National University of Computer and Emerging Sciences



Lab # 03

# Task 01

## server.cpp

// Server side implementation of UDP client-server model

#include <bits/stdc++.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define PORT 8080

#define MAXLINE 1024

using namespace std;

// Driver code

int main()

{

int sockfd;

char buffer[MAXLINE];

const char \*hello = "Hello from server";

struct sockaddr\_in servaddr, cliaddr;

// Creating socket file descriptor

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

perror("socket creation failed");

exit(EXIT\_FAILURE);

}

memset(&servaddr, 0, sizeof(servaddr));

memset(&cliaddr, 0, sizeof(cliaddr));

// Filling server information

servaddr.sin\_family = AF\_INET; // IPv4

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

servaddr.sin\_port = htons(PORT);

// Bind the socket with the server address

if (bind(sockfd, (const struct sockaddr \*)&servaddr,

sizeof(servaddr)) < 0)

{

perror("bind failed");

exit(EXIT\_FAILURE);

}

socklen\_t len;

int n;

len = sizeof(cliaddr); //len is value/result

n = recvfrom(sockfd, (char \*)buffer, MAXLINE,

MSG\_WAITALL, (struct sockaddr \*) &cliaddr,

&len);

buffer[n] = '\0';

printf("Client : %s\n", buffer);

sendto(sockfd, (const char \*)hello, strlen(hello),

MSG\_CONFIRM, (const struct sockaddr \*) &cliaddr,

len);

cout << "Hello message sent." << endl;

return 0;

}

## client.cpp

// Client side implementation of UDP client-server model

#include <bits/stdc++.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define PORT 8080

#define MAXLINE 1024

using namespace std;

// Driver code

int main()

{

int sockfd;

char buffer[MAXLINE];

const char \*hello = "Hello from client";

struct sockaddr\_in servaddr;

// Creating socket file descriptor

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

perror("socket creation failed");

exit(EXIT\_FAILURE);

}

memset(&servaddr, 0, sizeof(servaddr));

// Filling server information

servaddr.sin\_family = AF\_INET;

servaddr.sin\_port = htons(PORT);

servaddr.sin\_addr.s\_addr = INADDR\_ANY;

int n;

socklen\_t len;

sendto(sockfd, (const char \*)hello, strlen(hello),

MSG\_CONFIRM, (const struct sockaddr \*) &servaddr,

sizeof(servaddr));

cout << "Hello message sent." << endl;

n = recvfrom(sockfd, (char \*)buffer, MAXLINE,

MSG\_WAITALL, (struct sockaddr \*) &servaddr,

&len);

buffer[n] = '\0';

cout << "Server :" << buffer << endl;

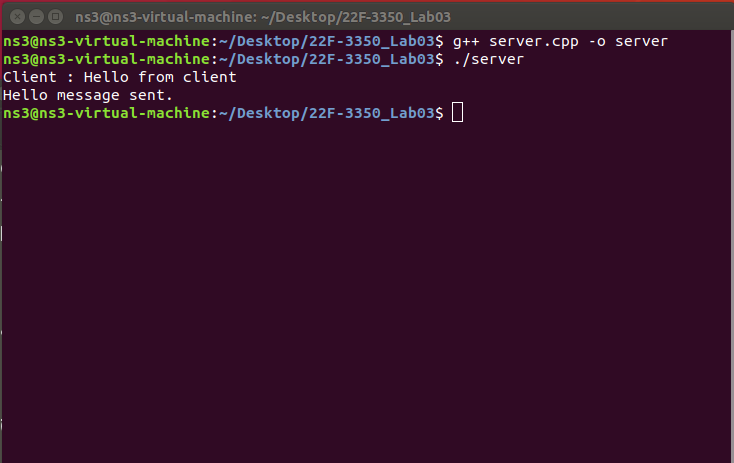
close(sockfd);

return 0;

