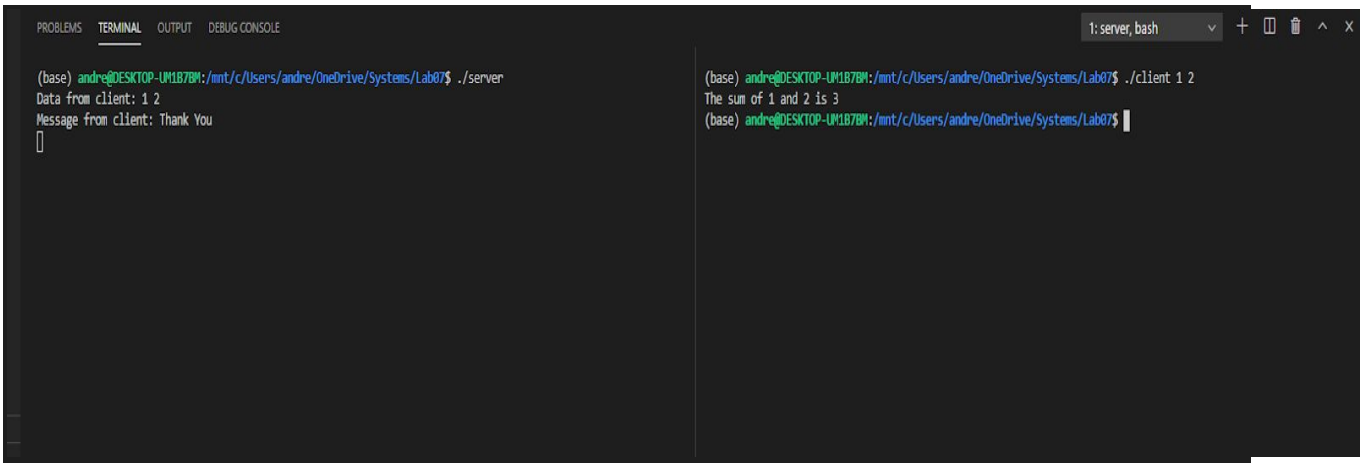


```c
/*********************************************
 *                  server.c
 *
 *  A more robust server that receives two
 *  integers from a client and returns their
 *  sum.  In return it gets a nice message
 *  from the client.
 *********************************************/

#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include <stdlib.h>
#include "lib.h"

int main(int argc, char **argv) {
    int sock, conn;
    int i;
    int rc;
    struct sockaddr address;
    socklen_t addrLength = sizeof(address);
    struct addrinfo hints;
    struct addrinfo *addr;
    int len;
    int arg1, arg2;
```

```
    int ret;
    char *message;

    struct data data;
    arg1 = data.arg1;
    arg2 = data.arg2;
    ret = data.result;

    /*
     *  set the hints structure to zero
     */
    memset(&hints, 0, sizeof(hints));

    /*
     *  want a stream, also address that will accept all
     *  connections on this host
     */
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE | AI_ADDRCONFIG;
    if((rc = getaddrinfo(NULL, "4321", &hints, &addr))) {
        printf("host name lookup failed: %s\n", gai_strerror(rc));
        exit(1);
    }

    /*
     *  use the first entry returned by getaddrinfo
     */
    sock = socket(addr->ai_family, addr->ai_socktype, addr->ai_protocol);
    if(sock < 0) {
        printf("Can't create socket\n");
        exit(1);
    }

    /*
     *  want to be able to reuse the address right after
     *  the socket is closed.  Otherwise must wait for 2 minutes
     */
    i = 1;
    setsockopt(sock, SOL_SOCKET, SO_REUSEADDR, &i, sizeof(i));
```

```
    rc = bind(sock, addr->ai_addr, addr->ai_addrlen);
    if(rc < 0) {
        printf("Can't bind socket\n");
        exit(1);
    }

    /*
     *  free results returned by getaddrinfo
     */
    freeaddrinfo(addr);

    rc = listen(sock, 5);
    if(rc < 0) {
        printf("Listen failed\n");
        exit(1);
    }

    /*
     *  accept an arbitrary number of connections in a loop
     */
    while((conn = accept(sock, (struct sockaddr*) &address, &addrLength))
        >= 0) {

        ret = receiveData(conn, &arg1);
        ret = receiveData(conn, &arg2);
        printf("Data from client: %d %d\n", arg1, arg2);

        arg1 = arg1 + arg2;
        sendData(conn, arg1, arg2);

        message = readString(conn);
        if(message != NULL) {
            printf("Message from client: %s\n", message);
            free(message);
        } else {
            printf("Error receiving message from client\n");
        }
        close(conn);
    }
```

```
    close(sock);
    exit(0);
}
```

```
  /*********************************************
0000664 0001750 0001750 00000004075 13627574234 010463  0
ustar   mark                            mark
/*****************************************
 *
 *               lib.c
 *
 *  Library of procedures that are useful
 *  for reliable network programming
 *****************************************/



#include <unistd.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <stdio.h>
#include <arpa/inet.h>
#include "lib.h"




/*
 *  Read count bytes from fd if at all possible.
 *  The only potential problem is an end of file
 *  which suggests a problem with the server.
