-write.c)* **Upis slučajnog broja u mapirani memorijski fajl**

```
#include <stdlib.h> #include
<stdio.h> #include <fcntl.h>
#include <sys/mman.h>
```

```
#include <sys/stat.h>
#include <time.h> #include
<unistd.h> #define
FILE_LENGTH 0x100

/* Vraća ujednačene slučajne brojeve u opsegu [low,high]. */

int random_range (unsigned const low, unsigned const high)
{
 unsigned const range = high - low + 1; return low +
 (int) (((double) range) * rand () / (RAND_MAX +
 1.0));


int main  (int argc,  char* const argv[])
{
  int fd; void* file_memory;

  /* Generator slučajnih brojeva.   */

  srand  (time  (NULL));

  /* Priprema fajl dovoljno velik za prihvat neoznačenog celog broja (intirdžer)*/


 fd = open
(argv[1],O_RDWR  |  O_CREAT,
S_IRUSR    |  S_IWUSR); lseek
(fd,    FILE_LENGTH+1,
SEEK_SET);write  (fd,  "",  1);
lseek  (fd,  0,  SEEK_SET);

/* Kreira mapiranu memoriju. */
file_memory = mmap (0, FILE_LENGTH, PROT_WRITE, MAP_SHARED, fd,
0);
close (fd);
/* Upisuje slučajan ceo broj u mapirani memorijski prostor. */
sprintf((char*) file_memory, "%d\n", random_range (-100, 100));
/* Oslobađa memoriju (nepotrebno jer se program završava). */
munmap (file_memory, FILE_LENGTH);

return 0;
```

Program mmap-write otvara fajl, kreirajući ga ako to prethodno nije urađeno. Treći argument to open određuje da je fajl otvoren za čitanje i upisivanje. Obzirom da ne znamo dužinu fajla, koristimo lseek da bi se uverili da je fajl dovoljno veliki da smesti ceo broj i onda se se vraćamo na početanu poziciju u fajlu.
Program mapira fajl i zatim zatvara fajl deskriptor zato što više nije potreban. Program zatim upisuje slučajan ceo broj u mapiranu memoriju, time i u fajl, a zatim demapira memoriju. Poziv

munmap je nepotreban zato što će Linux automatski izvršiti unmap fajla nakon završetka programa.

Listing 5.6 *(mmap-read.c)* **Čita intidžer iz mapiranog memorijskog fajla i udvostručuje ga**

```
#include <stdlib.h> #include <stdio.h>
#include <fcntl.h> #include <sys/mman.h>
#include <sys/stat.h> #include <unistd.h>
#define FILE_LENGTH 0x100

int main  (int argc,  char* const argv[])
{
  int fd;
  void* file_memory; int
  integer;

  /* Otvara fajl.     */
  fd = open  (argv[1],  O_RDWR,  S_IRUSR   |   S_IWUSR);
  /* Kreira mapiranu memoriju.    */
  file_memory = mmap  (0,  FILE_LENGTH,  PROT_READ   |   PROT_WRITE,
  MAP_SHARED,  fd,  0); close  (fd);



        /* Čita intidžer, štampa ga, i udvostručuje.    */
        scanf  (file_memory,   "%d", &integer);
        printf  ("value: %d\n", integer);
        sprintf  ((char*)  file_memory,   "%d\n", 2 * integer);
        /* Oslobađa memoriju (nepotrebno zato što se program završava).    */
        munmap  (file_memory,  FILE_LENGTH);

        return 0; }
```

Program mmap-read čita broj iz fajla i zatim upisuje u fajl udvostručenu vrednost. Prvo, on otvara fajl i mapira ga za čitanje i upisivanje. Obzirom da možemo predpostaviti da je fajl dovoljno veliki da prihvati neoznačeni ceo broj (intidžer), nije potrebno da koristimo lseek, kao u prethodnom programu. Program čita i rastavlja vrednosti van memorije koristeći sscanf i zatim ih formatira i upisuje duple vrednosti koristeći sprintf.

Ovo je primer izvršenja ovog programa. On mapira fajl /tmp/integer-file.

```
% ./mmap-write /tmp/integer-file
% cat /tmp/integer-file
42
% ./mmap-read /tmp/integer-file
value: 42
% cat /tmp/integer-file
```

Primeti da je tekst 42 zapisan na disk bez pozivanja upisa, i ponovo je pročitan bez poziva čitanja. Zapazi da ovaj program-primer upisuje i čita intidžer kao string (koristeći sprintf i sscanf) u demonstrativnu svrhu - nema potrebe da sadržaj mapiranog memorijskog fajla bude tekst. Možete smestiti i ponovo uzeti proizvoljne binarne brojeve iz mapiranog memorijskog fajla.

5.3.3 Deljivi pristup fajlu

Različiti procesi mogu komunicirati koristeći mapirane memorijske oblasti povezane istim fajlom. Postavite fleg MAP_SHARED tako da bilo koji upis u ove oblasti bude odmah prenet u osnovni fajl i bude vidljiv drugim procesima. Ako ne postavite ovaj fajl, Linux može baferovati upise pre nego što ih prenese u fajl.

