

Operating Systems

PRAC 1: SOCKETS

Alex Gutiérrez, Joel Amor

Curs: 2022/2023

Pere Tuset

Data d'entrega: 30/04/2023

Table of Contents

| | |
|-------------------------|-----------|
| Activity 1 | 3 |
| Activity 2 | 8 |
| Activity 3 | 10 |

Activity 1

- Client (cli1.c)

Client is responsible for guessing the random value the server generates.

How to execute:

`./cli1 [port] [ip]`, where port and IP are optional values, if not passed, default values are taken.

Execution example:

```
devasc@labvm:/media/sf_S0/PRAC1/Activity 1$ ./cli1
Connected to server
Guessing 50
Guess is too high
Guessing 25
Guess is too high
Guessing 12
Guess is too high
Guessing 6
Guess is too high
Guessing 3
Guess is too high
Guessing 1
Correct guess: 1
Number of iterations: 5
```

Code explanation:

Prints an error message and exits the program with error status.

```
void err_sys(const char *error_msg)
{
    perror(error_msg);
    exit(1);
}
```

Creates the socket and uses it to establish a connection with the server, then returns the socket. It also implements error handling.

```
int connectToServer(int port, const char *ip_addr)
{
    struct sockaddr_in server_address;
    memset(&server_address, 0, sizeof(server_address));
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr = inet_addr(ip_addr);
    server_address.sin_port = htons(port);
```

```

int sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (sock < 0)
{
    err_sys("Error creating socket");
}

int result = connect(sock, (struct sockaddr *)&server_address,
sizeof(server_address));
if (result < 0)
{
    err_sys("Error connecting to server");
}

printf("Connected to server\n");

return sock;
}

```

Reads the value sent by the server that will be used to know if it's the correct guess.

```

int readFromSocket(int sock, char *buffer)
{
    int n = read(sock, buffer, BUFFSIZE);
    if (n < 0)
    {
        err_sys("Error reading from socket");
    }
    return atoi(buffer);
}

```

Used to write the client guess to the socket, as an integer value.

```

void writeToSocket(int sock, int value, char *buffer)
{
    sprintf(buffer, "%d", value);
    int n = write(sock, buffer, BUFFSIZE);
    if (n < 0)
    {
        err_sys("Error writing to socket");
    }
}

```

Implements the game logic. The client's first guess is always 50. The guess is then written in the socket and the server returns the value used to know if the guess is higher or lower than the correct number.

```

void play(int sock)
{

```

```

int guess, min = 0, max = 100, iterations = 0;
char buffer[BUFFSIZE];

while (1)
{
    guess = (max + min) / 2;
    printf("Guessing %d\n", guess);

    writeToSocket(sock, guess, buffer);

    int result = readFromSocket(sock, buffer);

    if (result == 0)
    {
        printf("Correct guess: %d\n", guess);
        break;
    }
    else if (result > 0)
    {
        printf("Guess is too high\n");
        max = guess;
    }
    else if (result < 0)
    {
        printf("Guess is too low\n");
        min = guess;
    }
    else
    {
        err_sys("Error parsing response from server");
    }
    iterations++;
}
printf("Number of iterations: %d\n", iterations);
}

```

Sets the port and ip address to arguments, if passed. Creates a socket and connects it to the server using *connectToServer* function. Calls plays and handles socket closing when the game is finished.

```

int main(int argc, char *argv[])
{
    int port, sock;
    const char *ip_addr;

    if (argc == 1)
    {
        port = DEFAULT_PORT;
        ip_addr = DEFAULT_IP;
    }
    else if (argc == 2)
    {
        port = atoi(argv[1]);
        ip_addr = DEFAULT_IP;
    }
}

```

```

    }
    else if (argc == 3)
    {
        port = atoi(argv[1]);
        ip_addr = argv[2];
    }
    else
    {
        err_sys("Usage: cli1 [port] [ip]");
    }

    sock = connectToServer(port, ip_addr);

    play(sock);

    close(sock);
    exit(0);
}

```

- Server (ser1.c)

The server generates the random numbers that the client must guess and gives hints about the range of the number.

Server implements functions **err_sys**, **readFromSocket**, **writeToSocket** and **createSocket** the same way as the client does.

How to execute:

./ser1 [port], where port is an optional value, if not passed, default port is taken.

