```c
/* client.c - adapted from code for example client program that uses TCP */
/*Modified by Vincent Chu, Winter 2004.
http://www.sfu.ca/~vwchu
chuvincent (at) gmail (dot) com
*/

#define closesocket close
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <errno.h>
#include <netdb.h>
#include <unistd.h>
#include <sys/select.h>

#define PROTOPORT   20003       /* default protocol port number */
#define  max(a, b)   (a>b?a:b)
#define  min(a, b)   (a<b?a:b)

extern  int     errno;
char    localhost[] =   "localhost";    /* default host name        */
/*------------------------------------------------------------------------
 * Program:   client
 *
 * Purpose:   allocate a socket, connect to a server, read from a file,
 *            send the contents of the file, wait for echo, and print all
 *            echoed data to file and to screen.
 *
 * Syntax:    client [ host [port] ] [inputFile] [outputFile] [whichProtocol] [localIP]
 *
 *       host  - name of a computer on which server is executing
 *       port  - protocol port number server is using
 *       inputFile - file to send
 *       outputFile - file to write to
 *               whichProtocol  - protocol to use (udp/tcp) lowercase
 *               localIP - Local IP Address to bind client to
 *
 * Note:      All arguments are optional.  If no host name is specified,
 *        the client uses "localhost"; if no protocol port is
 *        specified, the client uses the default given by PROTOPORT.
 *            If no input file, it reads from the file 'input', if no
 *            output file is specified, it writes to the file 'output'.
 *            If no whichProtocol is specified, it assumes TCP.
 *            If no localIP is specified, bind will not be called and the
```

```
 *            operating system will automatically choose a IP and a port.
 *
 *---------------------------------------------------------------------
 */
main(argc, argv)
int argc;
char    *argv[];
{
    struct  hostent  *ptrh, *ptrh2;  /* pointers to a host table entry,one for destination, one for local   */
    struct  protoent *ptrp;  /* pointer to a protocol table entry   */
    struct  sockaddr_in sad; /* structure to hold an IP address to connect to   */
    struct  sockaddr_in cad; /* structure to hold an IP address to bind to locally  */
    int sd;      /* socket descriptor           */
    int port;        /* protocol port number        */
    char    *host,*localAddr;   /* pointer to host name and localendpointname*/
    int n;       /* number of characters read       */
    char    buf[1000];   /* buffer for data from the server */
    FILE * pFileIn;         /*file pointers for reading and writing files*/
    FILE* pFileOut;
    long lSize,lEchoed,lSent,lDummy; /*Number of data bits echoed and sent*/
    int iEchoedp, iSentp;  /*Number of packets echoed and sent*/
    char * buffer, *lpDummy;        /*buffer for holding our input file*/
    char * pstrzInputFname; /*input file file name*/
    char * pstrzOutputFname; /*output file file name*/
    char * pstrzWhichProtocol; /*indicate whether we're using tcp or udp*/
    int bIsTCP; /*flag indicating whether we're using tcp or udp*/
        /*following are structures that we use to see if there're more data echoed from our udp server
      if we're connecting as a udp client.  More details later in the file.*/
    fd_set readFd;
    struct timeval SelectTime;
    SelectTime.tv_sec = 5;
    SelectTime.tv_usec=0;
    iEchoedp=0;
    memset((char *)&sad,0,sizeof(sad)); /* clear sockaddr structure */
    memset((char *)&cad,0,sizeof(cad)); /* clear sockaddr structure */
    sad.sin_family = AF_INET;     /* set family to Internet */
    cad.sin_family = AF_INET;     /* set family to Internet */

    /*/////////////DEALING WITH COMMAND LINE ARGUMENTS/////////////*/