Alternativno, možete prinuditi Linux da unese baferovane unose u fajl na disku pozivajući msync. Njegova prva dva parametra određuju mapirani memorijski prostor, kao i kod munmap-a. Treći parameter može uzeti sledeće fleg vrednosti:

- MS_ASYNC—Ažuriranje je predviđeno, ali ne mora bezuslovno da se izvrši pre kraja poziva.

- MS_SYNC—Ažuriranje je trenutno; poziv msync se blokira dok se potpuno ne izvrši.

Nije moguće zajedno koristiti MS SYNC i MS ASYNC.■ MS_INVALIDATE—Sve drugo mapiranje fajlova je poništeno kako bi oni mogli videte ažurirane vrednosti.

Na primer: da bi osvežili deljivi fajl mapiran na adresi address mem_addr dužine mem_length bytes, pozovimo sledeće:

```
msync (mem_addr, mem_length, MS_SYNC | MS_INVALIDATE);
```

Kao i kod deljivih memorijskih segmenata, korisnici mapiranih memorijskih oblasti moraju uspostaviti i pridržavati se protokola za izbegavanje stanja zbrke. Na primer, semafor se može koristiti da spreči da u jednom trenutku jednoj mapiranoj memoriji pristupa više od jednog procesa. Alternativno, možete koristiti fcntl da postavite zabranu čitanja ili upisivanja fajla, kao što je opisano u Odeljku 8.3, "fcntl: Zaključavanje i druge fajl operacije," iz Poglavlja 8.

5.3.4 Privatno mapiranje (Private mapping)

Postavljanjem MAP_PRIVATE kod mmap kreira se oblast "kopiraj pri upisu" (copy-on-write) Bilo koji upis u oblast se reflektuje samo na ovu procesnu memoriju; drugi procesi koji mapiraju isti fajl neće videti promene. Umesto da upisuje direktno na stranicu koja je deljiva sa svim procesima, proces upisuje na privatnu kopiju ove stranice. Svako naredno čitanje i pisanje ovog procesa koristi ovu stranicu.

5.3.5 Druge upotrebe *mmap*-a

Pored interprocesne komunikacije poziv mmap se može koristiti i u druge svrhe. Jedna uobičajena upotreba je zamena čitanja i upisivanja. Na primer, umesto da eksplicitno pročita sadržaj fajla u memoriju, program će radije da mapira fajl i skenira ga koristeći memorijsko čitanje. Za neke programe ovo je pogodnije i takođe se može izvršiti brže nego eksplicitne U/I fajl operacije.

Jedna napredna i moćna tehnika koju koriste neki programi je izrada strukture podataka (obični

struct slučajevi, na primer) u mapiranom memorijskom fajlu. Pri kasnijem pozivu, program mapira taj fajl ponovo u memoriju, a strukture podataka se vraćaju u svoje prethodno stanje. Zapazite, da će pokazivači na ove strukture podataka biti pogrešni izuzev ako svi ne pokazuju u granicama istog mapiranog memorijskog prostora i ako se ne pobrine da se fajl ponovo mapira u isti adresni prostor koji je ranije zauzimao.
Druga pogodna tehnika je mapiranje specijalnog /dev/zero fajla u memoriju. Ovaj fajl, koji je opisan u Odeljku 6.5.2, "/dev/zero," iz Poglavlja 6, "Uređaji," se ponaša kao beskonačno dug fajl popunjen sa 0 bajtovima. Program kome je potreban izvor 0 bajtova može koristiti mmap fajl /dev/zero. Upisivanja u /dev/zero se odbacuju , tako da se mapirana memorija može koristiti u bilo koju svrhu. Uobičajeni memorijski alokatori često mapiraju /dev/zero da bi dobili delove preinicijalizovane memorije.

5.4 Pajpovi (cevi, engleski pipes)

Pajp je komunikacioni uređaj koji dozvoljava višedirekcionu komunikaciju. Podatak upisan na upisni kraj ("write end") pajpa, izčitava se sa kraja za čitanje "read end." Pajpovi su serijski uređaji; podaci se uvek čitaju iz pajpa u istom redu kako su upisani. Obično se pajp koristi za komunikaciju između dve niti u jednom procesu ili između procesa roditelja i procesa deteta. U šelu, simbol | kreira pajp. Na primer ova šel komanda prouzrokuje da šel stvori dva procesa deteta, jedna za ls i jedan za less:

```
% ls | less
```

Šel takođe stvara pajp spajajući standardni izlaz ls podprocesa sa standardnim ulazom less procesa. Imena fajlova izlistana pomoću ls se šalju ka less u istom onom poretku u kojem bi bili poslani direktno ka terminalu.
Kapacitet podataka pajpa je ograničen. Ako proces upisa piše brže nego što proces za čitanje može da preuzme podatke, i ako pajp ne može da sačuva višak podataka, proces upisa se blokira dok veći kapacitet ne postane dostupan. Ako čitač pokuša da pročita, a ne postoje raspoloživii podaci, on se blokira sve dok podaci ne postanu dostupni. Zato, pajpovi automaski sinhronizuju dva procesa.

