PURPOSE

Develop a practical understanding of task collaboration using socket programming, semaphores, multithreading, 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. Such requests may be by either providing the IP address of the server or the name of the host where the server is executing. Transmission errors will be handled by applying the CRC algorithm to detect errors and Hamming to both detect and correct a single bit errors.

PROBLEM

You are to develop a data processing system to process strings of characters. The solution must replace all instances of lowercase vowels with uppercase ones and append to the last set of strings the sum of all numbers found.

You are to design and implement a client/server application where the server will respond to different client requests by creating two dedicated new processes. One process will read from the socket, decode the received messages, and will create seven threads. The other process will encode the message processed by the threads and will write the resulting data to the socket. All communications between both the client and the server will be encoded according to the format defined in the project 2 format.

THE SERVER

Accepts multiple client requests and for each request creates two *child* processes. These processes are to be named **serverDecoder** and **ServerEncoder**. The **ServerDecoder** will read from the socket, decode the received messages, and will create the following seven threads:

> 1   The **charA** thread will read data provided by the decoder process and will replace all lowercase **a** character with its corresponding uppercase. It will share the data received with the *charE* thread through a queue of messages.

2   The **charE** thread component will scan the data shared and replace all lowercase **e** character with uppercase E. It will then share the modified data with the *charI* thread through another queue of messages.

3   The **charI** thread component will scan the data shared and replace all lowercase **i** character with uppercase **I**. It will then share the modified data with the *charO* thread through another queue of messages.

4   The **charO** thread component will scan the data shared and replace all lowercase **o** character with uppercase **O**. It will then share the modified data with the *charU* thread through another queue of messages.

5   The **charU** thread component will scan the data shared and replace all lowercase **u** character with uppercase **U**. It will then share the modified data with the *digit* thread through another queue of messages.

6   The **digit** thread component will scan the data shared and computes the sum of all digits found. I will then share the processed data with the *writer* thread through another queue of messages.

7   The **writer** thread will share the data received with the **serverEncoder** process.

The **serverEncoder** will encode the resulting data and will share it with the client through the socket connection.

CLIENT/SERVER IMPLEMENTATION DETAILS
1.   You must develop a module that implements a queue of character strings.
2.   This structure will be an array of pointers to strings with integers (pointers) to indicate the head and tail of the list.
3.   The maximum size of the buffer array will be 10.
4.   Threads should terminate when end of input data is reached.

THE CLIENT
Creates two processes, the **clientEncoder** and the **clientDecoder**. The **clientEncoder** will open input files, encode file contents, and will write data to the socket. The **clientDecoder** will read data from the socket, decode data, and will write the decoded data to a file.

INPUT TEST FILE (*intext.txt*)
You are encouraged to test your solution with large data sets. For testing your prototype, you are to name your input file as *intext.txt* and populate it with the following contents:

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*.

ERROR HANDLING

What to Do

You are to modify your solution for Project 2 to include 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 it to the entire message component of the frame for CRC checking and only to the data contents of the message for Hamming.
Your solution must

1. Include a way to allow the user to include errors.
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.
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 more than 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. A control mechanism such as a flag to allow the user to select either CRC or Hamming before the transmission process takes place.

Testing

You are to use two different text files for testing. The testing file used for project2 is to be used as one of such files. Your second 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.

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.
```

```c
 *
 * 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,
```

```c
                (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;
```

```c
    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. [20%] 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. [5%] Test documentation
  i. How you tested your program
  ii. Testing outputs
e. [5%] User documentation
  i. Where is your source
  ii. How to run your program
  iii. Describe parameter (if any)

2. Source Code
  a. [65%] Correctness
  b. [5%] Programming style
    i. Layering
    ii. Readability
    iii. Comments
    iv. Efficiency

WHAT TO SUBMIT

The following are to be submitted through **Blackboard**:

1. Your documentation for project 3. 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 Project3 Your Name*. Marks will be deducted if you do not follow this requirement.