**COMP 7005**

**Computer Networks & Protocols**

*Assignment-02*

*Design*

##### Anmol Mittal A01397754

##### February 9th, 2025

##### Course Reference Number (CRN): 92522

[**Purpose** 4](#_Toc189950226)

[**Data Types** 4](#_Toc189950227)

[Arguments 4](#_Toc189950228)

[Client 4](#_Toc189950229)

[Server 4](#_Toc189950230)

[Settings 5](#_Toc189950231)

[Client 5](#_Toc189950232)

[Server 5](#_Toc189950233)

[Context Client 5](#_Toc189950234)

[Server 6](#_Toc189950235)

[**Functions** 6](#_Toc189950236)

[Client 6](#_Toc189950237)

[Server 7](#_Toc189950238)

[**States** 8](#_Toc189950239)

[Client 8](#_Toc189950240)

[Server 8](#_Toc189950241)

[**State Table** 9](#_Toc189950242)

[Client 9](#_Toc189950243)

[Server 10](#_Toc189950244)

[**State Transition Diagram** 11](#_Toc189950245)

[**Pseudocode** 12](#_Toc189950246)

[Client 12](#_Toc189950247)

[1. validate\_arguments: 12](#_Toc189950248)

[2. parse\_arguments: 12](#_Toc189950249)

[3. validate\_arguments: 13](#_Toc189950250)

[4. is\_valid\_ip: 13](#_Toc189950251)

[5. is\_valid\_port: 14](#_Toc189950252)

[6. is\_valid\_file: 14](#_Toc189950253)

[7. is\_valid\_keyword: 15](#_Toc189950254)

[8. get\_file\_size: 15](#_Toc189950255)

[9. read\_file\_content: 16](#_Toc189950256)

[11. connect\_server: 17](#_Toc189950257)

[12. send\_message\_to\_server: 17](#_Toc189950258)

[13. receive\_server: 18](#_Toc189950259)

[Server 19](#_Toc189950260)

[1. validate\_argument\_number: 19](#_Toc189950261)

[2. parse\_arguments: 20](#_Toc189950262)

[3. validate\_arguments: 20](#_Toc189950263)

[4. is\_valid\_ip: 21](#_Toc189950264)

[5. is\_valid\_port: 21](#_Toc189950265)

[6. handle\_signal: 22](#_Toc189950266)

[7. create\_server\_fd: 22](#_Toc189950267)

[8. config\_server: 22](#_Toc189950268)

[9. accept\_client\_connections: 23](#_Toc189950269)

[10. process\_client\_message: 24](#_Toc189950270)

[11. vigenere\_cipher: 24](#_Toc189950271)

[12. cleanup: 25](#_Toc189950272)

# 

# **Purpose**

The programs implement inter-process communication using network sockets. The server accepts an IP address and port as parameters to run. The client sends content along with a key that is used to encrypt the content using the Vigenère cipher. Once the server receives the key and content, it encrypts the data and sends the encrypted content back to the client.

Both server and client program accept command line argument as follows:

* ./server -ip <Server IP Address> -p <Port to run on>
* ./client -ip <Server IP Address> -p <Port> -f <Filename> -key <Keyword>

# **Data Types**

## Arguments

### Client

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| argv | char\* | arguments |
| argc | integer | number of arguments |
| program\_name | char\* | name of the program |
| ip address | char\* | encryption server’s ip address |
| port | char\* | port, the server is listening on |
| file name | char\* | name of the file to encrypt |
| key | char\* | key used for implementing encryption |

### Server

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| argv | char\* [] | arguments |
| argc | integer | number of arguments |
| program\_name | char\* | name of the program |
| ip address | char\* | server’s ip address |
| port | char\* | port, for the server to listen on |

## Settings

### Client

|  |  |  |
| --- | --- | --- |
| **Field** | **Value** | **Description** |
| ip | integer | Size of the buffer for communication. |
| port | char\* | Validated port number |
| filename | char\* | Validated filename |
| keyword | char\* | Validated keyword (no digits) |
| file\_content | char\* | Content read from the file |
| client\_fd | int | Client socket file descriptor |
| server\_addr | struct sockaddr\_in | Server address configuration |