    /* 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 > 2 && strcmp(argv[2],"-")) {/*if protocol port specified*/
        port = atoi(argv[2]);   /* convert to binary        */
    } else {
        port = PROTOPORT;   /* use default port number  */
        printf("Using default port %d\n",port);
    }

```

```
101     if (port > 0)           /* test for legal value     */
102         sad.sin_port = htons((u_short)port);
103     else {              /* print error message and exit */
104         fprintf(stderr,"bad port number %s\n",argv[2]);
105         exit(1);
106     }
107
108     /* Check host argument and assign host name. */
109
110     if (argc > 1 && strcmp(argv[1],"-")) {
111         host = argv[1];     /* if host argument specified   */
112     } else {
113         host = localhost;
114     }
115     printf("Will connect to host %s\n",host);
116
117     /* Check command-line argument for the input file name*/
118     if (argc >3 && strcmp(argv[3],"-")){
119       pstrzInputFname=argv[3];
120     } else {
121       pstrzInputFname="input";
122     }
123     printf("Will read from file %s\n",pstrzInputFname);
124
125     /* Check command-line argument for the output file name*/
126     if (argc >4 && strcmp(argv[4],"-")){
127       pstrzOutputFname=argv[4];
128     } else {
129       pstrzOutputFname="output";
130     }
131     printf("Will write to file %s\n",pstrzOutputFname);
132
133     /* Check command-line argument for which protocol to use.  Default to tcp*/
134     if (argc >5 && strcmp(argv[5],"-")){
135       pstrzWhichProtocol=argv[5];
136     } else {
137       pstrzWhichProtocol="tcp";
138     }
139     printf("Will use %s protocol\n",pstrzWhichProtocol);
140
141     bIsTCP=!strcmp(pstrzWhichProtocol,"tcp");
142
143     if (argc > 6 && strcmp(argv[6],"-")) {
144             localAddr = argv[6];        /* if localIP argument specified    */
145     } else {
146         localAddr = localhost;
147     }
148     printf("Will bind to endpoint %s\n",localAddr);
149     /*////////////END DEALING WITH COMMAND LINE ARGUMENTS//////////////*/
150     /*The following section of code is copied from man page to handle file*/
```

```
151     /* open the input file that contains the data we want to send*/
152     pFileIn = fopen ( pstrzInputFname , "rb" );
153     if (pFileIn==NULL) exit (1);
154     pFileOut = fopen (pstrzOutputFname,"w+");
155
156     /*get the size of the file*/
157     fseek (pFileIn , 0 , SEEK_END);
158     lSize = ftell (pFileIn);
159     rewind (pFileIn);
160
161     /*allocate memory to read the entire file*/
162     buffer = (char*) malloc (lSize);
163     if (buffer == NULL) exit (2);
164
165     /*read the contents into the buffer*/
166     fread (buffer,1,lSize,pFileIn);
167
168     /*close the file handle*/
169     fclose (pFileIn);
170     /*end copying from manpage*/
171
172     /* Convert host name to equivalent IP address and copy to sad. */
173     ptrh = gethostbyname(host);
174     if ( ((char *)ptrh) == NULL ) {
175         fprintf(stderr,"invalid host: %s\n", host);
176         exit(1);
177     }
178     memcpy(&sad.sin_addr, ptrh->h_addr, ptrh->h_length);
179
180     /* Convert localendpoint  name/address to equivalent IP address and copy to cad. */
181     ptrh2 = gethostbyname(localAddr);
182     if ( ((char *)ptrh2) == NULL ) {
183         fprintf(stderr,"invalid host: %s\n", localAddr);
184         exit(1);
185     }
186     memcpy(&cad.sin_addr.s_addr, ptrh2->h_addr, ptrh2->h_length);
187
188
189     /* Map TCP or UDP transport protocol name to protocol number. */
190     if ( ((int)(ptrp = getprotobyname(pstrzWhichProtocol))) == 0) {
191         fprintf(stderr, "cannot map protocol to protocol number");
192         exit(1);
193     }
194
195         /* Create a socket, depending on whether we are running as tcp or udp server.