Execution example:

```

devasc@labvm:/media/sf_SO/PRAC1/Activity 1$ ./ser1
Client: 127.0.0.1
The random number is: 1

```

Code explanation:

Compares the correct number and the value received from the client and returns a hint to the client based on it.

```

int getReturnValue(int received, int random_number)
{
    if (received == random_number)
        return 0; // Value matches the random number
    else if (received > random_number)
        return 1; // Value is greater than the random number
    else

```

```

    return -1; // Value is smaller than the random number
}

```

Implements game logic. Generates a random number and handles the clients until it matches the guessed number.

```

void handleClient(int sock)
{
    char buffer[BUFFSIZE];
    int received, random_number, toreturn;

    // Seeds the random number generator
    srand(time(NULL));

    // Generates a random number between 0 and 100
    random_number = rand() % 101;
    printf("The random number is: %d\n", random_number);

    while (1)
    {
        received = readFromSocket(sock, buffer);
        toreturn = getReturnValue(received, random_number);
        writeToSocket(sock, toreturn, buffer);

        // Value matches guessed number.
        if (toreturn == 0)
            break;
    }
    close(sock);
}

```

Calls **handleClient** for every client connection.

```

void wait(int server_sock)
{
    struct sockaddr_in echoclient;
    while (1)
    {
        unsigned int clientlen = sizeof(echoclient);
        int client_sock = accept(server_sock, (struct sockaddr *)&echoclient,
        &clientlen);
        if (client_sock < 0)
            err_sys("Error accepting client");

        printf("Client: %s\n", inet_ntoa(echoclient.sin_addr));

        handleClient(client_sock);
    }
}

```

Sets the port to the argument, if passed. If not, it takes the default port. Calls create sockets and wait to handle game logic and client connections.

```

int main(int argc, char *argv[])
{
    int port = DEFAULT_PORT;
    if (argc > 1)
    {
        port = atoi(argv[1]);
    }
    else if(argc > 2)
    {
        err_sys("Usage: ser1 [port]");
    }
    int server_sock = createSocket(port);
    wait(server_sock);

    exit(0);
}

```

Activity 2

- Client (cli2.c)

How to execute:

./cli2 [port] [ip], where port and ip are optional values, if not passed, default values are taken.

Execution example:

```

devasc@labvm:/media/sf_50/PRAC1/Activity 2$ ./cli2
Connected to server
Guessing 50
Guess is too low
Guessing 75
Guess is too low
Guessing 87
Guess is too low
Guessing 93
Guess is too low
Guessing 96
Guess is too high
Guessing 94
Correct guess: 94
Number of iterations: 5

```

Code explanation:

Same functionalities as cli1.c.

- Server (cli2.c)

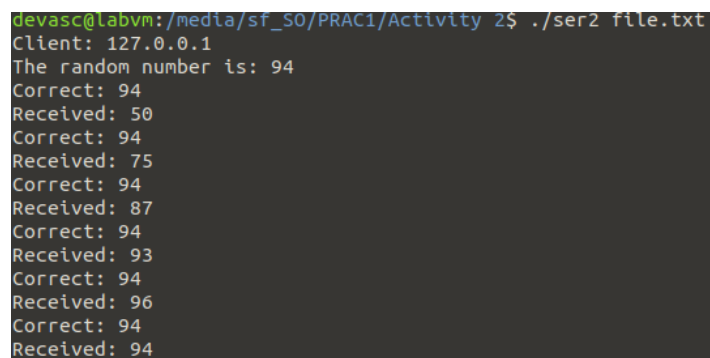
The server generates the random numbers that the client must guess and gives hints about the range of the number. The random number is now generated using the lines of a text file the user passes as an argument.

Server implements functionality just as ser1.c, excepting **readFile** function.

How to execute:

./ser2 [file_name] [port], where port is optional, if not passed, default port is taken.

Execution example:



```
devasc@labvm:/media/sf_50/PRAC1/Activity 2$ ./ser2 file.txt
Client: 127.0.0.1
The random number is: 94
Correct: 94
Received: 50
Correct: 94
Received: 75
Correct: 94
Received: 87
Correct: 94
Received: 93
Correct: 94
Received: 96
Correct: 94
Received: 94
```

Code explanation:

Reads a text file line by line, saving the line length in the lines array. Lines and line_count are global variables.

```
void readFile(char *file_name)
{
    char line[MAX_CHARS_PER_LINE];

    /* Open the file */
    FILE *file = fopen(file_name, "r");
    if (file == NULL)
    {
        err_sys("Error while opening file");
    }

    /* Read the file line by line */
    while (fgets(line, MAX_CHARS_PER_LINE, file) != NULL)
    {
        /* Saves the length of the line in the array*/
        lines[line_count] = strlen(line) % 100;
        line_count++;
    }
    fclose(file);
}
```

The random number to be guessed, is now generated taking a random line length.

```
int generateRandom()
{
    srand(time(NULL));
    return lines[rand() % line_count];
}
```

Activity 3

- Client (cli3.c)

How to execute:

./cli3 [port] [ip], where port and ip are optional values, if not passed, default values are taken.

Execution example:

```
devasc@labvm:/media/sf_50/PRAC1/Activity 3$ ./cli3
Connected to server
Guessing 50
Guess is too high
Guessing 25
Guess is too high
Guessing 12
Guess is too low
Guessing 18
Guess is too low
Guessing 21
Guess is too low
Guessing 23
Guess is too high
Guessing 22
Correct guess: 22
Number of iterations: 6
```

Code explanation:

Same functionalities as cli2.c.

