Operating Systems – Project-2

University at Albany
Department of Computer Science
ICSI 500

Project-2
Assigned: Thursday, October 12th, 2023
Due: Thursday, November 9th, by 11:59 PM. No submissions will be accepted after the due date.
<u>Unlimited number of submissions is allowed.</u>

PURPOSE

Develop a practical understanding of task collaboration using socket programming, and error handling.

OBJECTIVES

Develop a client/server application using Linux TCP sockets and the C programming language. Your solution will respond to service requests by clients. Transmission errors will be handled by applying the CRC algorithm to detect errors and Hamming to both detect and correct single bit errors.

PROBLEM

You are to develop a data processing system to process strings of characters. The application will allow a client to communicate with all the other clients connected to the server.

You are to design and implement a client/server application where the server will respond to different client requests. In essence, you are to create a one-on-one chat system that will support some of functionality available in systems such as WhatsApp, Facebook, WeChat, and Messenger, for instance. The concept of office chat that targets group chats such as Slack is not considered here. Your solution must support 6 users, and a chat history. The exercise is structured in two parts which are referred here as milestones. However, a single submission including both milestones is to be uploaded.

MILESTONES

Milestone-1 (Error Handling)

You are to use the *"encDec.h"* library you have developed for Project-1 and include error detection by CRC and error detection and correction by Hamming.

Milestone-2 (Client/Server)

You are to develop a client/server application to support a one-to-one chat system based on the solution you have constructed for Milestone-1.

Milestones due dates

Both milestones are due on Thursday, November 9th, by 11:59 PM. However, it is recommended that Milestore-1 be completed by Thursday, October 26th.

COMMUNICATING WITH SOCKETS

A socket is a generalized interprocess communication channel. Like a pipe, a socket is represented as a file descriptor. Unlike pipes sockets support communication between unrelated processes, and even between processes running on different machines that communicate over a network. In the GNU C Library, the header file sys/socket.h exists regardless of the operating system, and the socket functions always exist, but if the system does not really support sockets these functions always fail.

ERROR HANDLING

You are to include both an error detection component by means of **CRC** and an error detection and/or correction by using **Hamming.** You are to use

CRC-32 = $x^{32}+ x^{26}+ x^{23}+ x^{22}+ x^{16}+ x^{12}+ x^{11}+ x^{10}+ x^{8}+ x^{7}+ x^{5}+ x^{4}+ x^{2}+ x + 1$

and apply **CRC-32** to the **entire message** component of the frame (entire frame) for CRC checking and only to the **data contents** of the message for Hamming. You are to use the following for determining parity bits:

1. Set a parity bit to 1 if the total number of ones in the positions it checks is odd.
2. Set a parity bit to 0 if the total number of ones in the positions it checks is even.

Your solution must

1. Include a way to allow the user to include errors.
   a. Errors are to be checked during the receiving of every frame transmitted.
2. Display to the user the location where the error was inserted. Note that this characteristic is necessary for both debugging during your coding as well as for checking the correctness of your solution during grading your work. It is recommended that you design your error detection as well as correction components independently of your client/server module.
3. Display to the user a CRC transmission error message for the CRC detected error.
4. Display to the user a Hamming transmission error message when a single bit is corrupted and also a message to inform the user about the successful single bit error correction done by the Hamming algorithm.
5. Include a control mechanism to allow the user to select either CRC or Hamming before the transmission process takes place.

TESTING

You are to use three different text files for testing. The testing file used for project-1 as well as the ***Machine representation of the source code*** text are to be used as files. Your third file must be long enough to require many frames to be created. Your testing must include the following:

1. An error detection by CRC,
2. An error detection by Hamming, and
3. An error detection and correction by Hamming.

INPUT TEST FILE (*intext.txt*)

You are encouraged to test your solution with large data sets. The reason for this is to show that your solution can handle multiple frames. For testing your prototype, you are to name your input file as *intext.txt* and populate it with text. You are to use the following text during the early stages of your prototype development.

Machine representation of the source code*.*
Source code represents the part of process that contains the programming
language itself. You may use a text editor to write your source code file. A compiler will be used to produce a machine representation of your source code. Such
representation may show your code as hexadecimal or
binary formats. The resulting hexadecimal code will contain combinations of numbers such 1 3 5 8 2 4 6 9 12 21 13 31 14 41 15 51 16 61 17 71 18 81 19 91 100 or
combinations of characters and numbers such as 1A3B4C0DEFA, for example.

OUTPUT

The final data received by the client must be stored in a file named *result.txt*.

CLIENT/SERVER DYNAMICS

The server must always start before all clients. All messages to any of the clients must be automatically forward to the server first. It is the responsibility of the server to forward the received messages to the target client.

