```c
/* server.c - adapted from code for example server program that uses TCP and UDP */
/*Modified by Vincent Chu
http://www.sfu.ca/~vwchu
chuvincent (at) gmail (dot) com
*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>

#define PROTOPORT   20003       /* default protocol port number */
#define QLEN        6       /* size of request queue    */

int visits      =   0;      /* counts client connections    */
/*------------------------------------------------------------------------
 * Program:   server
 *
 * Purpose:   allocate a socket and then repeatedly execute the following:
 *      (1) wait for the next connection from a client
 *      (2) receive a packet from client
 *      (3) echo the packet back to the client
 *      (4) go back to step (2)
 *              (5) close the connection
 *
 * Syntax:    server [ port ] [ whichProtocol]
 *
 *       port  - protocol port number to use
 *       whichProtocol  - protocol to use (udp/tcp) lowercase
 *
 * Note:      The port argument is optional.  If no port is specified,
 *         the server uses the default given by PROTOPORT.  The server can
 *            handle both tcp and udp requests at the same time without
 *            restarting the server.  This is by default if the second
 *            argument is left empty.  You can also specify tcp or udp as the
 *            second argument if you just want one server not both.
 *
 *            In all cases, the argument can be '-' to indicate we want to use
 *            the default value of that argument.
 *------------------------------------------------------------------------
 */
main(argc, argv)
int argc;
char    *argv[];
{
    struct  hostent  *ptrh;  /* pointer to a host table entry   */
    struct  protoent *ptrp;  /* pointer to a protocol table entry   */
    struct  sockaddr_in sad; /* structure to hold server's address  */
    struct  sockaddr_in cad; /* structure to hold client's address  */
    int sd, sd2;     /* socket descriptors          */
```

```
    int port;        /* protocol port number        */
    int alen;        /* length of address           */
    int n;                  /* temporary variable                  */
    char    buf[1000];   /* buffer for string the server sends  */
        char * pstrzWhichProtocol; /*store whether we are using tcp or udp*/
        int bIsTCP;             /*flag storing whether we are using tcp */
    long lRecvd,lEchoed; /*for statistics*/
    int iRecvdp, iEchoedp; /*for statistics*/
    iRecvdp=0;  iEchoedp=0;
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
    sad.sin_family = AF_INET;     /* set family to Internet */
    sad.sin_addr.s_addr = INADDR_ANY; /* set the local IP address   */


    /* Check command-line argument for protocol port and extract    */
    /* port number if one is specified.  Otherwise, use the default */
    /* port value given by constant PROTOPORT           */

    if (argc > 1 && strcmp(argv[1],"-")) {/* if argument specified  */
        port = atoi(argv[1]);   /* convert argument to binary   */
    } else {
        port = PROTOPORT;   /* use default port number  */
    }
    if (port > 0)           /* test for illegal value   */
        sad.sin_port = htons((u_short)port);
    else {              /* print error message and exit */
        fprintf(stderr,"bad port number %s\n",argv[1]);
        exit(1);
    }
    printf("Listening on port %d\n",port);


    /* Check command-line argument for which protocol to use.  If
       none specify then start both tcp and udp */

        if (argc >2 ){
          pstrzWhichProtocol=argv[2];
      bIsTCP=!strcmp(pstrzWhichProtocol,"tcp");
    }

    if (argc<=2 || strcmp(argv[2],"-")==0){ /*if no option, then launch both*/
      if (fork()==0){ /*launch the child process as tcp server*/
        pstrzWhichProtocol="tcp";
        bIsTCP=1;
      }
      else { /*and itself runs as udp server*/
        pstrzWhichProtocol="udp";
        bIsTCP=0;
      }
    }
```

```
101             printf("Listening for %s packets\n",pstrzWhichProtocol);
102
103
104     /* Map protocol name to protocol number */
105
106     if ( ((int)(ptrp = getprotobyname(pstrzWhichProtocol))) == 0) {