196        If tcp, then create one that uses sock_stream, otherwi se, create one that uses datagram.*/
197
198     if (bIsTCP)
199       sd = socket(PF_INET, SOCK_STREAM, ptrp->p_proto);
200     else
```

```
201       sd = socket(PF_INET, SOCK_DGRAM, ptrp->p_proto);
202     if (sd < 0) {
203         fprintf(stderr, "socket creation failed\n");
204         exit(1);
205     }
206
207     /* Bind the created socket to a local endpoint.  If no argument is specified, this will be done
208        automatically by the operating system and hence the bind line will be skipped*/
209         if (argc > 6 && strcmp(argv[6],"-") && bind(sd, (struct sockaddr *)&cad, sizeof(cad)) < 0) {
210       fprintf(stderr,"bind failed\n");
211       exit(1);
212         }
213
214
215     /* Connect the socket to the specified server. For TCP, this means establishing a tcp connection..  For UDP, this↙
        means the socket would "remember" where we are sending data to each time we want to send something. */
216     if (connect(sd, (struct sockaddr *)&sad, sizeof(sad)) < 0) {
217         fprintf(stderr,"connect failed\n");
218         exit(1);
219     }
220     lSent=0l;
221
222     /* send all the data in our file buffer to the socket.  Depending on the protocol, the operating system
223        will cut up the buffer and send them as individual packets of appropriate size.  Hence, we cannot attempt
224        to measure how many packets we sent by counting how many times send(...) we called*/
225     iSentp=0;
226     /*In order to achieve this, we spawn a child process that does the sending.  The parent process will start
227       listening for replies*/
228         if (fork()==0){
229       lDummy=lSize;
230       lpDummy=buffer;
231       while (lDummy>0){
232         /*Since we already read the entire file into memory, we appropriately increment the pointer, so that
233           each time we call send, we only supply a size of at most sizeof buf*/
234         iSentp++;
235         lSent += (long)send(sd,lpDummy,min(sizeof(buf),lDummy),0);
236         lpDummy+=(long)sizeof(buf);
237         lDummy-=(long)sizeof(buf);
238         //printf("lDummy : %d",lDummy);
239       }
240       printf("\n\nNumber of data bits sent: %d\n",lSent);
241       printf("Number of packets sent: %d\n",iSentp);
242       /*The child decrements the reference counts and the process quits itself */
243       closesocket(sd);
244       free (buffer);
245       exit(0);
246     }
247     //  lSent = (long)send(sd,buffer,lSize,0);
248
249      lEchoed=0l;
```

```
250      do {
251        /*if we are using udp, then we wait for 2 seconds and see if there are more data.  If no, we quit the loop*/
252        if (!bIsTCP){
253          FD_ZERO(&readFd);
254          FD_SET(sd,&readFd);
255          select(255,&readFd,NULL,NULL,&SelectTime);
256          n=0;
257        }
258        /*if we are using tcp, then we always call recv.  Even though it would block if there's no data, it would
       return -1 if the other side has called close on socket (not necessary closing the connection yet) indicating no
       more data needs to be sent.*/
259        if (bIsTCP || FD_ISSET(sd, &readFd)==1){
260          n = recv(sd, buf, sizeof(buf), 0);
261          lEchoed+=(long)n;
262        }
263        if (n>0){
264          iEchoedp++;
265          // write(1,buf,n); /*Write receive data to screen*/
266          fwrite(buf,1,n,pFileOut);
267          if (lEchoed>=lSize) break;
268        }
269      } while ( n>0);
270
271     /* Close the socket. */
272
273     closesocket(sd);
274
275     /* Terminate the client program gracefully. */
276     fclose(pFileOut);
277     free (buffer);
278     //printf("\n\nNumber of data bits sent: %d\n",lSent);
279     printf("Number of data bits received: %d\n",lEchoed);
280     //printf("Number of packets sent: %d\n",iSentp);
281     printf("Number of packets received: %d\n",iEchoedp);
282     printf("Segments indicated may not be the true number of segments received and echoed.  This is because we are
       counting the n\
283 umber of calls of send(...) and recv(...), which also depends on the application buffer size, not just the tcp buffer
        size and MTU.\n");
284
285
286     exit(0);
287 }
288
```