1 Klijentski deo koda

client.c

```
/*
 * Ovo je je C kod clienta koji odgovara
 * prostom ECHO serveru
 *
 * Koristi sockete za komunikaciju,
 * errno sistem za ispisivanje
 * obavestenja i gresaka
 *
 * Ima dva bafera, pri cemu jedan
 * prihvata korisnicki unos i sluzi
 * za slanje tih podataka preko socketa
 * dok drugi ima fiksnu velicinu i
 * sluzi za prihvatanje podataka iz
 * sistemskog bafera
 *
 * Za citanje sa stdin ulaza, koristi
 * se getline funkcija, jer scanf string
 * od vise reci deli prema razmaku, pa
 * se zbog toga i komunikacija poremeti
 * i izgubi se potpuna kontrola u pogledu
 * ko je sledeci koji salje a ko prima podatke.
 *
 * Ne moramo da vodimo racuna o '\0'
 * karakteru na kraju stringa posto
 * se na njegovom kraju uvek nadje '\n'
 * zbog nacina na koji se snima unos
 * korisnicke poruke
 *
 * Kodirao: Ikac, e-mail: ikac.ikax@gmail.com
 */

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

int sockfd;

void sigpipe()
{
```

```
  //kada klijent pokusa da pise u socket koji je
  //sa druge strane zatvoren, ispisace informaciju
  //i zatvoriti svoju stranu
 printf("Zatvaram konekciju");
 close(sockfd);
}

int main(int argc, char *argv[])
{
 int portno, n;
 struct sockaddr_in serv_addr;
 struct hostent *server;

 signal(SIGPIPE, sigpipe);

 if (argc < 3) {
  fprintf(stderr, "uputstvo: %s hostname port\n",argv[0]);
  exit(1);
 }

 portno = atoi(argv[2]);

 sockfd = socket(AF_INET, SOCK_STREAM, 0);
 perror("Otvaranje socket-a");
 if (sockfd < 0)
  exit(1);

 server = gethostbyname(argv[1]);
 perror("Trazenje servera");

 if (!server)
  exit(1);

 memset(&serv_addr, '0', sizeof(serv_addr));
 serv_addr.sin_family = AF_INET; //koriste se Internet adrese
 memcpy(&serv_addr.sin_addr.s_addr, server->h_addr, server->
     h_length); //uzima se adresa servera
 serv_addr.sin_port = htons(portno); //uzima se port naveden kao
      argument pri pokretanju

 // ZAVRSETAK FIKSNOG DELA

 n = connect(sockfd, (struct sockaddr *)&serv_addr, sizeof(
     serv_addr));
 perror("Konektovanje na server");
 if (n < 0)
  exit(1);

 char *buffer_out; //skladiste za podatke koji se salju
```

```
 char buffer_in[256]; //skladiste za podatke koje cemo primiti
 int nbytes;  //broj bajtova koje smo primili u skladiste za
     prijem

 while (1) {
  printf("Unesi poruku: ");
  getline(&buffer_out, &nbytes, stdin); //prihvata korisnicki
      ulaz i vraca koliko je bajtova primljeno u nbytes
      promenljivu

  if (nbytes > 256)
   printf
       ("Poruka je ogranicena na 256 karaktera. Ostatak ce biti
           zanemaren\n");
  n = write(sockfd, buffer_out, 256); //slanje podataka iz
      skladista u koje smo upisali korisnicki unos
  perror("Slanje poruke");

  if (n < 0)
   break;

  memset(buffer_in, ' ', 256); //nulovanje celog skladista za
      prijem podataka
  n = read(sockfd, buffer_in, 256); //prihvatanje podataka u
      skladiste za prijem
  perror("Prihvatanje poruke");

  if (n < 0)
   break;

  printf("Server: %s\n", buffer_in);

 }
 close(sockfd);

 return 0;
}
```

