```c
#include <arpa/inet.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
//header located at /usr/include


/*
 Functions Used:
int socket(int domain, int type, int protocol);

int setsockopt(int socket, int level, int option_name,
       const void *option_value, socklen_t option_len);

 int bind(int socket, const struct sockaddr *address,
       socklen_t address_len);

int listen(int socket, int backlog);


int accept(int socket, struct sockaddr *restrict address,
       socklen_t *restrict address_len);

ssize_t send(int socket, const void *buffer, size_t length, int flags);


ssize_t recv(int socket, void *buffer, size_t length, int flags);

Structures Used:
/* Structure describing an Internet socket address.  */
/*
struct sockaddr_in
  {
    __SOCKADDR_COMMON (sin_);
    in_port_t sin_port;         // Port number.
    struct in_addr sin_addr;        // Internet address.

    // Pad to size of `struct sockaddr'.
    unsigned char sin_zero[sizeof (struct sockaddr) -
               __SOCKADDR_COMMON_SIZE -
               sizeof (in_port_t) -
               sizeof (struct in_addr)];
  };

 */
void memdump(const unsigned char *data, const unsigned int length);

int main(void) {
    int sockfd, new_sockfd, sock_options, bind_sock, listen_sock;
    struct sockaddr_in server_addr, client_addr; // My address information
    socklen_t sin_size;
    int recv_length=1, option_value=1;
    char buffer[1024];


    sockfd = socket(PF_INET, SOCK_STREAM, 0);
    if (sockfd == -1)
    {
        printf("Couldnt create a socket");
    }


    sock_options = setsockopt(sockfd, SOL_SOCKET, SO_REUSEADDR, &option_value, sizeof(int));
    if (sock_options == -1)
    {
        printf("Couldnt use setsockopt function");
    }


    server_addr.sin_family = AF_INET; // Host byte order
    server_addr.sin_port = htons(4444); // Short, network byte order
    server_addr.sin_addr.s_addr = 0; // Automatically fill with my IP.
    for(int i = 0; i < 8 ;i++)
    {
    server_addr.sin_zero[i] = 0;
    }


    bind_sock = bind(sockfd, (struct sockaddr *)&server_addr, sizeof(struct sockaddr));
    if(bind_sock == -1)
    {
        printf("Failed binding to socket");
    }

    listen_sock = listen(sockfd, 8);
    if (listen_sock == -1)
    {
        printf("Failed listening to socket");
    }


while(1) { // Accept loop.
    sin_size = sizeof(struct sockaddr_in);
    new_sockfd = accept(sockfd, (struct sockaddr *)&client_addr, &sin_size);
    if(new_sockfd == -1)
    {
        printf("failed accepting connection");
    }

    printf("server: got connection from %s port %d\n", inet_ntoa(client_addr.sin_addr), ntohs(
    client_addr.sin_port));

    send(new_sockfd, "*Hell0 world!*\n", 15, 0);

    recv_length = recv(new_sockfd, &buffer, 1024, 0);
    while(recv_length > 0)
    {
        printf("RECV: %d bytes\n", recv_length);
        memdump(buffer, recv_length);
        recv_length = recv(new_sockfd, &buffer, 1024, 0);
    }

}
}

void memdump(const unsigned char *data, const unsigned int length)
{

unsigned char byte;
unsigned int i, j;
for(i=0; i < length; i++) {
    byte = data[i];
    printf("%02x ", data[i]); // Display byte in hex.
if(((i%16)==15) || (i==length-1)) {
    for(j=0; j < 15-(i%16); j++)
    printf("   ");
    printf("|| ");
for(j=(i-(i%16)); j <= i; j++) { // Display printable bytes from line.
byte = data[j];
if((byte > 31) && (byte < 127)) // Outside printable char range
printf("%c", byte);
else
printf(".");
}
printf("\n"); // End of the dump line (each line is 16 bytes)
} // End if
}
}
```