All clients must register to the server. This means the server must have the knowledge of all valid clients' identification information. All information related to clients will be destroyed whenever the server is not available (server shutdown).

DEFINITION OF THE LANGUAGE/PROTOCOL

Before implementing the language used by the chat application, we have to know which instructions will be transmitted from a client to another as well as instructions that contain specific meaning to the server.

The dynamics of the client/server application includes the following:

- A client must register to the application through the server, by sending his/her login to the server. The login information needs to be unique. A combination of 8 characters is to be used as client identification. Letters and digits are to be used but client identification must always start with a letter.
- A client not identified can't use the chat service until recognized by the server.
- The server must always start before all clients. All messages to any of the clients must be automatically forward it to the server first. It is the responsibility of the server to forward the received messages to the target client.
- The server forwards the list of clients registered to each client current using the service.
- The server is responsible for storing the chat history between clients that are logged to the system.
- A client can send a text message to any client registered. Clients can type messages via the console and also copy and paste the contents of a text file.
- The server can disconnect any client and can send information to any client.
- All information related to clients will be destroyed whenever the server is not available (server shutdown).
- The server deletes the history of clients as they log out. The server also deletes the logged out client from the list of available clients and deallocates the resources assigned to the logged out client. All deallocated resources including clients' identification information must be made available for use by the system.
- The communication history of two clients is stored in the server by means of text files. Such files are to be named as follows: *ClientFromClientTo.txt* and *ClientToClientFrom.txt*. Assume, for instance, that clients C1 and C2 start a chat and that C1 sends a message to C2. The server will create the following files: C1C2.txt, and C2C1.txt to store their conversations. When, say, for instance, C2 logs out, all **.txt** files starting with C2 will be deleted.

The client/server communication will follow the Encoding/Decoding syntax:

<TAG>value</TAG>

where TAG represents a reserved word associated to an instruction and value is the value of the instruction. Tags need not be encoded for transmission.

Tags used to implement the communication language are:

- MSG : text message from a client to another
- LOGIN : authentication of a client to the server.
- LOGOUT: disconnect a client.
- LOGIN_LIST : contains a list of clients current using the application.
- INFO: Information message from the server.
- TO: Station where message is to be sent.
- FROM: Station that generated the message.
- BODY: Text contents of the message.
- ENCODE: Determines if either CRC or Hamming is used.

Consider a chat between client c1 and clients c2 and c3. The tag encodings that follow illustrate this activity where the # symbol is used as comment.

```
 <LOGIN> c1 </LOGIN> # Client c1 sends a login request to the server.

       <info> Welcome to 500-Chat </info> # Server confirms c1 is logged in.

 <LOGIN> c2 </LOGIN> # Client c2 sends a login request to the server.

       <info> Welcome to 500-Chat </info> # Server confirms c2 is logged in.

<LOGIN_LIST> </LOGIN_LIST> # Either c1 or c2 requested the current users logged in.

       <info> c1, c2 </info> # Server informs that c1, and c2 are logged i

 <MSG><ENCODE> CRC </ENCODE>

         <FROM>c1</FROM>

          <TO>c2</TO>

         <BODY>text</BODY>

 </MSG> # c1 sends message to c2 and sets CRC for error detection. The server

           # will process the MSG command and will forward the text message to

           # the user referred in the destination portion of the message.

 <LOGIN> c3 </LOGIN> # Client c3 sends a login request to the server.

<LOGOUT> c1 </LOGOUT> # Client c1 sends a logout request to the server.

<info> Server shutdown in 5 minutes </info> # Server informs it will shut down in 5 minutes.
```

SOFTWARE TO BE USED

You are to use the C programming language, any distribution of Linux, the "*encDec.h"* library created for project-1, and the "*sys/socket.h"* supporting library for the use of the GNU C library.

