Operativni sistemi

Upravljanje procesima i nitima

(Laboratorijska vežba II-1)

Zadatak 1

Korišćenjem programskog jezika C napisati Linux program koji treba da simulira funkcionisanje jednostavnog shell-a. Ovaj shell treba da izvršava programe koji se mogu naći u tekućem direktorijumu ili u bilo kom direktorijumu koji je naveden u PATH promenljivoj okruženja. Takođe, omogućeno je da se navođenjem znaka & kao poslednjeg argumenta komandne linije shell pozvani program izvršava u pozadini, a odmah prikaže prompt koji je spreman da primi novu komandu.

```
#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 retStatus;        /*Povratna vrednost sa kojom se komanda zavrsila */

   /* 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 )
   {

         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);

        }
        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 )
        {}
   }
}
```

Zadatak 2

Modifikovati rešenje iz Zadatka 1 tako da novokreirani shell može koristiti batch datoteke. To su obične tekstualne datoteke u kojima je u svakoj liniji po jedna komanda koju shell treba da izvrši.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
//maksimalan broj argumenata komande koju shell poziva
#define NUMARGS 20

/*Funkcija prihvata komandu i obradjuje je. U slucaju da je komanda logout
funkcija vraca 1. U svim ostalim slucajevima vraca 0*/
int obrada(char * komanda)
{
   char *argumenti[20]; /*Ulazni argumenti komande koju je korisnik zadao*/
   int  noArgs;          /*Broj ulaznih argumenata komande koju je korisnik
zadao*/
   int retStatus;        /*Povratna vrednost sa kojom se komanda zavrsila */
   int pid;

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

```
   if (strcmp(komanda, "logout") != 0)
   {
         pid = fork();
         if ( pid == 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);

         }
         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);

               return 0;
         }

   }
   else
         return 1;

}

int obrada dat(char * batch)
{

   FILE * f;
   char komanda[100];   /*Komanda koja se cita iz batch datoteke*/
   int ret = 0;

   f = fopen(batch, "r");

   /*Ukoliko je datoteka uspesno otvorena cita se linija po linija i izvrsavaju
se kao standardne komande */
   if (f != 0)
   {
         fgets(komanda, 100, f);
         while(!feof(f))
         {
               ret = obrada(komanda);/*Svaka linija batch datoteke se izvrsava
kao standardna datoteka */
               /*Ako je obradjena komanda logout prekida se dalja obrada batch
```

```
datotke*/
               if (ret == 1)
                     return ret;
               fgets(komanda, 100, f);
         }

   }
   else
         printf("Greska prilikom otvaranja batch datoteke\n");

   return ret;

}

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

   char komanda[100];     /*Komanda koju korisnik zadaje i koja se cita sa
standardnog ulaza*/
   int ret = 0;

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

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

         /*Batch datoteka se zadaje tako sto se na pocetku doda karakter @. U
toj situaciji se poziva funkcija za obradu batch datoteke */
         if (komanda[0] == '@')
         {
               /*Uklanjamo znak @ na pocetku i uzimamo samo prvi token iz
komande */
               char * batch = strtok(komanda + 1, " ");

               ret = obrada_dat(batch);
         }
         else
         /*Nije batch datoteke. Ide standardna obrada */
               ret = obrada(komanda);

   }
   /*Petlja se izvrsava sve dok neka od funkcija ne vrati 1 sto je znak da je
specificirana logout komanda */
   while (ret == 0);
}
```

Zadatak 3

Modifikovati rešenje iz Zadatka 1 tako da se sve akcije shell-a izvršavaju u zasebnim nitima.

