```
/*
** Note.h - Some generic external references
*/

extern struct Library   *IntuitionBase,
                        *SockBase;


/*
** Amiga System Includes
*/

#include <exec/types.h>
#include <exec/exec.h>
#include <dos/dos.h>
#include <dos/rdargs.h>
#include <dos/dostags.h>
#include <dos/dosextens.h>
#include <intuition/intuition.h>
#include <utility/tagitem.h>

/*
** Amiga System Prototypes
*/

#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/intuition_protos.h>

/*
** socket.library Includes
*/

#include <sys/types.h>
#include <sys/socket.h>
#include <ss/socket.h>
/* make sure you rename <ss/socket_pragmas.sas|manx>
to <ss/socket_pragmas.h> */
#include <ss/socket_pragmas.h>
#include <netdb.h>

/*
** ...and some generic ANSI stuff
*/

#include <stdio.h>
#include <string.h>
#include <stddef.h>
#include <stdarg.h>
#include <stdlib.h>
#include <errno.h>

/*
** The definition of the structure which is the message packet passed
** between the client and server.  This has been kept about as minimal
** as possible, but the buffers had to be designated that way to keep
** the code of the handler routines down in size.
**
** The #define's give the valid types that may be in the nn_Code field
** of the NetNote packet.
*/

struct NetNote
{
    int     nn_Code;
    int     nn_Retval;
    char    nn_Text[200],
            nn_Button[40];
};
```

```
#define NN_MSG  0
#define NN_ACK  1
#define NN_ERR  2

/*
** This definition is used in both the cl
** entries in the INET:DB/SERVICES file w
*/

#define APPNAME "notes"

#ifdef LATTICE
int CXBRK(void) { return(0); }  /* Disab
int chkabort(void) { return(0); }
#endif

/*
** End of note.h
*/
```

```
sed in both the client and server as the name any
 DB/SERVICES file will be under.


n(0); }  /* Disable Lattice CTRL/C handling */
eturn(0); }
```



```
;/* shownote.c - Execute to compile with SAS 5.10b
LC -b0 -cfistq -v -y -j73 shownote.c
Blink FROM LIB:c.o shownote.o TO shownote LIBRARY LIB:LC.l
quit
*/


/*
** Our Application Prototypes (specific to noter.c file)
*/
void    main( void );
void    TG_Init( void );
int     SS_Init( void );
int     DoER( char *, char *, char * );
void    AppPanic( char *, int );
void    HandleMsg( int );

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

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

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

struct Library *IntuitionBase, *SockBase;

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

#include    "note.h"


VOID    main( VOID )
{
    int     socket;         /* The socket */

    fd_set  sockmask,       /* Mask of open sockets */
            mask;           /* Return value of socketwait(

    long    umask;          /* AmigaDOS signal mask */



    /*
    ** Call TG_Init to prepare the generic Amiga stuff for
    */

    TG_Init();

    /*
    ** ...and SS_Init for the socket-specific arrangements,
    ** track of what it hands back.
    */

    socket = SS_Init();

    /*
    ** First, prepare the various masks for signal process
    */

    FD_ZERO( &sockmask );
    FD_SET( socket, &sockmask );
```

```
 SAS 5.10b

e LIBRARY LIB:LC.lib LIB:Amiga.lib



o noter.c file)



1.12.91)";



exed off our catch-all file
e.c) and server (shownote.c) include.



pen sockets */
lue of socketwait() */

signal mask */



ic Amiga stuff for use...



cific arrangements, keeping



for signal processing
```

```
    /*
    ** And we enter the event loop itself
    */

    while(1)
    {
        /*
        ** Reset the mask values for another pass
        */

        mask = sockmask;
        umask = SIGBREAKF_CTRL_C;

        /*
        ** selectwait is a combo network and Amiga Wait() rolled into
        ** a single call.  It allows the app to respond to both Amiga
        ** signals (CTRL-C in this case) and to network events.
        **
        ** Here, if the selectwait event is the SIGBREAK signal, we
        ** bail and AppPanic() but otherwise its a network event.
        ** This is a very crude way of handling the exit, but it
        ** is an effective one
        */

        if (selectwait( 2, &mask, NULL, NULL, NULL, &umask ) == -1 )
        {
            AppPanic("CTRL-C:\nProgram terminating!",0);
        }

        /*
        ** Since the contact between the client and server is so
        ** quick, an iterative server is adeqaute.  For cases where
        ** extended connections or concurrent connections are needed,
        ** either a state-machine or concurrent server would be a
        ** better choice.
        **/

        /*
        ** Here, we accept the pending connection (the only case
        ** possible with this mechanism) and dispatch to a routine
        ** which actually handles the client-server communication.
        */

        if (FD_ISSET( socket, &mask ))
        {
            HandleMsg( socket );
        }
        else
        {
            AppPanic("Network Signal Error!",0);
        }
    }
}


/*
**  AppPanic() - General Shutdown Routine
**
**  This routine serves to provide both a nice GUI way of indicating error
**  conditions to the user, and to handle all the aspects of shutting the
**  server down.  In a real-world application, you would probably separate
**  the error condition shutdown from the normal shutdown.  Since this is
**  an example, it would add needless complexity to the code.  Further,
**  as far as this server is concerned, shutting down _is_ an error state,
**  since SIGBREAKF_CTRL_C is a process-specific warning of shutdown.
*/
```

```
VOID    AppPanic( char *panictxt, int panicnum )
{
    char buffer[255];

    if (!panicnum)
    {
        DoER( APPNAME, panictxt, "OK" );
    }
    else
    {
        sprintf( (char *)&buffer, "%s\n\n%s", panictxt, strerror(panicnum));
        DoER( APPNAME, (char *)&buffer, "OK" );
    }
    if (SockBase)
    {
        cleanup_sockets();
        CloseLibrary(SockBase);
    }

    if (IntuitionBase)
    {
        CloseLibrary(IntuitionBase);
    }

    exit(RETURN_ERROR);

}

/*
**  DoER() - Attempt at a "generic" wrapper for EasyRequest()
**
**  Since EasyRequest(), though "easy", still requires some initialization
**  before it can be used, this routine acts as a wrapper to EasyRequest.
**  It also catches and provides a means to implement application-generic
**  values for what gets handed to the EasyRequest routine, making coding
**  just a wee bit easier.
*/
int     DoER( char *titletxt, char *msgtxt, char *btntxt )
{
    struct EasyStruct easy =    {
                                sizeof(struct EasyStruct),
                                NULL,
                                NULL,
                                NULL,
                                NULL
                                };

    int retval = 0;

    if (IntuitionBase)
    {
        if (titletxt)
        {
            easy.es_Title = titletxt;
        }
        else
        {
            easy.es_Title = APPNAME;
        }

        if (msgtxt)
        {
            easy.es_TextFormat = msgtxt;
        }
        else
        {
            easy.es_TextFormat = "Null message text!\nIsnt it?";
        }
```