5.4.1 Kreiranje pajpova

Da bi kreirali pajp, moramo koristiti pajp komandu. Sastoji se od celobrojnog niza veličine 2. Poziv pajpa smešta deskriptor čitanja fajla na poziciju 0 u nizu, a deskriptor upisa fajla na poziciju 1. Na primer, osmotrimo sledeći kod:

```
int pipe_fds[2]; int read_fd; int
write_fd;

    pipe  (pipe_fds); read_fd =
pipe_fds[0]; write_fd = pipe_fds[1];
```

Podaci upisani od strane deskriptora upisa fajla read_f d se mogu pročitati pomoću write_f d.

5.4.2 Komunikacija između procesa roditelja i procesa deteta.

Poziv pajpa kreira dva fajl deskriptore, koji su validni samo između tog procesa i njegove deca. Procesni fajl deskriptori ne mogu prelaziti između dva nepovezana procesa; međutim kada proces pozove fork, fajl deskriptori se kopiraju na novi proces dete. Zato, pajpovi mogu spajati jedino povezane procese.

U programu iz Listinga 5.7, fork stvara proces dete. Dete nasleđuje pajp fajl deskriptore. Roditelj upisuje string u pajp, a dete ga izčitava. Jednostavan program konvertuje ove fajl deskriptore u FILE* niz koristeći fdopen. Obzirom da radije koristimo nizove neko fajl deskriptore, možemo koristiti ulazno/izlazne funkcije višeg reda iz standardne C biblioteke kao što su printf i fgets.Listing 5.7 *(pipe.c)* **Korišćenje pajpa za komunikaciju sa procesom detetom**

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

  /* Upisuje COUNT kopije iz PORUKE u STREAM, praveći
  pauzu od jedne sekunde između svake. */

void writer (const char* message, int count, FILE* stream)
 { for (; count > 0; --count) {
/* Upisuje poruku u niz, i odmah ih šalje dalje.. */
fprintf (stream, "%s\n", message);
fflush (stream);
/* Ceka malo. */ sleep (1); } }

/* Čita proizvoljne stringove iz niza što je duže moguće. */

void reader (FILE* stream)
{

 char buffer[1024];
 /* Čita dok ne stignemo do kraj niza. Fgets čita do nove linije
 ili do kraja. */
  while (!feof (stream)
      && !ferror (stream)

&& fgets (buffer, sizeof (buffer), stream) != NULL)
fputs (buffer, stdout); }

int main ()
{

 int fds[2]; pid_t pid;
 /* Kreira pajp. Fajl deskriptori za dva kraja pajpa se
 smeštaju u fds. */ pipe (fds);
/* Forkuje se proces dete. */

pid = fork (); if (pid == (pid_t) 0) { FILE* stream;

/* Ovo je proces dete. Zatvara našu kopiju kraja za upis fajl
deskripotra.. */

close (fds[1]);

/* Konvertuje fajl deskriptor za čitanje u fajl object, i čita iz
njega. */

   stream = fdopen (fds[0], "r"); reader
   (stream);
          close (fds[0]); } else {
           /* Ovo je proces roditelj. */
```

```
          FILE* stream;
      /* Zatvara našu kopiju kraja za čitanje fajl deskripora. */
close (fds[0]);
            /* Konvertuje fajl descriptor upisa u fajl object, i
            upisuje u njega. */
          stream = fdopen (fds[1], "w");
          writer ("Hello, world.", 5, stream);
          close (fds[1]);