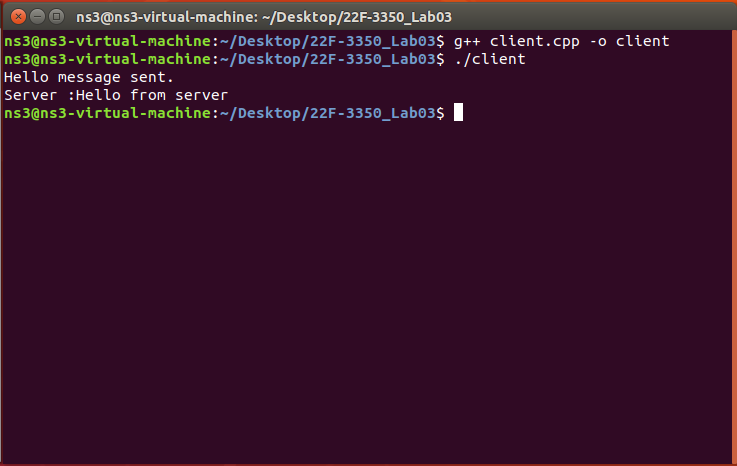
}

# Output

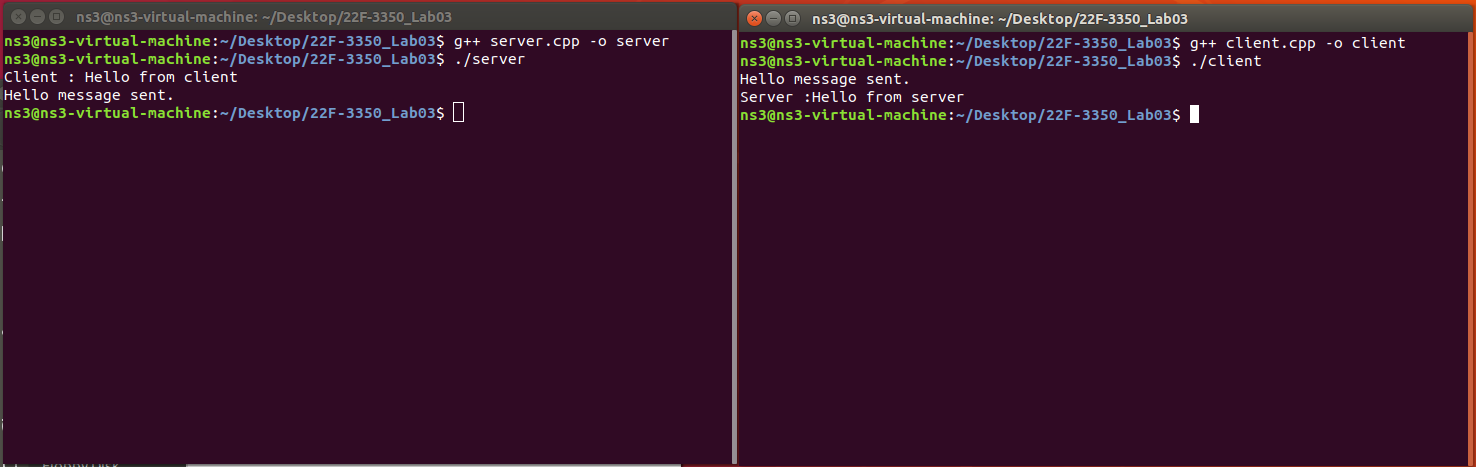
## Server:



## Client:



## Both:



# Task 02

## server.cpp

#include <bits/stdc++.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#include <fstream>

#define PORT 8080

#define MAXLINE 1024

using namespace std;

// Function to copy content from one file to another

bool copyFileContent(const string &srcFile, const string &destFile)

{

// Open the source file in binary mode for reading

ifstream src(srcFile, ios::binary);

if (!src.is\_open())

{

// If the source file cannot be opened, return false

cerr << "Error: Unable to open source file." << endl;

return false;

}

// Open the destination file in binary mode for writing (truncate if it exists)

ofstream dest(destFile, ios::binary | ios::trunc);

if (!dest.is\_open())

{

// If the destination file cannot be opened, return false

cerr << "Error: Unable to open destination file." << endl;

return false;

}

// Copy the content of the source file to the destination file

dest << src.rdbuf();

return true; // Return true if the copy was successful

}

int main()

{

int sockfd; // Socket file descriptor

char buffer[MAXLINE]; // Buffer to store incoming and outgoing data

struct sockaddr\_in servaddr, cliaddr; // Structures to hold server and client addresses