### Server

|  |  |  |
| --- | --- | --- |
| **Field** | **Value** | **Description** |
| ip | integer | Size of the buffer for communication. |
| port | char\* | Validated port number |
| server\_fd | int | server socket file descriptor |
| server\_addr | struct sockaddr\_in | Server address configuration |
| buffer | string | Content received from the client |
| keyword | string | Keyword received from the client |

### Context Client

|  |  |  |
| --- | --- | --- |
| **Field** | **Value** | **Description** |
| File | string | Reads the file content and manages communication with the server. |
| Keyword | string | Reads key and send it to the server. |
| EXIT |  | Programs exist automatically after a successful or failed connection. |

### Server

|  |  |  |
| --- | --- | --- |
| **Field** | **Value** | **Description** |
| Handle Signal | interger | Handle SIGINT for graceful shutdown. |

# **Functions**

## Client

|  |  |
| --- | --- |
| **Function** | **Description** |
| validate\_argument\_number | Validates the number of command-line arguments passed to the program. Ensure there are exactly 8 arguments (including the program name). |
| parse\_arguments | Passes the command-line arguments into variables for IP, port, filename, and keyword. |
| validate\_arguments | Validates the parsed arguments for correctness, including IP format, port range, filename existence, and keyword constraints. |
| is\_valid\_ip | Validates the format of the IP address to ensure it's in a valid IPv4 format. |
| is\_valid\_port | Validates if the port number is a valid integer between 1 and 65535 and doesn't have leading zeros. |
| is\_valid\_file | Checks if the specified file exists, is accessible, and is non-empty. |
| is\_valid\_keyword | Validates if the keyword contains any digits (which it shouldn’t), ensuring it is non-empty and meets keyword constraints. |
| get\_file\_size | Calculates and returns the size of the file in bytes. |
| read\_file\_content | Reads the content of the file into memory and returns the content as a string. |
| create\_client\_fd | Creates and returns a new client socket descriptor for communication. |
| connect\_server | Configures the server's connection parameters and establishes a connection to the server using the provided IP address and port. |
| send\_message\_to\_server | Send a message to the server by splitting the data into chunks if necessary and ensure all the data is sent correctly. |
| receive\_server\_response | Receives and prints the server’s response to the client in chunks. Prints the encrypted message received from the server. |
| close\_socket | Safely closes the client socket to end the connection. |

## Server

|  |  |
| --- | --- |
| **Function** | **Description** |
| validate\_argument\_number | Validates the number of arguments passed to the program. |
| parse\_arguments | Parses the command-line arguments to extract the IP address and port number. |
| validate\_arguments | Validates the IP address and port number to ensure they are correct. |
| is\_valid\_ip | Checks if the provided IP address is valid (IPv4 format). |
| is\_valid\_port | Checks if the provided port number is valid (between 1 and 65535). |
| handle\_signal | Handles the SIGINT signal (e.g., Ctrl+C) for graceful shutdown of the server. |
| create\_server\_fd | Creates the server socket using IPv4 and TCP protocols |
| config\_server | Configures the server with the given IP and port, and binds the socket. |
| accept\_client\_connections | Accepts client connections and processes messages in a loop. |
| process\_client\_message | Processes the message received from the client, including handling the keyword and message data. |
| vigenere\_cipher | Encrypts the text using the Vigenère cipher with the provided keyword. |
| cleanup | Cleans up resources, including closing the server socket. |

# **States**

## Client

|  |  |
| --- | --- |
| **State** | **Description** |
| START | Start client program and read the input file. |
| VALIDATE | Check the number of arguments, validate the IP address, port, file existence, and keyword. |
| OPEN FILE | Open the specified file for reading. |
| READ | Read the content of the file into memory. |
| INIT\_SOCKET | Create and initialize the client socket for communication. |
| CONNECT | Establish a connection to the server using the provided IP and port. |
| SEND | Transmit the keyword and file content to the server. |
| RECEIVE | Receive and display the encrypted response from the server. |
| CLEANUP | Close the client socket and free allocated memory for file content. |

