Operativni sistemi

Datavodi

(Laboratorijska vežba II-3)

Zadatak 1

Korišćenjem programskog jezika C napisati Linux program koji kreira dva dodatna procesa. Proces roditelj i ova dva novokreirana procesa deteta su povezana datavodima u red. Proces roditelj prihvata rečenicu po rečenicu koje korisnik unosi sa tastature. Korišćenjem datavoda proces roditelj unetu rečenicu prosleđuje prvom procesu detetu. Ovaj proces proverava da li je prvi karakter u rečenici veliko slovo i ukoliko nije konvertuje ga u odgovarajuće veliko slovo. Rečenicu zatim, koristeći datavod, šalje drugom procesu detetu. Naredni proces proverava da li je poslednji karakter u rečenici tačka (.). Ukoliko nije dodaje tačku i rečenicu štampa na standardnom izlazu.

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

int main()
{
  int pd1[2];
  int pd2[2];
  char linija[80];
  int pid1, pid2;

  if (pipe(pd1) == -1)
  {
    printf("Greska prilikom kreiranja prvog datavoda!\n");
    return -1;
  }

  if (pipe(pd2) == -1)
  {
    printf("Greska prilikom kreiranja drugog datavoda!\n");
    return -1;
  }

  if (pid1 = fork() != 0)
  {
    close(pd1[0]);
    close(pd2[0]);
    close(pd2[1]);
    do
    {
      gets(linija);

      write(pd1[1], linija, strlen(linija)+1);
      printf("Prvi proces poslao u pd1: %s\n", linija);
    }
    while(strcmp(linija, "KRAJ") != 0);

    printf("Prvi proces primio KRAJ\n");
    wait(NULL);
    close(pd1[1]);
    printf("Prvi proces izlazi!\n");
    return 0;
  }
```

```
else if (pid2 = fork() != 0)
{
  printf("Kreiran drugi proces!\n");
  char linija1[80];
  close(pd1[1]);
  close(pd2[0]);

  do
  {
    read(pd1[0], linija1, 80);
    printf("Drugi proces primio: %s\n", linija1);

    linija1[0] = toupper(linija1[0]);

    write(pd2[1], linija1, strlen(linija1)+1);
    printf("Drugi proces poslao: %s\n", linija1);
  }
  while(strcmp(linija1, "KRAJ") != 0);

  printf("Drugi proces primio KRAJ\n");

  wait(NULL);

  close(pd1[0]);
  close(pd2[1]);

  printf("Drugi proces izlazi!\n");
  exit(0);
}
else
{
  char linija2[80];

  printf("Kreiran treci proces!\n");
  close(pd1[0]);
  close(pd1[1]);
  close(pd2[1]);

  do
  {
    read(pd2[0], linija2, 80);
    printf("Treci proces primio: %s\n", linija2);

    if (linija2[strlen(linija2)-1] != '.')
    {
   linija2[strlen(linija2)+1] = '\0';
   linija2[strlen(linija2)] = '.';

    }

    printf("Modifikovana recenica: %s\n", linija2);
  }
  while(strcmp(linija2, "KRAJ.") != 0);

  printf("Treci proces primio KRAJ\n");
  close(pd2[0]);
  printf("Treci proces izlazi!\n");
  exit(0);
}
```