// Create a UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Initialize server and client address structures to zero

memset(&servaddr, 0, sizeof(servaddr));

memset(&cliaddr, 0, sizeof(cliaddr));

// Fill in the server's address information

servaddr.sin\_family = AF\_INET; // Use IPv4

servaddr.sin\_addr.s\_addr = INADDR\_ANY; // Bind to all available interfaces

servaddr.sin\_port = htons(PORT); // Set the port number

// Bind the socket to the server's address

if (bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

{

perror("Bind failed");

exit(EXIT\_FAILURE);

}

socklen\_t len = sizeof(cliaddr); // Length of the client address structure

int n; // Number of bytes received

// Receive the file names from the client

n = recvfrom(sockfd, (char \*)buffer, MAXLINE, MSG\_WAITALL, (struct sockaddr \*)&cliaddr, &len);

buffer[n] = '\0'; // Null-terminate the received data

// Parse the received file names (format: "source.txt,dest.txt")

string input(buffer);

size\_t pos = input.find(",");

if (pos == string::npos)

{

// If the input format is invalid, log an error and exit

cerr << "Invalid input format from client." << endl;

close(sockfd);

return -1;

}

string srcFile = input.substr(0, pos); // Extract the source file name

string destFile = input.substr(pos + 1); // Extract the destination file name

// Attempt to copy the content of the source file to the destination file

bool success = copyFileContent(srcFile, destFile);

// Prepare the response message based on the success or failure of the copy operation

const char \*response;

if (success)

{

response = "File copied successfully.";

}

else

{

response = "Failed to copy file.";

}

// Send the response back to the client

sendto(sockfd, response, strlen(response), MSG\_CONFIRM, (const struct sockaddr \*)&cliaddr, len);

cout << "Response sent to client: " << response << endl;

// Close the socket and exit

close(sockfd);

return 0;

}

## client.cpp

#include <bits/stdc++.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define PORT 8080

#define MAXLINE 1024

using namespace std;

int main()

{

int sockfd; // Socket file descriptor

char buffer[MAXLINE]; // Buffer to store incoming and outgoing data

struct sockaddr\_in servaddr; // Structure to hold server address

// Create a UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Initialize the server address structure to zero

memset(&servaddr, 0, sizeof(servaddr));

// Fill in the server's address information

servaddr.sin\_family = AF\_INET; // Use IPv4

servaddr.sin\_port = htons(PORT); // Set the port number

servaddr.sin\_addr.s\_addr = INADDR\_ANY; // Connect to the local server

// Prompt the user to enter the source and destination file names

string srcFile, destFile;

cout << "Enter source file name: ";

cin >> srcFile;

cout << "Enter destination file name: ";

cin >> destFile;

// Combine the file names into a single string separated by a comma (e.g., "source.txt,dest.txt")

string message = srcFile + "," + destFile;

// Send the file names to the server

sendto(sockfd, message.c\_str(), message.size(), MSG\_CONFIRM, (const struct sockaddr \*)&servaddr, sizeof(servaddr));

cout << "File names sent to server." << endl;

socklen\_t len = sizeof(servaddr); // Length of the server address structure

int n; // Number of bytes received

// Receive the server's response

n = recvfrom(sockfd, (char \*)buffer, MAXLINE, MSG\_WAITALL, (struct sockaddr \*)&servaddr, &len);

buffer[n] = '\0'; // Null-terminate the received data

// Display the server's response

cout << "Server response: " << buffer << endl;

// Close the socket and exit

close(sockfd);

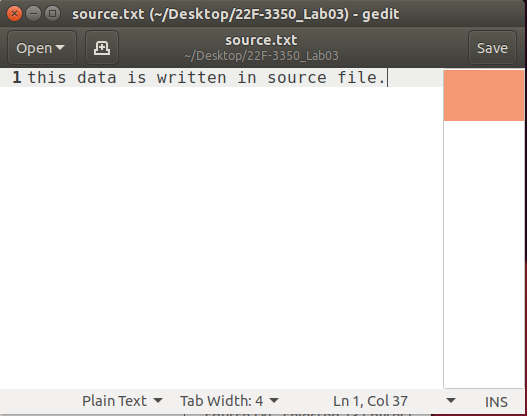
return 0;