## Server

|  |  |
| --- | --- |
| **State** | **Description** |
| START | Start the server program and read the command-line arguments (IP and port). |
| VALIDATE | Validate the number of arguments, check for missing or incorrect IP and port. |
| INIT\_SOCKET | Create the server socket using socket () and set up necessary socket configurations (bind and listen). |
| LISTEN | Configure the server to listen for incoming client connections. |
| ACCEPT | Accept an incoming client connection using accept () and prepare for communication. |
| PROCESS | Read client messages, apply Vigenère cipher encryption, and send back the encrypted response. |
| CLEANUP | Release resources, close sockets, and gracefully shut down the server upon receiving a signal. |

# **State Table**

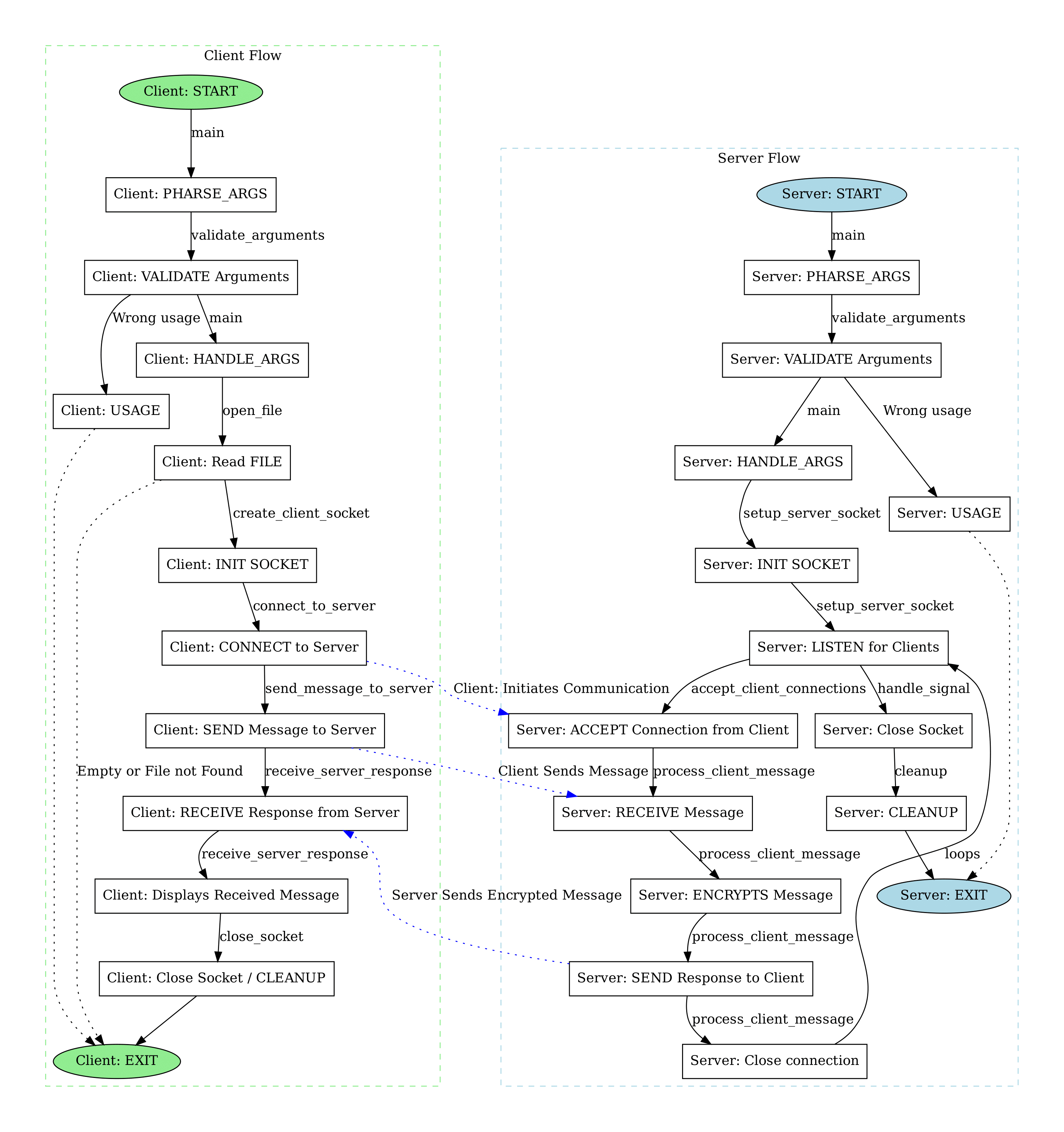
## Client

|  |  |  |
| --- | --- | --- |
| **From State** | **To State** | **Function** |
| START | VALIDATE | validate\_arguments |
| VALIDATE | OPEN FILE | prepare\_filename (parsing filename) |
| OPEN FILE | READ | open\_file (open the file) |
| READ | INIT\_SOCKET | create\_client\_socket |
| INIT\_SOCKET | CONNECT | connect\_to\_server |
| CONNECT | SEND | send\_message\_to\_server |
| SEND | RECEIVE RESPONSE | receive\_server\_response |
| RECEIVE RESPONSE | CLEANUP / CLOSE SOCKET | close\_socket |
| CLEANUP / CLOSE SOCKET | END | (exit program) |

## Server

|  |  |  |
| --- | --- | --- |
| **From State** | **To State** | **Description** |
| START | VALIDATE | validate\_arguments |
| VALIDATE | INIT\_SOCKET | setup\_server\_socket |
| INIT\_SOCKET | CONFIG\_SERVER | config\_server |
| CONFIG\_SERVER | LISTEN | listen\_for\_connections |
| LISTEN | ACCEPT | accept\_client\_connections |
| ACCEPT | PROCESS | process\_client\_message |
| PROCESS | LISTEN | return to listening for new clients |
| PROCESS | CLEANUP | handle\_signal |
| CLEANUP | EXIT | cleanup |

# **State Transition Diagram**



# **Pseudocode**

## Client

### validate\_arguments:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| argument\_count | integer | The passed Arguments |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Exits program if argument count is invalid |

#### Pseudo Code

function validate\_argument\_count(argc):

if argc != 8 + 1: // 8 arguments + program name

print\_error("Usage: -ip <IP> -p <Port> -f <File> -key <Keyword>")

exit\_program()

### parse\_arguments:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| argc | int | Number of command-line arguments |
| argv | char\*\* | Array of command-line arguments |
| ip | char\*\* | Pointer to store IP address |
| port | char\*\* | Pointer to store port number |
| filename | char\*\* | Pointer to store filename |