 */
int readn(int fd, char *buffer, int count) {
    char *ptr;
    int n;
    int left;

    ptr = buffer;
    left = count;
```

```
    while(left > 0) {
        n = read(fd, ptr, left);
        if(n == 0)
            break;
        if(n < 0) {
            if(errno == EINTR || errno == EAGAIN || errno == EWOULDBLOCK)
{
                n = 0;
            } else {
                return(-1);
            }
        }
        left -= n;
        ptr += n;
    }
    return(count-left);
}


/*
 *  Write count bytes on fd if at all possible.
 *  The only potential problem is an error that
 *  can't be recovered from.
 */
int writen(int fd, char *buffer, int count) {
    char *ptr;
    int n;
    int left;

    ptr = buffer;
    left = count;
    while(left > 0) {
        n = write(fd, ptr, left);
        if(n < 0) {
            if(errno == EINTR || errno == EAGAIN || errno == EWOULDBLOCK)
{
                n = 0;
            } else {
                return(-1);
            }
```

```
        }
        left -= n;
        ptr += n;
    }
    return(count-left);
}

 /*
  *  Reliably read a text string from fd.
  *  Returns NULL if it can't read the string
  *  otherwise a pointer to the string.
  */
char *readString(int fd) {
    short len;
    char *buffer;
    int ret;

    ret = readn(fd, (char*)&len, sizeof(len));
    if(ret <= 0)
        return(NULL);
    len = ntohs(len);
    buffer = (char*) malloc(len);
    ret = readn(fd, buffer, len);
    if(ret != len)
        return(NULL);
    else
        return(buffer);
}


/*
 *  Reliably write the string on fd.
 *  Returns -1 on failure, 0 on success.
 */
int writeString(int fd, char *string) {
    short len;
    int ret;
    short buffer;

    len = strlen(string)+1;
```

```
    buffer = htons(len);
    ret = writen(fd, (char*)&buffer, sizeof(len));
    if(ret != sizeof(len))
        return(-1);
    ret = writen(fd, string, len);
    if(ret != len)
        return(-1);
    else
        return(0);
}


int sendData(int sock, int arg1, int arg2) {
    int ret;

    ret = writen(sock, (char *) &arg1, sizeof(arg1));

    if (ret != sizeof(arg1)) {
        printf("Error sending argument one\n");
        close(sock);
        return(-1);
    }


    ret  = writen(sock, (char *) &arg2, sizeof(arg2));

    if (ret != sizeof(arg2)) {
        printf("Error sending second argument");
        close(sock);

        return(-1);
    }
    return(0);
}



int receiveData(int sock, int *result) {
    int ret;
```

```
    ret = readn(sock, (char *) result, sizeof(*result));

    if (ret != sizeof(*result)) {
        printf("Error reading result. \n");
        close(sock);
        return(-1);
    }
    return(0);
}
```

```
 /*****************************************************
 *                          client.c
 *
 *  A more robust client example.  In this example the
 *  client has two arguments, integers.  These integers
 *  are sent to the server, which returns the sum of the
 *  two integers.
 *****************************************************/

#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <unistd.h>
#include "lib.h"



int main(int argc, char **argv) {
    struct addrinfo hints;
    struct addrinfo *addr;
    struct sockaddr_in *addrinfo;
    int rc;
    int sock;
    char buffer[512];
    int len;
    int arg1, arg2;
```

```
    int result;
    int ret;



    // use struct
    struct data data;
    arg1 = data.arg1;
    arg2 = data.arg2;
    result = data.result;

    if(argc != 3) {
        printf("Usage: client num1 num2\n");
        exit(1);
    }

    arg1 = atoi(argv[1]);
    arg2 = atoi(argv[2]);

    /*
     *  clear the hints structure to zero
     */
    memset(&hints, 0, sizeof(hints));

    /*
     *  want a stream on a compatible interface
         */
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_ADDRCONFIG;
    /*
     *  localhost is the name of the current computer
     */
    rc = getaddrinfo("localhost", NULL, &hints, &addr);
    if(rc != 0) {
        printf("Host name lookup failed: %s\n", gai_strerror(rc));
        exit(1);
    }

    /*
     *  use the first result from getaddrinfo
```

```
     */
    addrinfo = (struct sockaddr_in *) addr->ai_addr;

    sock = socket(addrinfo->sin_family, addr->ai_socktype,
addr->ai_protocol);
    if(sock < 0) {
        printf("Can't create socket\n");
        exit(1);
    }

    /*
     *  specify the port number
     */
    addrinfo->sin_port = htons(4321);

    rc = connect(sock, (struct sockaddr *) addrinfo, addr->ai_addrlen);
    if(rc != 0) {
        printf("Can't connect to server\n");
        exit(1);
    }

    /*
     *  free the results returned by get addrinfo
     */
    freeaddrinfo(addr);
    sendData(sock, arg1, arg2);
    receiveData(sock, &result);


    printf("The sum of %d and %d is %d\n", arg1, arg2, result);
    ret = writeString(sock, "Thank You");
    if(ret) {
        printf("Error sending thank you\n");
        close(sock);
        exit(1);
    }

    close(sock);

    exit(0);
```

```
}
```