```
/* Multiplexing Daemon
 *
 * -Thanx to Ballbach, Iretd, Muffin.
 *
 *   This is a server that allows multiple connections at once.   It is not really "robust" but
 *   more of a skeleton, so you can add on however you like.   What happens to the data
 *   is up to you.   It's probably easiest if you have the functions that mess with the data in
 *   another file, that is what I did.   Compiles fine under linux.
 *                                                                              -Seti
 */

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

/*   Some stuff to determine what to do with the data we get
          #include "login_functions.c"
          #include "decode.c"
*/

#define PORT 4019
#define MAX_CONNECTIONS 15

char welcomeMessage[] = "Multiplexing Daemon\n";
void setup(fd_set *, int *, int *, int);
int free_slot(int *, int *);

main()

int sockfd, fds[MAX_CONNECTIONS], bytes, retval, freei, x=0, connections=0;
int len = sizeof(struct sockaddr_in);
struct sockaddr_in my_addr, their_addr;
fd_set in;
struct timeval timeout;
char buf[50];

         timeout.tv_sec = 5;
         timeout.tv_usec = 0;

         my_addr.sin_family = AF_INET;
         my_addr.sin_port = htons(PORT);
         my_addr.sin_addr.s_addr = INADDR_ANY;
         bzero(&(my_addr.sin_zero), 8);

         if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
```

```
            exit(1);

        if(bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr))==-1)
            exit(1);

        if(listen(sockfd, MAX_CONNECTIONS) == -1)
            exit(1);

while(1)

        setup(&in, &sockfd, fds, connections);
        retval = select(MAX_CONNECTIONS, &in, NULL, NULL, &timeout);

 if(retval)


        if(FD_ISSET(sockfd, &in))

                printf("Accepting new connection...\n");
                freei = free_slot(&fds, &connections);

            if(!(freei == -1) && !(freei == MAX_CONNECTIONS))

                printf("Creating socket fds[%d]: FD #:%d\n", freei, fds[freei]);
                fds[freei] = accept(sockfd, (struct sockaddr *)&their_addr, &len);
                FD_SET(fds[freei], &in);
                send(fds[freei], welcomeMessage, strlen(welcomeMessage), 0);




/* Check for incoming data and exceptions */

setup(&in, &sockfd, fds, connections);
retval = select(MAX_CONNECTIONS, &in, NULL, NULL, &timeout);

 if(retval)

        for(x=0; x<connections; x++)

                if(FD_ISSET(fds[x], &in))

                        printf("Data Available\n");
                        if((bytes = read(fds[x], buf, sizeof(buf))) <=0)

                                close(fds[x]);
                                FD_CLR(fds[x], &in);
                                fds[x] = 0;


                       else

                                buf[bytes] = '\0';
```

```
                                        printf("Data Received on fd[%d]: %s\n", x, buf);
                                         //this is in decode.c
                                        //decode(buf, fds[x]);






/*** Initialize & Setup file descriptors
 */
void setup(fd_set *in, int *mainsock, int *fds, int i)

int x;

          FD_ZERO(in);
          FD_SET((*mainsock), in);

          for(x=0; x<i; x++)
                    FD_SET(fds[x], in);



/*** Return a free connection slot, if one is available
 */
int free_slot(int *fds, int *connections)

int x;
int tmp[MAX_CONNECTIONS];

          for(x=0; x<(*connections); x++)

                    if(fds[x] == 0)

                              fds[x] = tmp[x];   //fix it later
                              return x;



 if((*connections) < MAX_CONNECTIONS)
          return (*connections)++;

 if((*connections) == MAX_CONNECTIONS)
          return (*connections);

return -1;
```