}

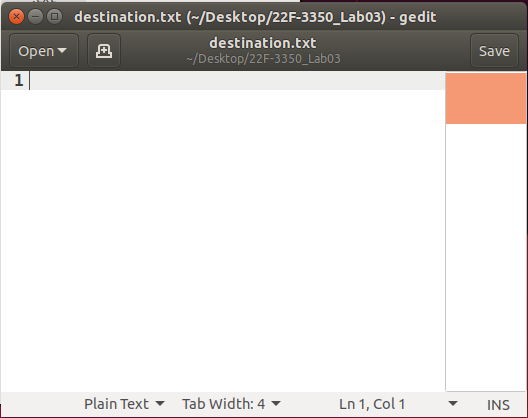
# Output

## Before:

## source.txt

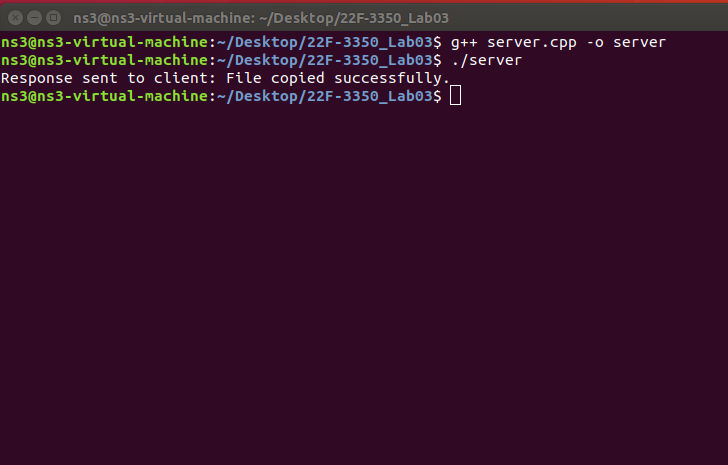


## destination.txt

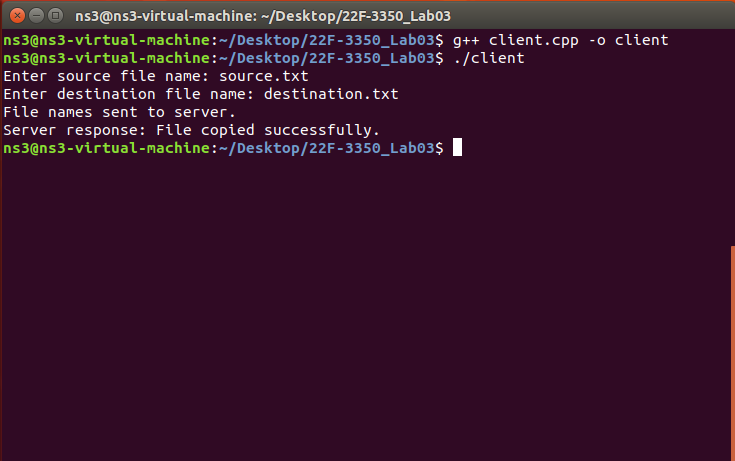


## Script:

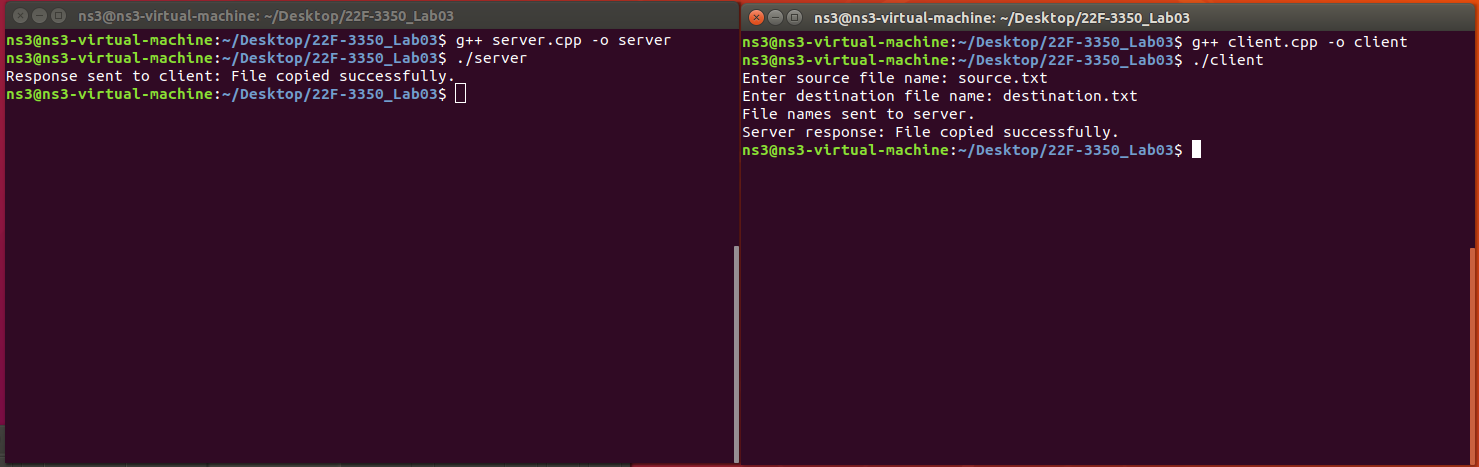
## Server:



## Client:

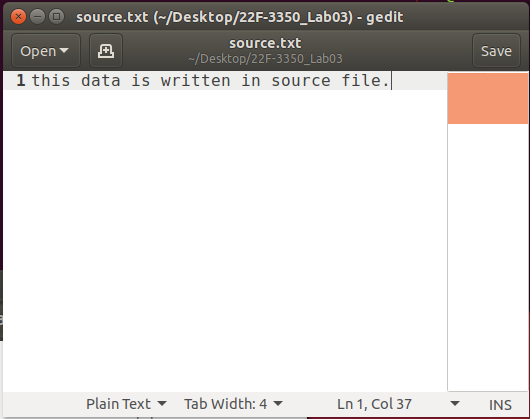


## Both:

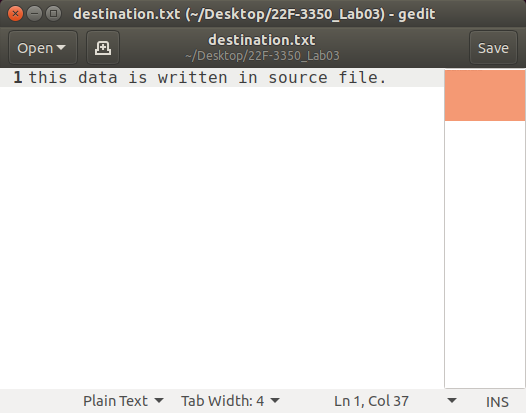


## After

## source.txt



## destination.txt



# Task 03

## server.cpp

#include <bits/stdc++.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define PORT 8080

#define MAXLINE 1024

using namespace std;

// Function to calculate the sum of even-positioned digits

int calculateEvenPositionedSum(const string &number)

{

int sum = 0;

// Iterate through the digits, considering 1-based positions

for (size\_t i = 0; i < number.length(); ++i)

{

if ((i + 1) % 2 == 0)

{ // Check if the position is even

sum += number[i] - '0'; // Convert character to integer

}

}

return sum;

}

int main()

{

int sockfd; // Socket file descriptor

char buffer[MAXLINE]; // Buffer to store incoming and outgoing data

struct sockaddr\_in servaddr, cliaddr; // Structures to hold server and client addresses

// Create a UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Initialize server and client address structures to zero

memset(&servaddr, 0, sizeof(servaddr));

memset(&cliaddr, 0, sizeof(cliaddr));

// Fill in the server's address information

servaddr.sin\_family = AF\_INET; // Use IPv4

servaddr.sin\_addr.s\_addr = INADDR\_ANY; // Bind to all available interfaces

servaddr.sin\_port = htons(PORT); // Set the port number

// Bind the socket to the server's address

if (bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

{

perror("Bind failed");

exit(EXIT\_FAILURE);