```
}
```

Zadatak 2

Modofikovati rešenje iz prethodnog zadatka takod a su tri procesa datavodima povezani u krug. Drugi proces dete ne štampa rečenicu već je datavodom šalje procesu roditelju koji je onda štampa na standardni izlaz bez ikakve modifikacije.

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

int main()
{
  int pd1[2];
  int pd2[2];
  int pd3[2];
  char linija[80];
  int pid1, pid2;

  if (pipe(pd1) == -1)
  {
    printf("Greska prilikom kreiranja prvog datavoda!\n");
    return -1;
  }

  if (pipe(pd2) == -1)
  {
    printf("Greska prilikom kreiranja drugog datavoda!\n");
    return -1;
  }

  if (pipe(pd3) == -1)
  {
    printf("Greska prilikom kreiranja treceg datavoda!\n");
    return -1;
  }

  if (pid1 = fork() != 0)
  {
    close(pd1[0]);
    close(pd2[0]);
    close(pd2[1]);
    close(pd3[1]);
    do
    {
      gets(linija);

      write(pd1[1], linija, strlen(linija)+1);
      printf("Prvi proces poslao u pd1: %s\n", linija);

      read(pd3[0], linija, 80);
      printf("Modifikovana recenica: %s\n", linija);
    }
    while(strcmp(linija, "KRAJ.") != 0);

    printf("Prvi proces primio KRAJ\n");
    wait(NULL);
    close(pd1[1]);
```

```
  printf("Prvi proces izlazi!\n");
  return 0;
}
else if (pid2 = fork() != 0)
{
  printf("Kreiran drugi proces!\n");
  char linija1[80];
  close(pd1[1]);
  close(pd2[0]);
  close(pd3[0]);
  close(pd3[1]);

  do
  {
    read(pd1[0], linija1, 80);
    printf("Drugi proces primio: %s\n", linija1);

    linija1[0] = toupper(linija1[0]);

    write(pd2[1], linija1, strlen(linija1)+1);
    printf("Drugi proces poslao: %s\n", linija1);
  }
  while(strcmp(linija1, "KRAJ") != 0);

  printf("Drugi proces primio KRAJ\n");

  wait(NULL);

  close(pd1[0]);
  close(pd2[1]);

  printf("Drugi proces izlazi!\n");
  exit(0);
}
else
{
  char linija2[80];

  printf("Kreiran treci proces!\n");
  close(pd1[0]);
  close(pd1[1]);
  close(pd2[1]);
  close(pd3[0]);

  do
  {
    read(pd2[0], linija2, 80);
    printf("Treci proces primio: %s\n", linija2);

    if (linija2[strlen(linija2)-1] != '.')
    {
   linija2[strlen(linija2)+1] = '\0';
   linija2[strlen(linija2)] = '.';

    }
    printf("Treci proces poslao: %s\n", linija2);
    write(pd3[1], linija2, strlen(linija2)+1);
  }
  while(strcmp(linija2, "KRAJ.") != 0);
```

```
    printf("Treci proces primio KRAJ\n");
    //sleep(1);
    close(pd2[0]);
    close(pd3[1]);
    printf("Treci proces izlazi!\n");
    exit(0);
  }
}
```

Zadatak 3

Korišćenjem programskog jezika C napisati Linux program demonstrira mogućnosti redirekcije standardnog ulaza i izlaza. Program prihvata dva ulazna argumenta: putanju do izvršne datoteke i putanju do tekstualne datoteke. Program otvara tekstualnu datoteku i redirektuje standardni izlaz na tu otvorenu datoteku. Zatim kreira novi proces i učitava specificiranu izvršnu datoteku. Efekat ovog programa je da se izlaz svakog pozvanog programa redirektuje u izabranu tekstualnu datoteku.

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/file.h>

int main(int argc, char* argv[])
{
  int fd;
  // Otvara tekstualnu datoteku
  fd = open(argv[2], O_CREAT | O_TRUNC | O_RDWR, 0777);
  // Redirektuje standardni izlaz na otvorenu tekstualnu datoteku
  dup2(fd, 1);
  // Zatvara deskriptor za datoteku (kopija ostaje)
  close(fd);
  // Poziva izvrsnu datoteku
  execlp(argv[1], argv[1], NULL);
  printf("Ne bi trebalo nikada da se izvrsi!");
}
```

Zadatak 4

Korišćenjem programskog jezika C napisati Linux program koji kao ulazni argument prihvata putanju do jedne izvršne datoteke. Program treba da pozove na izvršenje tako prosleđenu izvršnu datoteku i njen izlaz prikazuje na standardnom izlazu ekran po ekran. Ovo se postiže tako što se izlaz pozvane izvršne datoteke prosleđuje programu **more** korišćenjem datavoda.

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

int main(int argc, char* argv[])
{
  int pd[2];
  int pid;
  char* args[10];
  int i;
```

```
  if ( pipe(pd) < 0 )
    exit(1);
  pid = fork();
  if ( pid == 0 )
  {
    close(1);
    dup(pd[1]);

    for (i=1; i < argc; i++)
      args[i-1] = argv[i];
    args[argc - 1] = NULL;

    //execlp(argv[1], argv[1], NULL);
    execvp(args[0], args);
  }
  else
  {
    close(0);
    dup(pd[0]);
    execlp("more", "more", NULL);
  }
}
```