2 Serverski deo koda

server.c

```
/*
 * Ovo je je C kod ECHO servera
 * Koristi sockete za komunikaciju,
 * errno sistem za ispisivanje
 * obavestenja i gresaka
 *
 * Ima dva bafera, pri cemu jedan
 * prihvata korisnicki unos i sluzi
 * za slanje tih podataka preko socketa
 * dok drugi ima fiksnu velicinu i
 * sluzi za prihvatanje podataka iz
 * sistemskog bafera
 *
 * Za citanje sa stdin ulaza, koristi
 * se getline funkcija, jer scanf string
 * od vise reci deli prema razmaku, pa
 * se zbog toga i komunikacija poremeti
 * i izgubi se potpuna kontrola u pogledu
 * ko je sledeci koji salje a ko prima podatke.
 *
 * Ne moramo da vodimo racuna o '\0'
 * karakteru na kraju stringa posto
 * se na njegovom kraju uvek nadje '\n'
 * zbog nacina na koji se snima unos
 * korisnicke poruke
 *
 * Kodirao: Ikac, e-mail: ikac.ikax@gmail.com
 */

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

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

  int sockfd, newsockfd, portno;
 int n;
```

```
struct sockaddr_in serv_addr;


if (argc < 2) {
 fprintf(stderr, "uputstvo: %s port\n", argv[0]);
 exit(1);
}

sockfd = socket(AF_INET, SOCK_STREAM, 0); //kreiranje socket-a
    koji ce sluziti da osluskujemo zahteve za konekcijama
perror("Creating socket");

if (sockfd < 0)
 exit(1);

memset(&serv_addr, '0', sizeof(serv_addr));

portno = atoi(argv[1]); //prihvatamo zeljeni port

serv_addr.sin_family = AF_INET; //koriste se Internet adrese
serv_addr.sin_addr.s_addr = INADDR_ANY; //slusamo i na javnoj
    adresi i lokalnoj
serv_addr.sin_port = htons(portno); //slusamo na zeljenom portu

n = bind(sockfd, (struct sockaddr *)&serv_addr, sizeof(
    serv_addr));
perror("Vezivanje za zeljeni port");

if (n < 0)
 exit(1);

n = listen(sockfd, 5);
perror("Server osluskuje moguce zajteve");

if (n < 0)
 exit(1);

// ZAVRSETAK FIKSNOG DELA

char *buffer_out; //skladiste za podatke koji se salju
char buffer_in[256]; //skladiste za podatke koje cemo primiti
int nbytes;  //broj bajtova koje smo primili u skladiste za
    prijem

while (1) {
 newsockfd = accept(sockfd, 0, 0); //prihvatanje konekcije ne
     vodeci brigu o podacima klijenta
 perror("Prihvatanje konekcije");
```

```
  if (newsockfd < 0)
   break;

  else {
   while (1) {
    memset(buffer_in, ' ', 256);
    n = read(newsockfd, buffer_in, 256);
    perror("Prihvatanje poruke");

    if (n < 0)
     break;

    printf("Klijent: %s\n", buffer_in);

    printf("Unesi poruku: ");
    getline(&buffer_out, &nbytes, stdin);

    if (nbytes > 256)
     printf
         ("Poruka je ogranicena na 256 karaktera. Ostatak ce
             biti zanemaren\n");

    n = write(newsockfd, buffer_out, 256);
    perror("Pisanje u socket");

    if (n < 0)
     break;

   }
   close(newsockfd);
      perror("Zatvaram konekciju");
  }
 }

 close(sockfd);
 return (0);
}
```

Literatura

[1] Understanding big and little endian byte order, *http://betterexplained.com/articles/understanding-big-and-little-endian-byte-order/*

[2] Beej's guide to socket programming in C, *http://beej.us/guide/bgnet/output/html/singlepage/bgnet.html*

[3] Errors: errno in UNIX programs, Chris Herborth, *http://www.ibm.com/developerworks/aix/library/au-errnovariable/*