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

//maksimalan broj argumenata komande koju shell poziva
```

```
#define NUMARGS 20

/*Funkcija prihvata komandu i obradjuje je. U slucaju da je komanda logout
funkcija vraca 1. U svim ostalim slucajevima vraca 0*/
void * obrada(void * arg)
{
   char *argumenti[20]; /*Ulazni argumenti komande koju je korisnik zadao*/
   int  noArgs;          /*Broj  ulaznih  argumenata  komande  koju  je  korisnik
zadao*/
   int retStatus;        /*Povratna vrednost sa kojom se komanda zavrsila */
   int pid;
   char * komanda = (char *)arg;

   /* 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 )
   {}

   if (strcmp(komanda, "logout") != 0)
   {
         pid = fork();
         if ( pid == 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);

         }
         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);

               return 0;
         }

   }
   else
         return 1;

}
```

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

   char komanda[100];     /*Komanda koju korisnik zadaje i koja se cita sa
standardnog ulaza*/
   int ret = 0;
   pthread_t nit;

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

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

         /*Kreira se nit za obradu komande koju je shell prihvatio*/
         pthread_create(&nit, NULL, (void *)obrada, (void *)komanda);

         pthread_join(nit, &ret);

   }
   /*Petlja se izvrsava sve dok neka od funkcija ne vrati 1 sto je znak da je
specificirana logout komanda */
   while (ret == 0);
}
```

Zadatak 4

Korišćenjem programskog jezika C napisati Linux program koji na standardnom izlazu štampa sve svoje ulazne argumente ukoliko ih ima. Kreirati drugi program koji poziva prethodni program i prosleđuje mu sve svoje argumente (za kreiranje iskoristiti sistemske pozive execl i execv).

```
/*Prvi program koji sve svoje ulazne argumente stampa na standardnom izlazu.
Program je implementiran u datoteci II-1-4a.c*/
#include <stdio.h>
#include <stdlib.h>

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

   /*U petlji se stampaju svi ulazni argumenti programa*/
   for (i = 0; i < argc; i++)
         printf("argv[%d] = %s\n", i, argv[i]);

   return 0;
}
```

```
/*Drugi program koji poziva prvi program i prosledjuje mu sve svoje ulazne
argumente. Program je implementiran u datoteci II-1-4b.c*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_ARGS 20
#define MAX_ARG_LENGTH 100
```

```
int main(int argc, char * argv[])
{
   int i;
   char * argumenti[MAX_ARGS];

   /*Argument na poziciji 0 je uvek jednak imenu programa koji pozivamo. */
   argumenti[0] = (char *)malloc(MAX_ARG_LENGTH * sizeof(char));
   strcpy(argumenti[0], "II-1-4a");

   /*Sve ostale ulazne argumente (osim nultog) kopiramo u vektor sa argumetnima
koji cemo proslediti programu koji pozivamo*/
   for (i = 1; i < argc; i++)
   {
         argumenti[i] = (char *)malloc(MAX_ARG_LENGTH * sizeof(char));
         strcpy(argumenti[i], argv[i]);
   }

   /*Poslednji argument mora da bude NULL da bi ukazao na kraj vektora sa
argumentima*/
   argumenti[argc] = NULL;

   if (fork() == 0)
   {/*Izvrsava  novokreirani proces*/
         if (execv("./II-1-4a", argumenti) < 0)
         {
               printf("Doslo je do greske!\n");
               exit(1);
         }
   }
   else
   /*Izvrsava proces roditelj koji ceka da se dete zavrsi*/
         wait(NULL);

   /*Oslobadjamo memoriju koja je zauzeta za vektor argumenata */
   for (i = 0; i < argc; i++)
         free(argumenti[i]);

   return 0;
}
```

