

```

if ((snum = socket( AF_INET, SOCK_STREAM, 0 )) == -1)
{
    AppPanic("Socket Creation:",errno);
}

/*
** Here we clear and prepare the information to give our socket
** a real address on the system.
*/

memset( &sockaddr, 0, len );
sockaddr.sin_family = AF_INET;

/*
** Following is commented out for ease of testing purposes!
**
** {
**     struct servent *servptr;
**     char *serv = APPNAME;
**
**     if ((servptr = getservbyname( serv, "tcp" )) == NULL)
**     {
**         AppPanic("Service not in inet:db/services list!",0);
**     }
**     sockaddr.sin_port = servptr->s_port;
** }
*/

sockaddr.sin_port = 8769;

sockaddr.sin_addr.s_addr = INADDR_ANY;

/*
** Having everything set up, we now attempt to allocate the port number
** for our socket.  If this fails, we bail.
*/

if ( bind( snum, (struct sockaddr *)&sockaddr, len ) < 0 )
{
    AppPanic("Socket Binding:",errno);
}

/*
** Okay, the socket is as ready as it gets.  Now all we need to do is to
** tell the system that the socket is open for business.  In an ideal
** world, this needs to be checked for errors, but for the scope of the
** example, it isnt necessary.  By the way, the '5' in the listen() call
** indicates the "queue size" for number of outstanding requests.
*/

listen( snum, 5 );

/*
** And last, we pass the socket number back to the main routine.
*/

return snum;
}

/*
** HandleMsg() - Handles client connection and message display
**
** This is where 90% of the "function" of the program occurs.  This routine
** connects the server to the client socket, gets the incoming message pkt,
** acknowledges it, displays it, then terminates the client connection.
** For doing all that, its small, a testament to how easily the actual work
** can be done.
*/
void HandleMsg( int sock )

```

```

{
    struct NetNote in;          /* Buffer for incoming packets */

    struct sockaddr_in saddr;  /* Socket address from accept() */
    struct in_addr sad;        /* Internet address component */

    struct hostent *hent;      /* Internet host information */

    int nsock,                 /* New socket from accept() */
        len,                  /* Length of addr from accept() */
        retv;                 /* Return value from DoER call */

    char rname[80],            /* Buffer for titlebar string */
        *hname,               /* Ptr to the hostname */
        *dd_addr;             /* Ptr to the dotted-decimal address */

    /*
    ** We accept() the attempted connection on socket 'sock'
    ** which also yields the addr of the remote machine.  Then we
    ** attempt to convert the name to something meaningful.
    ** First, we clear the stuff...
    */

    bzero( (char *)&rname, 80);
    bzero( (char *)&saddr, sizeof(struct sockaddr_in) );
    bzero( (char *)&sad, sizeof(struct in_addr) );
    len = sizeof(struct sockaddr_in);

    /*
    ** Then we accept the connection on the socket
    */
    /* Bug fixed 1/93: line below checked for wrong return value. Sorry Dale! */
    if ((nsock = accept( sock, (struct sockaddr *)&saddr, &len )) < 0)
    {
        AppPanic("Accept:",errno);
    }

    /*
    ** Break the internet address out of the sockaddr_in structure and then
    ** create a dotted-decimal format string from it, for later use
    */

    sad = saddr.sin_addr;
    dd_addr = inet_ntoa(sad.s_addr);

    /*
    ** Use the internet address to find out the machine's name
    */

    if ( !(hent =
            gethostbyaddr( (char *)&sad.s_addr,
                          sizeof(struct in_addr),
                          AF_INET )))
    {
        AppPanic("Client resolution:\nAddress not in hosts db!", 0 );
    }
    hname = hent->h_name;

    /*
    ** Form the string which goes into the title bar using name & address
    */

    sprintf( rname, "FROM: %s (%s)", hname, dd_addr );

```

```

/*
** Okay, now the waiting packet needs to be removed from the connected
** socket that accept() gave back to us. Verify its of type NN_MSG and
** if not, set return type to NN_ERR. If it is, then display it and
** return an NN_ACK message.
*/

recv( nsock, (char *)&in, sizeof(struct NetNote), 0 );
if (in.nn_Code == NN_MSG)
{
    DisplayBeep(NULL);
    DisplayBeep(NULL);
    retv = DoER( rname, (char *)&in.nn_Text, (char *)&in.nn_Button );
    in.nn_Code = NN_ACK;
    in.nn_Retval = retv;
}
else
{
    in.nn_Code = NN_ERR;
}

/*
** Having dealt with the message one way or the other, send the message
** back at the remote, then disconnect from the remote and return.
*/

send( nsock, (char *)&in, sizeof(struct NetNote), 0 );
s_close( nsock );

```

```

/* sendnote.c - Execute to compile with SAS 5.10b
LC -b0 -cfistq -v -y -j73 sendnote.c
Blink FROM LIB:c.o sendnote.o TO sendnote LIBRARY LIB:LC.lib LIB:Amiga.lib
quit
*/

/*
** Our Application Prototypes (specific to notes.c file)
*/

void    main( int, char ** );
void    FinalExit( int );

/*
** Application-specific defines and globals
*/

char Version[] = "\0$VER: SendNote 1.2 (1.12.91)";

#define TEMPLATE    "Host/A,Text,Button"
#define OPT_HOST    0
#define OPT_TEXT    1
#define OPT_BUTTON  2
#define OPT_COUNT   3

/*
** The library bases...we need em later...
*/

struct Library *IntuitionBase, *SockBase;

/*
** All other includes and protos are indexed off our catch-all file
** note.h which both the client (sendnote.c) and server (shownote.c)
** include.
*/

#include    "note.h"

void    main(int argc, char **argv)
{
    struct RDArgs *rdargs;    /* ReadArgs() return information */
    struct sockaddr_in serv;  /* Server's Internet Address */
    struct hostent *host;    /* The located host info */
    struct NetNote out;      /* Message packet for send/recv */

    long opts[OPT_COUNT] = {
        0L,
        (long)"== PING! ==",
        (long)"OK"
    };

    int sock;                /* The working socket */

    char *hostnam,           /* Arg of hostname */
        *text,              /* Arg of text to be sent */
        *button;            /* Arg of button text */

    /*
    ** Process arguments using new (2.0) dos calls.
    */

    rdargs = (struct RDArgs *)ReadArgs( (UBYTE *)TEMPLATE, opts, NULL );
    if(rdargs == NULL)

```