}

socklen\_t len = sizeof(cliaddr); // Length of the client address structure

int n; // Number of bytes received

cout << "Server is running and waiting for client..." << endl;

while (true)

{

// Receive the number from the client

n = recvfrom(sockfd, buffer, MAXLINE, MSG\_WAITALL, (struct sockaddr \*)&cliaddr, &len);

buffer[n] = '\0'; // Null-terminate the received data

cout << "Received number from client: " << buffer << endl;

// Calculate the sum of even-positioned digits

string number(buffer);

int sum = calculateEvenPositionedSum(number);

// Prepare the response message

string response = "Sum of even-positioned digits: " + to\_string(sum);

// Send the response back to the client

sendto(sockfd, response.c\_str(), response.size(), MSG\_CONFIRM, (const struct sockaddr \*)&cliaddr, len);

cout << "Response sent to client: " << response << endl;

}

close(sockfd);

return 0;

}

## client.cpp

#include <bits/stdc++.h>

#include <stdlib.h>

#include <unistd.h>

#include <string.h>

#include <sys/types.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <netinet/in.h>

#define PORT 8080

#define MAXLINE 1024

using namespace std;

int main()

{

int sockfd; // Socket file descriptor

char buffer[MAXLINE]; // Buffer to store incoming and outgoing data

struct sockaddr\_in servaddr; // Structure to hold server address

// Create a UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Initialize the server address structure to zero

memset(&servaddr, 0, sizeof(servaddr));

// Fill in the server's address information

servaddr.sin\_family = AF\_INET; // Use IPv4

servaddr.sin\_port = htons(PORT); // Set the port number

servaddr.sin\_addr.s\_addr = INADDR\_ANY; // Connect to the local server

// Prompt the user to enter a decimal integer

string number;

cout << "Enter a decimal integer: ";

cin >> number;

socklen\_t len = sizeof(servaddr); // Length of the server address structure

int n; // Number of bytes received

// Send the number to the server

sendto(sockfd, number.c\_str(), number.size(), MSG\_CONFIRM, (const struct sockaddr \*)&servaddr, sizeof(servaddr));

cout << "Number sent to server." << endl;

// Receive the server's response

n = recvfrom(sockfd, buffer, MAXLINE, MSG\_WAITALL, (struct sockaddr \*)&servaddr, &len);

buffer[n] = '\0'; // Null-terminate the received data

// Display the server's response

cout << "Server response: " << buffer << endl;

close(sockfd);

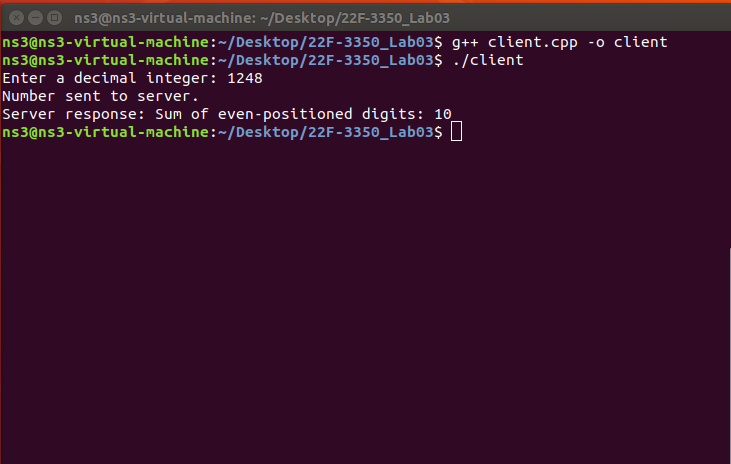
return 0;