        return 0; }
```

Na početku main-a, fds se deklariše kao celobrojni niz veličine 2. Pajp poziv kreira pajp i smešta deskriptore za upis i čitanje u taj niz. Program tada forkuje proces dete. Nakon zatvaranja kraja za čitanje pajpa, proces roditelj počinje da upisuje stringove u pajp. Nakon zatvaranja kraja za upis pajpa, dete čita stringove iz pajpa.

Zapazite da nakon upisivanja u funkciji za upis, roditelj flešuje pajp pozivajući fflush. U suprotnom, može se desiti da string ne bude poslat odmah kroz pajp.

Kada pozivate komandu ls | less, dešavaju se dva forka: jedan za ls dete proces i jedan less dete proces. Oba ova procesa nasleđuju pajp fajl deskriptore kako bi mogli komunicirati koristeći pajp. Ako želite da ostvarite komunikaciju između nepovezanih procesa koristite FIFO, a što je i osmotreno u Odeljku 5.4.5, "FIFO."

5.4.3 Preusmeravanje standardnog ulaza, izlaza i strimova grešaka

Često ćete želeti da kreirate proces dete i da postavite jedan kraj pajpa kao njegov standardni ulaz ili standardni izlaz. Korišćenjem poziva dup2, možete izjednačiti jedan fajl deskriptor sa drugim. Na primer, da bi se preusmerio standardni ulaz procesa na fajl descriptor fd, koristite sledeću liniju: dup2 (fd, STDIN_FILENO);

Simbolička konstanta STDIN_FILENO predstavlja fajl deskriptor za standardni ulaz, koji ima vrednost 0. Poziv zatvara standardni ulaz i ponovo ga otvara kao duplikat fd-a kako bi se mogao koristiti za razmenjivanje. Izjednačeni fajl deskriptori dele istu poziciju fajla i iste fajl status flegove. Zato se karakteri pročitani pomoću fd-a ne čitaju ponovo kroz standardni ulaz. Program iz Listinga 5.8 koristi dup2 da pošalje izlaz iz pajpa ka komandi za sortiranje. [2] Nakon kreiranja pajpa, program se forkuje. Proces roditelj ispisuje stringove kroz pajp. Proces dete pridružuje fajl deskriptor za čitanje pajpa na svoj standardni ulaz koristeći dup2. Tada se izvršava program za sortiranje.

Listing 5.8 *(dup2.c)* **Preusmeravanje izlaza pajpa pomoću** *dup2*

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

```
 int main ()
{
   int fds[2]; pid_t pid;
   /* Kreira pajp. Fajl deskriptori dva kraja pajpa se
   smeštaju u fds. */ pipe (fds);
   /* Forkuje se proces dete. */
   pid = fork (); if (pid ==
   (pid_t) 0) {
      /* Ovo je proces dete. Zatvara se naša kopija kraja za
      upis fajl deskriptora. */
    close (fds[1]);
/* Povezuje se kraj za čitanje pajpa na standardni ulaz. */
    dup2 (fds[0], STDIN_FILENO);
/* Zamenjuje se proces dete sa programom za sortiranje. */
    execlp ("sort", "sort", 0);
   } else {
/* Ovo je proces roditelj. */
    FILE* stream;
/* Zatvara se naša kopija kraja za čitanje fajl deskriptora. */
    close (fds[0]);
      /* Kovertuje se fajl deskriptor za upis u fajl object, i
      upisuje se u njega. */
    stream = fdopen (fds[1], "w");
    fprintf (stream, "This is a test.\n");
    fprintf (stream, "Hello, world.\n");
    fprintf (stream, "My dog has fleas.\n");
    fprintf (stream, "This program is great.\n");
    fprintf (stream, "One fish, two fish.\n");
    fflush (stream);
    close (fds[1]);
/* Čeka se da proces dete završi. */
   waitpid (pid, NULL, 0);
   return 0;}
```

5.5 Soketi

Soket je bidirekcioni uređaj koji se može koristiti za komunikaciju sa drugim procesom na istoj mašini ili sa pokrenutim procesom na drugim mašinama. Soketi su jedina interprocesna komunikacija koju ćemo osmatrati u ovom poglavlju, a koja dozvoljava komunikaciju između procesa na različitim računarima. Internet programi kao što su Telnet, rlogin, FTP, talk, i World Wide Web koriste sokete.

Na primer, možete dobiti WWW stranicu sa Web servera koristeći Telnet program zato što oba koriste sokete za mrežnu komunikaciju.[4] Da bi otvorili konekciju ka WWW serveru na www.codesourcery.com, koristite telnet www.codesourcery.com 80. Magična konstanta 80 određuje konekciju ka Web serveru i programsko pokretanje www.codesourcery.com umesto nekog drugog procesa. Pokušajte da ukucate GET / nakon što je konekcija uspostavljena. Ova komanda šalje poruku kroz soket ka Web serveru, koji odgovara šaljući HTML kod home page-a (početne strane) i zatim zatvara konekciju, na primer:

```
% telnet www.codesourcery.com 80 Trying
206.168.99.1...
Connected to merlin.codesourcery.com (206.168.99.1). Escape character is
'"]'.  GET / <html> <head> <meta http-equiv="Content-Type"
content="text/html;  charset=iso-8859-1 ">
```

3. Zapazite da samo Windows NT može kreirati imenovane pajpove; Windows *9x* mogu formirati samo klijent konekcije.

4. Obično ćete koristiti telnet da povežete Telnet server sa udaljenim pristupima (logins). Ali telnet možete koristiti takođe za konekciju sa serverom druge vrste i onda ukucavati komentare direktno na njega.

5.5.1 Koncept soketa

Kada kreirate soket, morate odrediti tri parametra: stil komunikacije, namespace (ime prostora) i protokol.

Tip ili stil komunikacije kontroliše kako soketi tretiraju poslate podatke i određuje broj partnera za komunikaciju. Kada je podatak poslat kroz soket, on se grupiše u delove koji se nazivaju paketi. Stil komunikacije određuje kako se postupa sa ovim paketima i kako se oni adresiraju od pošiljaoca do primaoca.

■ *Konekcijski stil* garantuje isporuku svih paketa u poretku kako su i poslani. Ako su paketi izgubljeni ili je promenjen njihov redosled usled problema u mreži, primalac automatski zahteva od pošiljaoca njihovo ponovno slanje.

Konekcijski stil soket je kao telefonski poziv: Adrese pošiljaoca i primaoca su fiksne na početku komunikacije i tokom konekcije.

■ *Datagram* stil ne garantuje isporuku ili red. Paketi mogu biti izgubljeni ili preorganizovani tokom transporta kroz mrežu u zavisnosti od mrežnih grešaka ili drugih uslova. Svaki paket mora biti obeležen zajedno sa svojom destinacijom i nije garantovano da će biti isporučen. Sistem garantuje samo "najbolji trud," tako da paketi mogu nestati ili biti preorganizovani tokom transporta.

Soketi sa datagram stilom se ponašaju više kao poštanska pisma. Pošiljalac specifira adresu primaoca za svaku poruku pojedinačno.

Namespace soketa određuje kako se piše *adresa soketa*. Adresa soketa određuje jedan kraj soket konekcije. Na primer, soket adrese u ”lokalnom prostoru imena” su obična fajl imena. "Internet prostor imena ," soket adrese se sastavljaju od Internet adrese (takođe poznate kao Internet protokol adrese - *Internet Protocol address* ili IP adrese-*IP address)* hosta prikačenog na mrežu i broja porta. Port brojevi su različiti između više soketa na istom hostu.
Protokol određuje kako se prenose podaci. Neki protokoli su TCP/IP, primarni protokol koji se koristi kod Interneta; AppleTalk mrežni protokol i UNIX lokalni komunikacioni protokol (local communication protocol). Nisu podržane sve kombinacije stilova, imena prostora i protokola.

5.5.2 Sistemski pozivi

Soketi su fleksibilnijji od prethodno osmotrenih komunikacionih tehnika. Ovo su sistemski pozivi koji uključuju sokete:

socket—Kreira soket
closes—Uništava soket

connect—Kreira konekciju između dva socketa

bind—Obeležava serverski soket sa adresom

listen—Konfiguriše soket da prihvati uslove

accept—Prihvata konekciju i kreira novi soket za konekciju

Soketi su predstavljeni pomoću fajl deskriptora.

Kreiranje i uništavanje soketa

Funkcije socket i close kreiraju i uništavaju sokete, respektivno. Kada kreirate sokete specifirajte tri soket izbora: namespace, komunikacioni stil i protokol. Za namespace parametar koristite konstante koje počinju sa PF_ (skraćenica "familija protokola"). Na primer, PF_LOCAL ili PF_UNIX određuje lokalni namespace, a PF_INET određuje internet namespace. Za protokol komunikacionog stila, koristite konstante koje počinju sa SOCK_. Koristite SOCK_STREAM soket sa konekcijkim stilom, ili koristite SOCK_DGRAM za soket sa datagram stilom.

Treći parametar, protokol, određuje mehanizam nišeg reda za slanje i primanje podataka. Svaki protokol je odgovarajuć za određenu kombinaciju namespace-stil. Obzirom da obično postoji jedan najbolji protokol za svaki takav par, postavljanjem 0 obično se dobija dobar protokol. Ako je soket uspešan, on vraća fajl deskriptor za soket. Možete čitati, upisivati itd. u soket isto kao i kod drugih fajl deskriptora. Kada ste završili sa soketom, pokrenite close kako biste ga uklonili.

Poziv *connect*

Da bi kreirali konekciju između dva soketa, klijent poziva connect, specifirajući adresu soketa servera na koji se konektuje. *Klijent* je proces koji pokreće konekciju, a *server* je proces koji čeka da prihvati konekcije. Klijent poziva connect kako bi odpočeo konekciju od lokalnog soketa do soketa servera specifiranog drugim argumentom. Treći argument je dužina u bajtima adresne strukture na koju se ukazuje drugim argumentom. Formati adresa soketa se razlikuju u zavisnosti od namespace soketa.

Slanje informacija

Bilo koja tehnika za upis u fajl deskriptor se može koristiti da bi se upisivalo u soket. Pogledajte Dodatak B za diskusiju o Linux-ovim U/I funkcijama nižeg reda i o nekim posledicama njihove upotrebe. Funkcija send, koja je posebna za soket fajl deskriptore, omogućava alternativan upis sa nekoliko dodatnih mogućnosti; za informacije pogledajte man stranu.

5.5.3 Serveri

Životni ciklus servera se sastoji od kreiranja soketa konekcijskog stila, dodeljivanja adrese soketu, postavljanje poziva za osluškivanje koji omogućava konekciju sa soketima, postavljanje poziva za prihvatanje dolazne konekcije, i na kraju zatvaranje soketa. Podaci se ne čitaju ili upisuju direktno kroz server soket, već svaki put kada program prihvati novu konekciju, Linux kreira poseban soket koji koristi za transfer podataka preko te konekcije. U ovom odeljku, upoznaćemo se sa bind, listen i accept pozivima.Adresa mora biti dodeljena serverskom soketu korišćenjem bind ako klijent treba da ga pronađe. Prvi argument je soket fajl deskriptor. Drugi argument je pokazivač na soket adresnu strukturu; format zavisi od familije soket adrese. Treći argument je dužina adresne strukture u bajtima. Kada je adresa dodeljena soketu sa konekcijskim stilom, on mora pokrenuti osluškivanje kako bi ukazivao da je server. Drugi argument određuje koliko nerešenih konekcija stoji u redu za čekanje. Ako je red za čekanje pun, dodatne konekcije će biti odbijene. Ovo ne ograničava ukupan broj konekcija koje server može da opsluži; ovo ograničava samo broj konekcija koje pokušavaju da se konektuju, a koje još nisu prihvaćene.
Server prihvata konekciju od klijenta pokretanjem accept-a. Prvi argument je soket fajl deskriptor. Drugi argument ukazuje na soket adresnu strukturu, koja se popunjava sa soket adresom klijenta. Treći argument je dužina u bajtima soket adresne strukture. Server može koristiti adresu klijenta da bi odlučio da li stvarno želi da komunicira sa klijentom. Poziv za prihvat kreira novi soket za komunikaciju sa klijentom i vraća odgovarajući fajl deskriptor. Originalni server soket nastavlja da prihvata nove klijent konekcije. Da bi pročitali podatak sa sokete bez njegovog uklanjanja iz reda za čekanje, koristite recv. On koristi iste argumente kao read, plus dodatni FLAG argument. Fleg MSG_PEEK prouzrokuje da podatak bude pročitan, ali ne i uklonjen sa ulaznog reda za čekanje (queue).

5.5.4 Lokalni soketi

Soketi koji spajaju procese na istom računaru mogu koristiti lokalni namespace predstavljen sinonimima PF_LOCAL i PF_UNIX. Oni se zovu *lokalni sokketi* ili *UNIX-domain soketi.* Njihove soket adrese, određene imenom fajlova, se koriste samo kada se kreiraju konekcije.

Soket ime je određeno u strukturi sockaddr_un. Potrebno je postaviti sun_family polje na AF_LOCAL, ukazujući time da je ovo lokalni namespace. Polje sun_path određuje ime fajla koji se koristi, i može biti dugo najviše 108 bajtova. Stvarna dužina strukture sockaddr_un trebalo bi da se obračuna korišćenjem makroa SUN_LEN. Može se koristiti bilo koje fajl ime, ali proces mora imati prava upisa nad direktorijumom, koje dozvoljava dodavanje fajlova u direktorijum. Da bi se konektovao na soket, proces mora imati pravo čitanja fajla. I pored toga što različiti računari mogu deliti isti fajl sistem, samo procesi koji se izvršavaju na istom računaru mogu komunicirati preko lokalnih namespace soketa.

Jedini dopustljiv protokol za lokalni namespace je 0.

Pošto se nalazi u fajl sistemu, lokalni soket je prikazan kao fajl. Na primer, zapazite inicijal s:

```
% ls  -l /tmp/socket
srwxrwx--x      1 user     group       0 Nov 13 19:18 /tmp/socket
```

Pozovite unlink za uklanjanja lokalanog soketa kada ste završili sa njim.

5.5.5 Primer korišćenja lokalnog namespace soketa

Ilustrovaćemo sokete sa dva programa. Server program u Listingu 5.10, kreira lokalni namespace soket i čeka na konekciju preko njega. Kada primi konekciju, on čita tekstualnu poruke od konekcije i ispisuje ih sve do zatvaranja konekcije. Ako je jedna od ovih poruka "quit" servererski program uklanja soket i završava. Server soket program uzima putanju do soketa kao argument sa komandne linije

Listing 5.10 *(socket-server.c)* **Lokal namespace soket server**

```
#include <stdio.h> #include
<stdlib.h> #include
<string.h> #include
<sys/socket.h> #include
<sys/un.h> #include
<unistd.h>

  /* Čita poruke sa soketa i prikazuje ih. Nastavlja
  do zatvaranja soketa. Vraća nonzero ako klijent
  pošalje "quit" poruku, a nulu u drugom slučaju. */
int server (int client_socket){

   while (1) { int
   length; char* text;
   /* Prvo, čita dužinu tekst poruke sa soketa. Ako
   čitanje vrati nulu, klijent zatvara konekciju. */
    if (read (client_socket, &length, sizeof
   (length)) == 0)
     return 0;
   /* Alocira se bafer da bi prihvatio
   poruku. */ text = (char*) malloc
   (length);
   /* Čita sam tekst, i prikazuje ga. */
   read (client_socket, text, length);
   printf ("%s\n", text);
   /* Oslobađa bafer. */
    free (text);
   /* Ako klijent pošalje poruku "quit," sve je
   završeno. */
   if (!strcmp (text, "quit")) return 1; }


int main  (int argc,  char* const argv[])
{
```

```
   const char* const socket_name = argv[1];int socket_fd;
   struct sockaddr_un name;
   int client_sent_quit_message;

