Predavanje 4

Ethernet komunikacija, manipulisanje procesima

Na primeru jednostavne server/klijent aplikacije upoznaćemo rad sa socket-ima, a usput i savladati kreiranje novih procesa i manipulisanje njima.

server:

server.c

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



/*ovo je jezgro serverske funkcionalnosti-ova funkcija se poziva
nakon uspostavljanja veze sa klijentom kako bi klijentu bilo
omoguceno da komunicira sa serverom. */
void doprocessing (int sock)
{
    /*lokalne promenljive*/
    int n;
    char buffer[256];
    char sendBuf[256];
    char tempstr[256];

    bzero(buffer,256);
    int done = 0;


    while (!done)
    {
       n = read(sock,buffer,255);
       buffer[n] = 0;//terminiraj string primljen od strane klijenta
       if (strcmp(buffer,"quit") == 0){
              done = 1;
              printf("Client closed connection..\n");
         }
       else
       {
              printf("Received: %s\n",buffer);
       }

    }
    close(sock);
}
```

```
/* glavni program serverske aplikacije */
int main( int argc, char *argv[] )
{
    int sockfd, newsockfd, portno, clilen;
    char buffer[256];
    struct sockaddr_in serv_addr, cli_addr;
    int  n;

    /* najpre se poziva uvek socket() funkcija da se registruje socket:
       AF_INET je neophodan kada se zahteva komunikacija bilo koja
       dva host-a na Internetu;
       Drugi argument definise tip socket-a i moze biti SOCK_STREAM ili SOCK_DGRAM:
       SOCK_STREAM odgovara npr. TCP komunikaciji, dok SOCK_DGRAM kreira npr. UDP kanal
       Treci argument je zapravo protokol koji se koristi: najcesce se stavlja 0
        sto znaci da OS sam odabere podrazumevane protokole za dati tip socket-a (TCP za
       SOCK_STREAM  ili UDP za SOCK_DGRAM)  */
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
    {
        perror("ERROR opening socket");
        exit(1);
    }

    /* Inicijalizacija strukture socket-a */
    bzero((char *) &serv_addr, sizeof(serv_addr));
    portno = 5001;
    serv_addr.sin_family = AF_INET; //mora biti AF_INET
    /* ip adresa host-a. INADDR_ANY vraca ip adresu masine na kojoj se startovao server*/
    serv_addr.sin_addr.s_addr = INADDR_ANY;
    /* broj porta-ne sme se staviti kao broj vec se mora konvertovati u
       tzv. network byte order funkcijom htons*/
    serv_addr.sin_port = htons(portno);



    /* Sada bind-ujemo adresu sa prethodno kreiranim socket-om */
    if (bind(sockfd, (struct sockaddr *) &serv_addr,
                          sizeof(serv_addr)) < 0)
    {
         perror("ERROR on binding");
         exit(1);
    }
    printf("Server started.. waiting for clients ...\n");

    /* postavi prethodno kreirani socket kao pasivan socket
       koji ce prihvatati zahteve za konekcijom od klijenata
       koriscenjem accept funkcije */
    listen(sockfd,5); //maksimalno 5 klijenata moze da koristi moje usluge
    clilen = sizeof(cli_addr);

    while (1)
    {
       /*ovde ce cekati sve dok ne stigne zahtev za konekcijom od prvog klijenta*/
        newsockfd = accept(sockfd,
                 (struct sockaddr *) &cli_addr, &clilen);
       printf("Client connected...\n");
        if (newsockfd < 0)
        {
            perror("ERROR on accept");
            exit(1);
        }

        /* Kreiraj child proces sa ciljem da mozes istovremeno da
       komuniciras sa vise klijenata */
        int  pid = fork();
        if (pid < 0)
        {
            perror("ERROR on fork");
           exit(1);
        }
```

```
        if (pid == 0)
        {
            /* child proces ima pid 0 te tako mozemo znati da li
             se ovaj deo koda izvrsava u child ili parent procesu */
            close(sockfd);
            doprocessing(newsockfd);
            exit(0);
        }
        else
        {
             /*ovo je parent proces koji je samo zaduzen da
             delegira poslove child procesima-stoga ne moras
             da radis nista vec samo nastavi da osluskujes
             nove klijente koji salju zahtev za konekcijom*/
            close(newsockfd);
        }
    } /* end of while */
}
```

Klijent:

client.c

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

/* definisanje globalnih promenljivih */

int main(int argc, char *argv[])
{
    int sockfd = 0;
    struct sockaddr_in serv_addr;
    int done;
    char * line = NULL;
    size_t len = 0;
    int nread;
    /* klijentska aplikacija se poziva sa ./ime_aplikacija ip_adresa_servera */
    if(argc != 2)
    {
        printf("\n Usage: %s <ip of server> \n",argv[0]);
        return 1;
    }


    /* kreiraj socket za komunikaciju sa serverom */
    if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
       printf("\n Error : Could not create socket \n");
       return 1;
    }
    memset(&serv_addr, 0, sizeof(serv_addr));

    /* kao i podatke neophodne za komunikaciju sa serverom */
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(5001);

    /* inet_pton konvertuje ip adresu iz stringa u format
       neophodan za serv_addr strukturu */
    if(inet_pton(AF_INET, argv[1], &serv_addr.sin_addr)<=0)
    {
        printf("\n inet_pton error occured\n");
        return 1;
    }

    /* povezi se sa serverom definisanim preko ip adrese i porta */
    if (connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\n Error : Connect Failed \n");
        return 1;
    }
    printf("Connected to server... Send message to server, or type 'quit' to exit\n");

    /* udji u petlju u kojoj ces slati poruke server sve dok ne posaljes "quit" */
    done = 0;
    while (!done){
       nread=getline(&line, &len, stdin);
       //ignore '\n'
       nread--;
       line[nread] = '\0';
       if (strcmp(line,"quit") == 0){
              done = 1;
              printf("Closing connection with the server..\n");
        }
       /* posalji poruku serveru */
       write (sockfd, line, strlen(line));
    }
    close(sockfd);
    return 0;
}
```