Zadatak 5

Korišćenjem programskog jezika C napisati Linux program koji iz tekućeg direktorijum kopira proizvoljnu datoteku u direktorijum tmp koji se nalazi u tekućem direktorijumu.

```
/*Program  koji  iz  tekuceg  direktorijuma  kopira  proizvoljnu  datoteku  u
direktorijum tmp koji se nalazi u tekucem direktorijumu*/
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
   int pid;
   char datoteka[100];
   /*Sa standardnog ulaza citamo ime datoteke koju zelimo da kopiramo*/
   scanf("%s", datoteka);

   pid = fork();
```

```
   if (pid == 0)
   {/*Izvrsava proces dete koji ce obaviti kopiranje
   Kopiranje se vrsi komandom: cp ime_dat ./tmp
   Koristimo funkciju execlp jer ulazne argumente procesa koji pozivamo
prosledjujemo u vidu liste a izvrsna datoteka (Linux komanda cp) se nalazi
   negde na putanji koja je definisana environment promenljivom PATH. Prvi
argument je ime izvrsne datoteke koju pozivamo, drugi argument odgovara argv[0]
i
        opet je ime datoteke koju pozivamo, treci argument je argv[1] i
predstavlja datoteku koju kopiramo, cetvrti argument odgovara argv[2] i
predstavlja
        putanju gde kopiramo i peti argument je NULL da bi ukazao na kraj liste
argumenata*/
         if (execlp("cp", "cp", datoteka, "./tmp", NULL) < 0)
         {
               printf("Doslo je do greske!\n");
               exit(1);
         }
   }
   else
   /*Izvrsava proces roditelj koji ceka da se njegov proces dete zavrsi odnosno
da se kopiranje izvrsi*/
         wait(NULL);

   return 0;
}
```

```
/*Program koji iz tekuceg direktorijuma kopira proizvoljnu datoteku u
direktorijum tmp koji se nalazi u tekucem direktorijumu*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>


int main(int argc, char * argv[])
{
   int i;
   char * argumenti[4];

   int pid;
   char datoteka[100];
   /*Sa standardnog ulaza citamo ime datoteke koju zelimo da kopiramo*/
   scanf("%s", datoteka);

   pid = fork();
   if (pid == 0)
   {/*Izvrsava proces dete koji ce obaviti kopiranje
   Kopiranje se vrsi komandom: cp ime_dat ./tmp */

         /*Argument na poziciji 0 je uvek jednak imenu programa koji pozivamo.
*/
         argumenti[0] = (char *)malloc(100 * sizeof(char));
         argumenti[0] = "cp";

         /*Argument na poziciji 1 je datoteka koju kopiramo */
         argumenti[1] = (char *)malloc(100 * sizeof(char));
         strcpy(argumenti[1], datoteka);
```

```
        /*Argument na poziciji 2 je direktorijum u koji kopiramo */
        argumenti[2] = (char *)malloc(100 * sizeof(char));
        argumenti[2] = "./tmp";

        /*Poslednji argument mora da bude NULL da bi ukazao na kraj vektora sa
argumentima*/
        argumenti[3] = NULL;

   /*Koristimo funkciju execvp jer ulazne argumente procesa koji pozivamo
prosledjujemo u vidu vektora a izvrsna datoteka (Linux komanda cp) se nalazi
   negde na putanji koja je definisana environment promenljivom PATH. Prvi
argument je ime izvrsne datoteke koju pozivamo, drugi argument je vektor
   sa ulaznim argumentima koje prosledjujemo programu koji pozivamo. */
        if (execvp("cp", argumenti) < 0)
        {
              printf("Doslo je do greske!\n");
              exit(1);
        }
   }
   else
   /*Izvrsava proces roditelj koji ceka da se njegov proces dete zavrsi odnosno
da se kopiranje izvrsi*/
        wait(NULL);


   return 0;
}
```