}

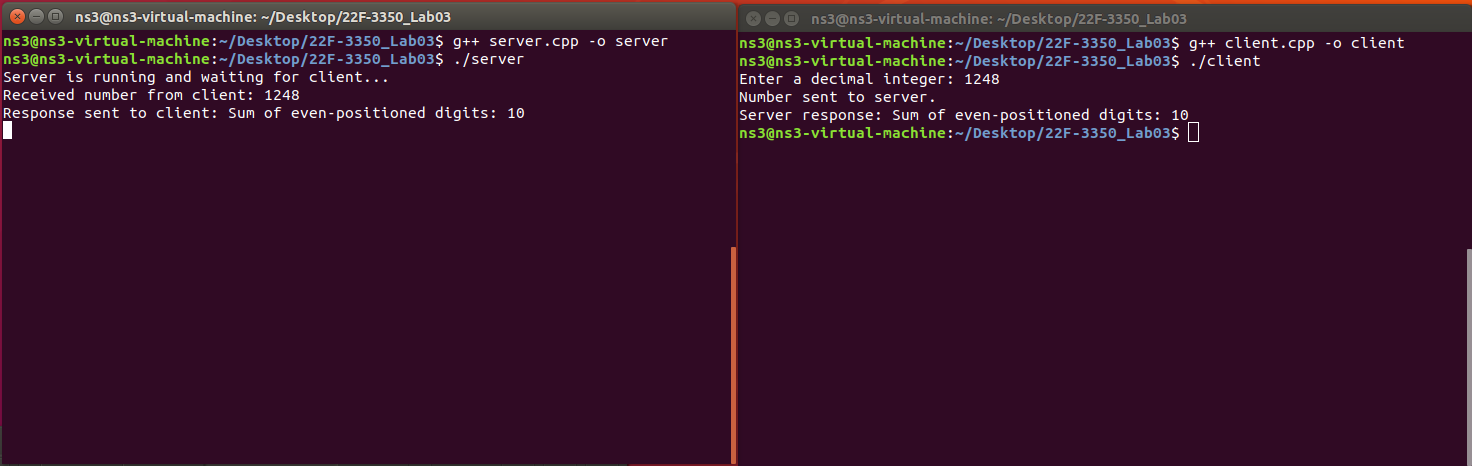
# Output

## Server:

## Client:



## Both:



# Task 04

## server.cpp

#include <bits/stdc++.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <fstream>

#define PORT 8080

#define MAXLINE 1024

using namespace std;

// Function to decrypt the file content using bitwise XOR

string decryptFileContent(const string &content, char key)

{

string decryptedContent = content;

for (char &ch : decryptedContent)

{

ch ^= key; // XOR each character with the key

}

return decryptedContent;

}

int main()

{

int sockfd; // Socket file descriptor

char buffer[MAXLINE]; // Buffer to store incoming data

struct sockaddr\_in servaddr, cliaddr; // Structures to hold server and client addresses

// Create a UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Initialize server and client address structures

memset(&servaddr, 0, sizeof(servaddr));

memset(&cliaddr, 0, sizeof(cliaddr));

// Fill in the server's address information

servaddr.sin\_family = AF\_INET; // Use IPv4

servaddr.sin\_addr.s\_addr = INADDR\_ANY; // Bind to all available interfaces

servaddr.sin\_port = htons(PORT); // Set the port number

// Bind the socket to the server's address

if (bind(sockfd, (const struct sockaddr \*)&servaddr, sizeof(servaddr)) < 0)

{

perror("Bind failed");

exit(EXIT\_FAILURE);

}

socklen\_t len = sizeof(cliaddr); // Length of the client address structure

int n; // Number of bytes received

cout << "Server is running and waiting for client..." << endl;

// Receive the encrypted file content from the client

n = recvfrom(sockfd, buffer, MAXLINE, MSG\_WAITALL, (struct sockaddr \*)&cliaddr, &len);

buffer[n] = '\0'; // Null-terminate the received data

cout << "Received encrypted file content from client." << endl;

// Get the decryption key from the user

char key;

cout << "Enter decryption key (a single character): ";

cin >> key;

// Decrypt the file content

string encryptedContent(buffer);

string decryptedContent = decryptFileContent(encryptedContent, key);

// Write the decrypted content to a new file

ofstream outputFile("output.txt");

if (!outputFile.is\_open())

{

cerr << "Error: Unable to open output file." << endl;

close(sockfd);

return -1;

}

outputFile << decryptedContent;

outputFile.close();

cout << "Decrypted file content written to output.txt." << endl;

close(sockfd);

return 0;