   /* Kreira soket. */
   socket_fd = socket   (PF_LOCAL,  SOCK_STREAM,  0);
   /* Ukazuje da je ovo server.    */
   name.sun_family = AF_LOCAL;
   strcpy (name.sun_path, socket_name);
   bind (socket_fd, &name, SUN_LEN (&name));
   /* Čeka – osluškuje konekcije. */
   listen (socket_fd, 5);
   /* Ponavlja prihvatanje konekcija, činići da jedan server
   radi sa svakim klijentom. Nastavlja dok klijent ne
   pošalje "quit" poruku. */
   do {
    struct sockaddr_un client_name;
    socklen_t client_name_len;
    int client_socket_fd;

/* Prihvata konekciju. */
    client_socket_fd = accept (socket_fd, &client_name,
    &client_name_len);
/* Rukuje sa konekcijom. */
    client_sent_quit_message = server (client_socket_fd);
/* Zatvara naš kraj konekcije. */
    close (client_socket_fd);
}
   while (!client_sent_quit_message);

   /* Uklanja soket fajl. */ close
   (socket_fd); unlink
   (socket_name);

   return 0; }
```

Klijentski program u Listingu 5.11. konektuje se na lokal namespace soket i šalje poruku. Putanja do soketa i poruka su određeni na komandnoj liniji.

Listing 5.11 *(socket-client.c)* **Lokal namespace soket klijent**

```
   #include <stdio.h> #include <string.h>
   #include <sys/socket.h> #include
   <sys/un.h> #include <unistd.h>
/* Upisuje u soket TEXT koji je dat putem fajl deskriptora SOCKET_FD.
*/
```

```
void write_text (int socket_fd, const char*
text) {
   /* Upisuje broj bajtova u string, uključujući i
   NUL-završetak. */
 int length = strlen (text) + 1;
 write (socket_fd, &length, sizeof (length));
 /* Upisuje string. */
write (socket_fd, text,
length); }

int main (int argc, char* const argv[])
  {
 const char* const socket_name =
 argv[1]; const char* const message =
 argv[2]; int socket_fd; struct
 sockaddr_un name;

 /*Kreira soket. */
 socket_fd = socket (PF_LOCAL, SOCK_STREAM, 0);
 /* Čuva ime servera u adresi soketa. */
 name.sun_family = AF_LOCAL;
 strcpy (name.sun_path, socket_name);
 /* Konektuje soket. */
 connect (socket_fd, &name, SUN_LEN (&name));
 /* Upisuje tekst sa komandne linije u soket. */
 write_text (socket_fd, message);
 close (socket_fd);
 return 0; }
```

Pre nego što klijent pošalje tekst poruke, on šalje dužinu teksta šaljući bajtove intidžer varijable dužine. Takođe, server čita dužinu teksta čitajući iz soketa iz intidžer varijable. Ovo omogućava serveru da alocira odgovarajuću veličinu bafera za prihvat tekst poruke pre čitanja sa soketa.

Da bi probali ovaj primer, pokrenite server program u jednom prozoru. Odredite putanju do soketa – na primer, /tmp/socket.

```
% ./socket-server /tmp/socket
```

U drugom prozoru, pokrenite klijenta nekoliko puta, određujući istu soket putanju, i poruke da ih pošaljete klijentu:% ./socket-client /tmp/socket "Hello, world." % ./socket-client /tmp/socket "This is a test."

Program server prihvata i ispisuje ove poruke. Da bi zatvorili server, pošaljite poruku "quit" sa klijenta:

```
% ./socket-client /tmp/socket "quit"
```

Program server se zatvara.

5.5.6 Internet-Domain soketi

UNIX-domain soketi se mogu koristiti samo za komunikaciju između dva procesa na istom računaru. Za razliku od prethodno navedenih, *Internet-domain soketi,* se mogu koristiti za konekciju procesa na različitim mašinama povezanih mrežom.

Soketi koji povezuju procese preko interneta koriste Internet namespace koji se predstavlja pomoću PF_INET. Najčećši protokoli su TCP/IP. *Internet protokol* (IP)*,* protokol nižeg reda koji prenosi pakete kroz Internet deleći ih i ponovo ih spajajući ako je to potrebno. On garantuje samo "best-effort" (najbolji trud) isporuku, tako da paketi mogu nestati ili biti reorganizovani tokom transporta. Svaki računar učesnik je određen korišćenjem IP broja. *Transmission Control Protocol (TCP-protokol kontrole prenosa),* oslanjajući se na IP, pruža pouzdan i uredan transport. On dozvoljava uspostavu telefonske konekcije između dva računara i zasigurava da će prenos podataka biti siguran i uredan.

DNS Imena

Obzirom da je lakše zapamtiti imena nego brojeve, *Domain Name Service (DNS)* povezuje imena kao što je www.codesourcery.com sa jedinstvenim kompjuterskim IP brojem. DNS je implementiran pomoću svetske hijararhije imena servera, s'tim što vi ne morate da razumete DNS protokol kako bi koristili host imena u svojim programima.

Internet soket adrese sadrže dva dela: broj mašine i porta. Ove informacije se čuvaju u strukturi sockaddr_in variable. Postavite sin_family polje na AF_INET da bi ukazali da je ovo Internet namespace adresa. Polje sin_addr čuva Internet adresu željene mašine u obliku 32-bitnog intidžer IP broja. Port *broj* pomaže da se razlikuje različiti soketi koji su dodeljeni računaru. Obzirom da različite mašine čuvaju višebajtne vrednosti u različitim bajt porecima, koristite htons da bi konvertovali port broj u *mrežni bajt poredak (network byte order).* Pogledajte ip man stranu za više informacija.

Da bi konvertovali ljudima čitljiva host imena, ili brojeve u standardnom označavanju tačkama (kao što je 10.0.0.1) ili DNS imena (kao što je www.codesourcery.com) u 32-bitne IP brojeve, možete koristiti gethostbyname. Ovo vraća pokazivač na stukturu hostent structure; polje h_addr field sadrži host IP broj. Pogledajte primer programa u Listingu 5.12.

Listing 5.12 ilustruje upotrebu Internet-domain soketa. Program dobija home page (početnu stranu) sa Web servera čije je host imene određeno na komandnoj liniji.

Listing 5.12 *(socket-inet.c)* **Čita sa WWW servera**

```
#include <stdlib.h> #include <stdio.h>
#include <netinet/in.h> #include
<netdb.h> #include <sys/socket.h>
#include <unistd.h> #include
<string.h>

  /* Prikazuje sadržaj home page-a sa server soketa. Vraća
  indikator uspeha. */

void get_home_page (int socket_fd)
{
 char buffer[10000]; ssize_t
```

```
        number_characters_read;

        /* Šalje HTTP GET komandu za home page. */
        sprintf (buffer, "GET /\n");
        write (socket_fd, buffer, strlen (buffer));
        /* Čita sa soketa. Poziv za čitanje možda ne može vratiti
        sve podatke u jednom trenutku, tako da nastavlja da pokušava
        dok ga ne prekinemo. */
        while (1) {
         number_characters_read = read (socket_fd, buffer, 10000);
          if (number_characters_read == 0) return;
         /* Ispisuje podatke na standardni izlaz. */
      fwrite (buffer, sizeof (char), number_characters_read, stdout);



       int main (int argc, char* const argv[]) {
        int socket_fd;
        struct sockaddr_in name;
        struct hostent* hostinfo;

        /* Kreira soket. */
        socket_fd = socket (PF_INET, SOCK_STREAM, 0);
        /* Čuva ime servera u adresu soketa. */
        name.sin_family = AF_INET;
        /* Konvertuje iz stringa u brojeve. */
        hostinfo = gethostbyname (argv[1]);
        if (hostinfo == NULL)
        return 1; else


name.sin_addr = *((struct in_addr *) hostinfo->h_addr); /* Web
servers use port 80. */ name.sin_port = htons (80);



/* Konektuje se na Web server */
   if (connect (socket_fd, &name, sizeof (struct sockaddr_in)) == -1)
   {
    perror ("connect");
return 1;
}
   /* Traži server home page (početnu stranu). */
   get_home_page (socket_fd);

   return 0; }
```

Ovaj program uzima hostname Web servera sa komandne linije (ne URL—tj. Bez "http://"). On poziva gethostbyname da bi preveo ime hosta u numeričku IP adresu i da bi zatim konektovao strim (TCP) soket na port 80 hosta. Web serveri koriste *Hypertext Transport Protocol (HTTP),* tako da program izdaje HTTP GET komandu i server odgovara šaljući tekst home stranice.

Standardni Port brojevi

Po dogovoru, Web serveri čekaju na konekcije na portu 80. Većina Internet mrežnih servisa je povezana pomoću standardnih port brojeva. Na primer, sigurni Web serveri koji koriste SSL čekaju na konekciju na portu 443, a mejl serveri (koji koriste SMTP) koriste port 25.

Na GNU/Linux sistemima, the veza između imena protokola/servisa i standardnih port brojeva je navedena u fajlu /etc/services. Prva kolona je ime protokola ili ime servisa. Druga kolona navodi broj porta i vrstu konekcije: tcp za konekcijski orjentisanu, ili udp za datagram.

Ako implementirate uobičajene mrežne servise koristeći Internet-domain sokete, koristite portove veće od 1024.

Na primer, da bi dobili home stranicu sa Web sajta www.codesourcery.com, koristite sledeće:

```
% ./socket-inet www.codesourcery.com <html> <meta http-equiv="Content-
Type" content="text/html;  charset=iso-8859-1">
```