Zadatak 6

Korišćenjem programskog jezika C napisati Linux koji sortira vrste celobrojne matrice čiji se elementi unose sa tastature. Za sortiranje svake vrste pokrenuti posebnu nit.

```
/*Program koji sortira vrste celobroje matrice pri cemu se za sortiranje svake
vrste pokrece posebna nit*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define N 4

/*matrica koju sortiram je deklarisana kao globalna promenljiva da bi bila
vidljiva u svim funkcijama pa samim tim i u funkciji niti*/
int a[N][N];

/*Funkcija niti*/
void * sortiranje(void * arg)
{
   /*Ulazni argument je redni broj vrste koju ce nit sortirati. Posto je ulazni
argument void * potrebno ga je kastovati u int * a zatim ocitati vrednost*/
   int k = *((int *)arg);
   int i, j, pom;

/*Sortiranje matrice */
   for (i = 0; i < N - 1; i++)
        for(j = i + 1; j < N; j++)
              if (a[k][i] > a[k][j])
              {
                    pom = a[k][i];
```

```
                    a[k][i] = a[k][j];
                    a[k][j] = pom;
                }

}

int main()
{
   int i, j;

   /*Niz identifikatora niti. Po jedan za svaku vrstu*/
   pthread_t niti[N];

   /*Sa standardnog ulaza citamo elemente matrice*/
   for(i = 0; i < N; i++)
         for(j = 0; j < N; j++)
               scanf("%d", &a[i][j]);

   /*Kreiramo niti */
   for (i = 0; i < N; i++)
         pthread_create(&niti[i], NULL, (void *)sortiranje /*funkcija nit*/,
(void *)&i /*ulazni argument niti odnosno u ovom slucaju redni broj vrste*/);

   /*Main f-ja ceka da se niti zavrse*/
   for (i = 0; i < N; i++)
         pthread_join(niti[i], NULL);

   /*Na standardni izlaz stampamo sortiranu matricu*/
   for(i = 0; i < N; i++)
   {
         for(j = 0; j < N; j++)
               printf("%d\t", a[i][j]);
         printf("\n");
   }

   return 0;
}
```

Zadatak 7

Korišćenjem programskog jezika C napisati Linux koji na standardnom izlazu ispisuje rečenicu "Ovo je test za niti!" pri čemu svaku reč u rečenici ispisuje posebna nit.

```
/*Program koji na standardnom izlazu ispisuje recenicu "Ovo je test za niti!"
pri cemu svaku rec ispisuje posebna nit*/
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>

#define N 5

/*Niz koji sadrzi reci koje se stampaju deklarisemo kao globalnu promenljivu
kako bi bile vidljive u svim funkcijama pa i u funkciji niti*/
char reci[N][20] = {"Ovo", "je", "test", "za", "niti!"};

/*Struktura koja definise ulazne argumente niti*/
struct argument
{
```

```
   /*Redni broj reci koja se stampa*/
   int rec;
        /*Nit koja treba da odstampa prethodnu rec i koja se zbog toga ceka*/
   pthread_t prethodna_nit;
};

/*Funkcija niti*/
void * stampanje(void * arg)
{
   /*Ulazni argument je struktura kojaniti[i]; sadrzi neophodne informacije za
stampanje*/
   struct argument * p = (struct argument *)arg;

   /*samo nit koja stampa prvu rec ne ceka prethodnu nit. Sve ostale niti cekaju
nit koja stampa prethodnu rec*/
   if (p->rec > 0)
         pthread_join(p->prethodna_nit, NULL);

   printf(" %s", reci[p->rec]);
   sleep(3);

}

int main()
{

   int i;
   /*Niz ulaznih argumenata niti. Po jedan za svaku nit.*/
   struct argument args[N];
   /*Niz identifikatora niti. Po jedan za svaku rec*/
   pthread_t niti[N];

   /*Kreiramo niti */
   for (i = 0; i < N; i++)
   {
         args[i].rec = i;
         if (i > 0)
               args[i].prethodna_nit = niti[i - 1];
         pthread_create(&niti[i],  NULL,  (void  *)stampanje  /*funkcija  nit*/,
(void *)&args[i] /*ulazni argument niti */);
   }

   /*Main f-ja ceka da se poslednja nit zavrsi*/
   pthread_join(niti[N-1], NULL);

   return 0;
}
```