- Server (ser3.c)

How to execute:

./ser3 [udp_port] [ip] [tcp_port]

Execution example:

```

devasc@labvm:/media/sf_50/PRAC1/Activity 3$ ./ser3 9999 127.0.0.1 8888
Number: 22
Client: 127.0.0.1
The random number is: 22
Correct: 22
Received: 50
Correct: 22
Received: 25
Correct: 22
Received: 12
Correct: 22
Received: 18
Correct: 22
Received: 21
Correct: 22
Received: 23
Correct: 22
Received: 22

```

Code explanation:

The TCP connection is handled the same way as ser2. Ser3 sends two requests to file3 UDP server to receive the information used to generate a random guess number.

```

/* Create UDP socket */
udp_socket = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (udp_socket < 0)
{
    err_sys("Error socket");
}

/* Configure/set socket address for the server (UDP) */
memset(&echoserver, 0, sizeof(echoserver)); /* Erase the memory area */
echoserver.sin_family = AF_INET; /* Internet/IP */
echoserver.sin_addr.s_addr = inet_addr(server_ip); /* IP address */
echoserver.sin_port = htons(udp_port); /* Server port */

/* Try to send the word to server */
result = sendto(udp_socket, "-", 5, 0, (struct sockaddr *)&echoserver,
sizeof(echoserver));
if (result < 0)
{
    err_sys("Error writing on socket");
}

/* Set the maximum size of address to be received */
clientlen = sizeof(echoclient);

/* Wait for echo from server */
received = recvfrom(udp_socket, buffer, BUFFER_SIZE, 0, (struct sockaddr
*)&echoclient, &clientlen);
if (received < 0)
{
    err_sys("Error reading");
}

/* Number of Lines*/
int lines = atoi(buffer);

```

```

srand(time(NULL));
sprintf(buffer, "%d", rand() % lines);

result = sendto(udp_socket, buffer, 5, 0, (struct sockaddr *)&echoserver,
sizeof(echoserver));
if (result < 0)
{
    err_sys("Error writing on socket");
}

/* Wait for echo from server */
received = recvfrom(udp_socket, buffer, BUFFER_SIZE, 0, (struct sockaddr
*)&echoclient, &clientlen);
if (received < 0)
{
    err_sys("Error reading");
}

```

- file (file3.c)

Implements an UDP server that reads a file, obtaining the total number of lines and stores the length of each line in an array. When a connection with server3 is established, it sends the data in two communications.

How to execute:

./file 3 [file_name] [port], where port is an optional value.

Execution example:

```

devasc@labvm:/media/sf_SO/PRAC1/Activity 3$ ./file3 file.txt
Client: 127.0.0.1, Message: -
Client: 127.0.0.1, Message: 2
Number: 73
Client: 127.0.0.1, Message: -
Client: 127.0.0.1, Message: 1
Number: 22

```

Code explanation:

The file is read the same way as ser2 **readFile** function.

Creates a UDP socket and binds it.

```

/* Create UDP socket */
sock = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP);
if (sock < 0)
{
    err_sys("Error socket");
}

```

```

/* Prepare sockaddr_in structure for server address */
memset(&echoserver, 0, sizeof(echoserver)); /* Erase the memory area */
echoserver.sin_family = AF_INET; /* Internet/IP */
echoserver.sin_addr.s_addr = htonl(INADDR_ANY); /* Receive requests from any IP
address valid on server */
echoserver.sin_port = htons(port); /* Server port */

/* Get size of echoserver structure */
serverlen = sizeof(echoserver);

/* Bind that socket with the OS, to be able to receive messages on that socket */
result = bind(sock, (struct sockaddr *)&echoserver, serverlen);
if (result < 0)
{
    err_sys("Error bind");
}

```

When a request is received, sends lines information in two communications.

```

/* Receives first request */
received = recvfrom(sock, buffer, BUFFER_SIZE, 0, (struct sockaddr
*)&echoclient, &clientlen);
if (received < 0)
{
    err_sys("Error receiveing word from client");
}

/* Print client address */
fprintf(stderr, "Client: %s, Message: %s\n", inet_ntoa(echoclient.sin_addr),
buffer);

/* Try to send number of lines*/
sprintf(buffer, "%d", line_count);
result = sendto(sock, buffer, received, 0, (struct sockaddr *)&echoclient,
sizeof(echoclient));
if (result != received)
{
    err_sys("Error writing message back to the client");
}

/* Receives second request */
received = recvfrom(sock, buffer, BUFFER_SIZE, 0, (struct sockaddr
*)&echoclient, &clientlen);
if (received < 0)
{
    err_sys("Error receiveing word from client");
}

/* Print client address */
fprintf(stderr, "Client: %s, Message: %s\n", inet_ntoa(echoclient.sin_addr),
buffer);

```

```
printf("Number: %d\n", lines[atoi(buffer)]);

/* Try to send number of lines*/
sprintf(buffer, "%d", lines[atoi(buffer)]);
result = sendto(sock, buffer, received, 0, (struct sockaddr *)&echoclient,
sizeof(echoclient));
if (result != received)
{
    err_sys("Error writing message back to the client");
}
```