}

## client.cpp

#include <bits/stdc++.h>

#include <sys/socket.h>

#include <arpa/inet.h>

#include <unistd.h>

#include <fstream>

#define PORT 8080

#define MAXLINE 1024

using namespace std;

// Function to encrypt the file content using bitwise XOR

string encryptFileContent(const string &content, char key)

{

string encryptedContent = content;

for (char &ch : encryptedContent)

{

ch ^= key; // XOR each character with the key

}

return encryptedContent;

}

int main()

{

int sockfd; // Socket file descriptor

struct sockaddr\_in servaddr; // Structure to hold server address

// Create a UDP socket

if ((sockfd = socket(AF\_INET, SOCK\_DGRAM, 0)) < 0)

{

perror("Socket creation failed");

exit(EXIT\_FAILURE);

}

// Initialize the server address structure

memset(&servaddr, 0, sizeof(servaddr));

servaddr.sin\_family = AF\_INET; // Use IPv4

servaddr.sin\_port = htons(PORT); // Set the port number

servaddr.sin\_addr.s\_addr = INADDR\_ANY; // Connect to the local server

// Read the file content

ifstream inputFile("input.txt");

if (!inputFile.is\_open())

{

cerr << "Error: Unable to open input file." << endl;

close(sockfd);

return -1;

}

string fileContent((istreambuf\_iterator<char>(inputFile)), istreambuf\_iterator<char>());

inputFile.close();

// Get the encryption key from the user

char key;

cout << "Enter encryption key (a single character): ";

cin >> key;

// Encrypt the file content

string encryptedContent = encryptFileContent(fileContent, key);

// Send the encrypted content to the server

sendto(sockfd, encryptedContent.c\_str(), encryptedContent.size(), MSG\_CONFIRM,

(const struct sockaddr \*)&servaddr, sizeof(servaddr));

cout << "Encrypted file content sent to server." << endl;

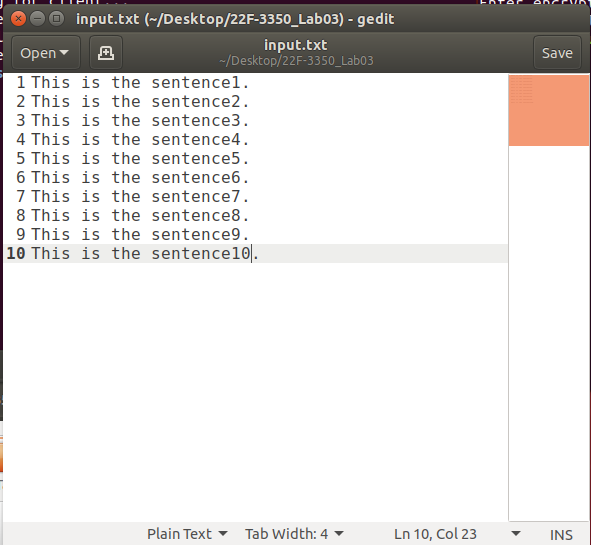
close(sockfd);

return 0;

}

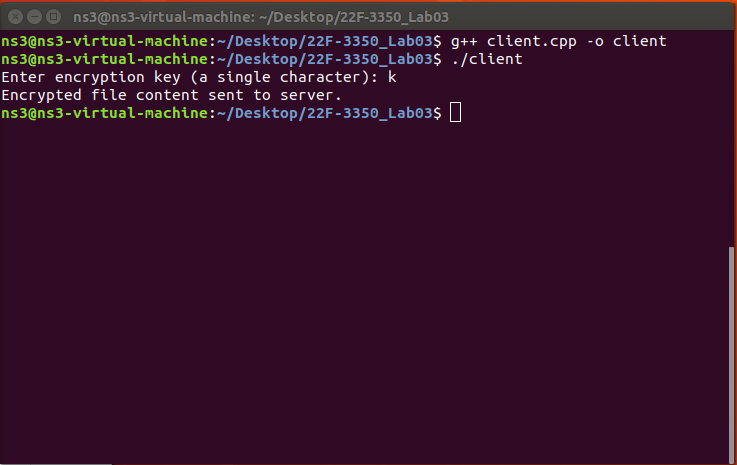
# Output

## input.txt



## Server:

## Client:



## Both:

## output.txt