SAMPLE CODE

```
/* Program: server.c
 * A simple TCP server using sockets.
 * Server is executed before Client.
 * Port number is to be passed as an argument.
 *
 * To test: Open a terminal window.
 * At the prompt ($ is my prompt symbol) you may
 * type the following as a test:
 *
 * $ ./server 54554
 * Run client by providing host and port
 *
 *
 */
#include <arpa/inet.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

void error(const char *msg)
{
    perror(msg);
    exit(1);
}

int main(int argc, char *argv[])
{
     int sockfd, newsockfd, portno;
     socklen_t clilen;
     char buffer[256];
     struct sockaddr_in serv_addr, cli_addr;
     int n;
     if (argc < 2) {
         fprintf(stderr,"ERROR, no port provided\n");
         exit(1);
     }
     fprintf(stdout, "Run client by providing host and port\n");
     sockfd = socket(AF_INET, SOCK_STREAM, 0);
     if (sockfd < 0)
        error("ERROR opening socket");
     bzero((char *) &serv_addr, sizeof(serv_addr));
     portno = atoi(argv[1]);
     serv_addr.sin_family = AF_INET;
     serv_addr.sin_addr.s_addr = INADDR_ANY;
     serv_addr.sin_port = htons(portno);
     if (bind(sockfd, (struct sockaddr *) &serv_addr,
              sizeof(serv_addr)) < 0)
              error("ERROR on binding");
     listen(sockfd,5);
     clilen = sizeof(cli_addr);
     newsockfd = accept(sockfd,
                 (struct sockaddr *) &cli_addr,
                 &clilen);
     if (newsockfd < 0)
        error("ERROR on accept");
     bzero(buffer,256);
     n = read(newsockfd,buffer,255);
     if (n < 0)
        error("ERROR reading from socket");
     printf("Here is the message: %s\n",buffer);
     n = write(newsockfd,"I got your message",18);
     if (n < 0)
        error("ERROR writing to socket");
     close(newsockfd);
     close(sockfd);
     return 0;
}

/*
 * Simple client to work with server.c program.
 * Host name and port used by server are to be
 * passed as arguments.
 *
 * To test: Open a terminal window.
 * At prompt ($ is my prompt symbol) you may
 * type the following as a test:
 *
 * $./client 127.0.0.1 54554
 * Please enter the message: Programming with sockets is fun!
 * I got your message
 * $
 *
 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void error(const char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
    int sockfd, portno, n;
    struct sockaddr_in serv_addr;
    struct hostent *server;

    char buffer[256];
    if (argc < 3) {
       fprintf(stderr,"usage %s hostname port\n", argv[0]);
       exit(0);
    }
    portno = atoi(argv[2]);
    sockfd = socket(AF_INET, SOCK_STREAM, 0);
    if (sockfd < 0)
        error("ERROR opening socket");
    server = gethostbyname(argv[1]);
    if (server == NULL) {
        fprintf(stderr,"ERROR, no such host\n");
        exit(0);
    }
    bzero((char *) &serv_addr, sizeof(serv_addr));
    serv_addr.sin_family = AF_INET;
    bcopy((char *)server->h_addr,
         (char *)&serv_addr.sin_addr.s_addr,
         server->h_length);
    serv_addr.sin_port = htons(portno);
    if (connect(sockfd,(struct sockaddr *)
&serv_addr,sizeof(serv_addr)) < 0)
        error("ERROR connecting");
    printf("Please enter the message: ");
    bzero(buffer,256);
    fgets(buffer,255,stdin);
    n = write(sockfd,buffer,strlen(buffer));
    if (n < 0)
         error("ERROR writing to socket");
    bzero(buffer,256);
    n = read(sockfd,buffer,255);
    if (n < 0)
         error("ERROR reading from socket");
    printf("%s\n",buffer);
    close(sockfd);
    return 0;
}
```

DOCUMENTATION

Your program should be developed using GNU versions of the C compiler. It should be layered, modularized, and well commented. The following is a tentative marking scheme and what is expected to be submitted for this assignment:

1. External Documentation (as many pages necessary to fulfill the requirements listed below.) including the following:
    a. Title page
    b. A table of contents
    c. [12%] System documentation
        i. A high-level data flow diagram for the system
        ii. A list of routines and their brief descriptions
        iii. Implementation details
    d. [13%] Test documentation
        i. [1%] How you tested your program
        ii. [12%] Testing outputs
            - Tag language commands and responses
            - CRC error detection
            - Hamming detection and correction
            - One to one chat
    e. [3%] User documentation
        i. Where is your source
        ii. How to run your program
        iii. Describe parameter (if any)

2. Source Code
    a. [70% total] Correctness
        i. [10%] Tag language implementation
        ii. [15%] CRC error detection
        iii. [15%] Hamming error detection and correction
        iv. [10%] Single client and server communication
            - Text file used by the client.
        v. [20%] Single client to client chat
            - Text files used by both clients.
    b. [2%] Programming style
        i. Layering
        ii. Readability

iii. Comments
iv. Efficiency

WHAT TO SUBMIT

No group submissions will be accepted. The following are to be submitted through **Brightspace**:

1. Your documentation for project-2. This document must be typeset and saved in MS Word.
2. Copies of all your source code files as well as their executables.
3. Copies of all data used for the testing of your solution.
4. All input files used and their generated output files.

You are to place all files that are related to your solution in a .zip file. Your .zip file must follow the format: *500 Project-2 Your Name*. Marks will be deducted if you do not follow this requirement.