Zadatak 5

Korišnjenjem programskog jezika C kreirati dva Linux procesa koji međusobno komuniciraju korišćenjem daavoda. Jedan proces čita red po red proizvoljne datoteke i šalje ih korišćenjem datavoda drugom procesu. Drugi proces prihvata podatke iz datavoda i štampa ih na standardnom izlazu.

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

//Maksimalna velicina bafera za slanje i prihvatanje poruka
#define MAX_BUF 80

int main(int argc, char * argv[])
{
   int pid;
   int pd[2];
   char buffer[MAX_BUF];
   FILE * f;
   int count = 0;

   //Najpre se kreira datavod. U slucaju greske program se prekida
   if (pipe(pd) < 0)
   {
         printf("Doslo je do greske prilikom kreiranja datavoda.\n");
         exit(1);
   }

   //Kreira se proces dete sa kojim ce ici komunikacija
   pid = fork();

   //U slucaju greske program se prekida
   if (pid < 0)
   {
```

```
        printf("Doslo je do greske prilikom kreiranja novog procesa.\n");
        exit(1);
   }

   //Deo koda koji izvrsava proces dete
   if (pid == 0)
   {
        //Proces dete samo upisuje podatke pa zatvara kraj datavoda za citanje
        close(pd[0]);

        //Otvara se datoteka iz koje se citaju podaci
        f = fopen("podaci", "r");

        fgets(buffer, MAX_BUF, f);
        while(!feof(f))
        {
              write(pd[1], buffer, MAX_BUF);

              fgets(buffer, MAX_BUF, f);
        }

        //Kraj komunikacija pa se zatvara i kraj datavoda za pisanje
        close(pd[1]);
   }
   else
   {    //Deo koda koji izvrsava proces roditelj
        //Proces roditelj cita podatke iz datavoda pa se zatvara kraj datavoda
za pisanje
        close(pd[1]);

        while(1)
        {
              //Citamo  podatek  iz  datavoda.  Funkcija  vraca  broj  ocitanih
bajtova
              count = read(pd[0], buffer, MAX_BUF);

              //Ako je ocitano 0B znaci da je komunikacija zavrsena
              if (count == 0)
                    break;

              printf("%s", buffer);
        }

        //Komunikacija je zavrsena pa zatvaramo i kraj datavoda za citanje
        close(pd[0]);
   }
}
```

Druga varijanta:

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

//Maksimalna velicina bafera za slanje i prihvatanje poruka
#define MAX_BUF 80

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

```
   int pid;
   int pd[2];
   char buffer[MAX_BUF];
   int count = 0;

   //Najpre se kreira datavod. U slucaju greske program se prekida
   if (pipe(pd) < 0)
   {
         printf("Doslo je do greske prilikom kreiranja datavoda.\n");
         exit(1);
   }

   //Kreira se proces dete sa kojim ce ici komunikacija
   pid = fork();

   //U slucaju greske program se prekida
   if (pid < 0)
   {
         printf("Doslo je do greske prilikom kreiranja novog procesa.\n");
         exit(1);
   }

   //Deo koda koji izvrsava proces dete
   if (pid == 0)
   {
         //Proces dete samo upisuje podatke pa zatvara kraj datavoda za citanje
         close(pd[0]);