107         fprintf(stderr, "cannot map protocol to protocol number");
108         exit(1);
109     }
110
111     /* Create a socket, depending on whether we are running as tcp or udp server.  If tcp, then create one that uses
        sock_stream, otherwise, create one that uses datagram.*/
112
113     if (bIsTCP)
114         sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
115     else
116       sd = socket(AF_INET, SOCK_DGRAM,ptrp->p_proto);
117     if (sd < 0) {
118         fprintf(stderr, "socket creation failed\n");
119         exit(1);
120     }
121
122     /* Bind a local address to the socket */
123     /*In the udp and tcp servers running at the same time case, we also bind the two servers' sockets to listen to
        the same port.  This is ok because tcp connection require both endpoints to define the connection, and it would
        not accept a udp connection - and vice versa.*/
124     if (bind(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
125         fprintf(stderr,"bind failed\n");
126         exit(1);
127     }
128
129     /* Specify size of request queue */
130         /* we only call the listen function if we're running as a tcp server*/
131     if (bIsTCP && listen(sd, QLEN) < 0) {
132         fprintf(stderr,"listen failed\n");
133         exit(1);
134     }
135
136
137
138     /* Main server loop for tcp - accept and handle requests */
139     while (bIsTCP) { /*bIsTCP will not change once defined from the beginning*/
140
141         alen = sizeof(cad);
142         if ( (sd2=accept(sd, (struct sockaddr *)&cad, &alen)) < 0) {
143             fprintf(stderr, "accept failed\n");
144             exit(1);
145         }
146         iRecvdp=0; iEchoedp=0;
147         visits++;
```

```
148         sprintf(buf,"\n\nThis server has been contacted %d time%s with tcp\n",
149             visits,visits==1?".":"s.");
150         printf("Server says: %s", buf);
151
152         n = recv(sd2, buf, sizeof(buf), 0); /*recieves from client*/
153         lRecvd=(long)n;
154         lEchoed=0l;
155
156         while (n > 0) {
157           iRecvdp++; iEchoedp++;
158           write(1,"Client says:\n", 13); /*write it out to screen*/
159           write(1,buf,n);
160           lEchoed+=(long)send(sd2,buf,n,0); /*echoed back to client*/
161           n = recv(sd2, buf, sizeof(buf), 0);/*see if there's more*/
162           lRecvd+=(long)n;
163         }
164         /*output statistics*/
165         printf("\n\nNumber of data bits received: %d\n",lRecvd);
166         printf("Number of data bits echoed: %d\n",lEchoed);
167         printf("Number of segments received: %d\n",iRecvdp);
168         printf("Number of segments echoed: %d\n",iEchoedp);
169         printf("Segments indicated may not be the true number of segments received and echoed.  This is because we   ↙
        are counting the number of calls of send(...) and recv(...), which also depends on the application buffer size,  ↙
        not just the tcp buffer size and MTU.\n");
170
171         close(sd2);
172     }
173
174     /* Main server loop to handle udp*/
175     while (!bIsTCP){  /*bIsTCP will not change once defined from the beginning*/
176       alen = sizeof(cad);
177       /*cad would contain the address info of the client after the following call*/
178
179       n = recvfrom(sd, buf, sizeof(buf), 0, (struct sockaddr*)&cad, &alen);
180
181       while (n>0){
182
183         write(1,"Client says: ", 13);
184         write(1,buf,n);
185         /*echo back to the client*/
186         sendto(sd, buf, n, 0, (struct sockaddr*)&cad, sizeof(cad));
187         printf("\n\nNumber of data bits received: %d\n",n);
188         printf("Number of data bits echoed: %d\n",n);
189         printf("Number of segments received: 1\n");
190         printf("Number of segments echoed: 1\n");
191         /*see if there's more*/
192         n = recvfrom(sd, buf, sizeof(buf), 0, (struct sockaddr*)&cad, &alen);
193       }
194
195     }
```

```
196     /*we can't close the main listening socket for udp connection until the very end.. otherwise we can't handle udp ↙
        connections anymore after one udp connection */
197     if (!bIsTCP) close(sd);
198 }
199
```