         //Deskriptor kraja datavoda koji se koristi za pisanje konvertujemo u
string
         sprintf(buffer, "%d", pd[1]);

         //Ucitavamo drugu izvrsnu datoteku
         execl("II-3-5b", "II-3-5b", buffer, NULL);

         //Ukoliko se izvrsi ovaj deo koda znaci da je doslo do greske prilikom
ucitavanja izvrsne datoteke
         printf("Doslo je do greske prilikom ucitavanja izvrsne datoteke\n");
         close(pd[1]);
         exit(1);
   }
   else
   {     //Deo koda koji izvrsava proces roditelj
         //Proces roditelj cita podatke iz datavoda pa se zatvara kraj datavoda
za pisanje
         close(pd[1]);

         while(1)
         {
               //Citamo  podatak  iz  datavoda.  Funkcija  vraca  broj  ocitanih
bajtova
               count = read(pd[0], buffer, MAX_BUF);

               //Ako je ocitano 0B znaci da je komunikacija zavrsena
               if (count == 0)
                     break;

               printf("%s", buffer);
         }
```

```
        //Komunikacija je zavrsena pa zatvaramo i kraj datavoda za citanje
        close(pd[0]);
   }
}
```

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

//Maksimalna velicina bafera za slanje i prihvatanje poruka
#define MAX_BUF 80

int main(int argc, char * argv[])
{
   int pd1;
   char buffer[MAX_BUF];
   FILE * f;


   //Najpre se proverava da li ima dovoljno ulaznih argumenatareira se proces
dete sa kojim ce ici komunikacija
   if (argc < 2)
   {
         printf("Nema dovoljno ulaznih argumenata\n");
         exit(1);
   }

   //U slucaju da je sve u redu sa ulaznim argumentima, argument sa pozicije 1,
koji predstavlja deskriptor datavoda, se konvertuje u int
   pd1 = atoi(argv[1]);

   //Deskriptor datavoda mora da bude vrednost razlicita od 0
   if (pd1 == 0)
   {
         printf("Preneta je pogresna vrednost za deskriptor datavoda.\n");
         exit(1);
   }



   //Otvara se datoteka iz koje se citaju podaci
   f = fopen("podaci", "r");

   fgets(buffer, MAX_BUF, f);
   while(!feof(f))
   {
         write(pd1, buffer, MAX_BUF);

         fgets(buffer, MAX_BUF, f);
   }

   //Kraj komunikacija pa se zatvara i kraj datavoda za pisanje
   close(pd1);

}
```

Zadatak 6

Modifikovati jednostavnu implementaciju shell-a (iz laboratorijske vežbe II-1) dodavanjem podrške

za pipe (|) odnosno mogućnost preusmeravanja izlaza jedne komande na ulaz druge. Uzeti da najviše dve komande mogu da budu povezane pipe-om.

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

//maksimalan broj argumenata komande koju shell poziva
#define NUMARGS 20

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

   char komanda[100];     /*Komanda koju korisnik zadaje i koja se cita sa
standardnog ulaza*/
   char *argumenti[20]; /*Ulazni argumenti komande koju je korisnik zadao*/
   int  noArgs;          /*Broj ulaznih argumenata komande koju je korisnik
zadao*/
   int pipePos;          /*Pozicija simbola | ukoliko se on nalazi u zadatoj
komandi*/
   int retStatus;        /*Povratna vrednost sa kojom se komanda zavrsila */
   int pd[2];       /*Krajevi datavoda koji ce se koristiti za komunikaciju
izmedju dva procesa*/
   char *argumenti1[20];/*Pomocni niz ulaznih argumenata */
   int i;

   /* Prikazuje se prompt*/
   printf("\nprompt> ");

   /*Cita se komanda*/
   gets(komanda);

   /* Argumenti komande se izdvajaju kao tokeni. Za izdvajanje argumenata se
koristi funkcija strtok. */
   noArgs = 0;
   argumenti[noArgs++] = strtok(komanda, " ");
   while ( ( argumenti[noArgs++] = strtok(0, " ") ) != NULL )
   {}

   /*Ukoliko je zadata komanda LOGOUT shell zavrsava sa svojim radom*/
   while (strcmp(komanda, "logout") != 0 )
   {

         /*Utvrdjuje se da li se medju argumentima komande nalazi simbol | */
         pipePos = 0;
         while ((pipePos < noArgs - 1) && (strcmp(argumenti[pipePos], "|") !=
0))
               pipePos++;

         /*Naisli smo na simbol | . To znacu da postoje dve komande koje treba
da izvrsimo i kod kojih treba da izvrsimo preusmeravanja izlaza jedne na ulaz
druge*/
         if (pipePos < noArgs - 1)
         {
               /*Kreira se datavod*/
               pipe(pd);

               /*Prva komanda*/
```

```
                if (fork() == 0)
                {
                      sleep(5);

                      for (i = 0; i < pipePos; i++)
                            argumenti[i] = argumenti[i];

                      argumenti1[pipePos] = NULL;

                      /*Zatvara se kraj datavoda koji se koristi za citanje */
                      close(pd[0]);

                      /*Standardni  izlaz  se  preusmerava  na  ulaz  datavoda  za
pisanje*/
                      close(1);
                      dup(pd[1]);

                      /*Ucitava se izvrsna datoteka*/
                      //execvp(argumenti1[0], argumenti1);
                      execlp("ls", "ls", NULL);
                      printf("Greska prilikom izvrsavanja komande\n");
                      exit(-1);
                }

                /*Druga komanda*/
                if (fork() == 0)
                {
                      sleep(5);

                      for (i = pipePos + 1; i < noArgs; i++)
                            argumenti1[i - pipePos - 1] = argumenti[i];


                      /*Zatvara se kraj datavoda za pisanje*/
                      close(pd[1]);

                      /*Standardni  ulaz  se  preusmerava  na  kraj  datavoda  za
citanje*/
                      close(0);
                      dup(pd[0]);

                      /*Ucitava se izvrsna datoteka*/
                      execvp(argumenti1[0], argumenti1);
                      //execlp("more", "more", NULL);
                      printf("Gresak prilikom izvrsavanja komande\n");
                      exit(-1);
                }

                close(pd[0]);
                close(pd[1]);

          }
          /*Ako nismo nasli | sve se izvrsava kao i ranije*/
          else if (fork() == 0)
          {
          /* U novom procesu shell izvrsava zadatu komandu odgvarajucom verzijom
exec
           *  komande. Ukoliko exec komanda vrati neki kod to znaci da je doslo
do greske i
```

```
          *  treba eksplicitno prekinuti novi proces komandom exit.
          * U novom procesu shell izvrsava zadatu komandu */
               sleep(5);

         /*Ukoliko je na kraju komande zadat & treba ga ukloniti iz vektora
argumenata koji se prosledjuj novom procesu*/
               if ( strcmp(argumenti[noArgs - 2], "&") == 0 )
                     argumenti[noArgs - 2] = NULL;

               execvp(argumenti[0], argumenti);
               printf("\nGRESKA PRI IZVRSENJU KOMANDE!\n");
               exit(-1);

         }


         if (strcmp(argumenti[noArgs - 2], "&") != 0 && pipePos < noArgs)
         {
               wait(NULL);
               wait(NULL);
         }
         else if ( strcmp(argumenti[noArgs - 2], "&") != 0 )
         /* Ukoliko je poslednji navedeni argument nije & treba sacekati da se
novi proces
          *  zavrsi pre prikaza prompt-a. */
               wait(&retStatus);

         /* Ponoviti prikazivanje prompta i izdvajanje argumenata */
         printf("\nprompt> ");
         gets(komanda);

         noArgs = 0;
         argumenti[noArgs++] = strtok(komanda, " ");
         while ( ( argumenti[noArgs++] = strtok(0, " ") ) != NULL )
         {}
   }
}
```