| keyword | char\*\* | Pointer to store keyword |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Exits program if any argument is missing |

#### Pseudo Code

function parse\_arguments(argv):

for each argument in argv:

if argument is "-ip": next\_arg = IP

if argument is "-p": next\_arg = Port

if argument is "-f": next\_arg = Filename

if argument is "-key": next\_arg = Keyword

if any parameter missing:

print\_error("Missing arguments")

exit\_program()

### **validate\_arguments:**

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| ip | char\*\* | Pointer to store IP address |
| port | char\*\* | Pointer to store port number |
| filename | char\*\* | Pointer to store filename |
| keyword | char\*\* | Pointer to store keyword |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Exits program if any argument is missing |

#### Pseudo Code

function validate\_arguments(ip, port, filename, keyword):

if not valid\_ip(ip):

print\_error("Invalid IP format")

if not valid\_port(port):

print\_error("Invalid port number")

if not valid\_file(filename):

print\_error("File error")

if not valid\_keyword(keyword):

print\_error("Invalid keyword")

### **is\_valid\_ip:**

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| ip | const char\* | IP address to validate |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| 1 (true) | IP address is valid |
| 0 (false) | IP address is invalid |

#### Pseudo Code

function valid\_ip(ip):

return ip matches IPv4 pattern

### is\_valid\_port:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| port | const char\* | Port number to validate |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| 1 (true) | Port number is valid |
| 0 (false) | Port number is invalid |

#### Pseudo Code

function valid\_port(port):

convert port to number

return 1 <= port <= 65535 and no leading zeros

### is\_valid\_file:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| filename | const char\* | Filename to validate |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| 1 (true) | File exists and is non-empty |
| 0 (false) | File does not exist or is empty |

#### Pseudo Code

function valid\_file(filename):

if file exists and not empty:

return true

else:

return false

### is\_valid\_keyword:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| keyword | const char\* | Keyword to validate |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| 1 (true) | Keyword is valid (no digits) |
| 0 (false) | Keyword contains digits |

#### Pseudo Code

function valid\_keyword(keyword):

if keyword not empty and contains no digits

return true

else:

return false

### get\_file\_size:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| file | FILE\* | File pointer |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| long | Size of the file in bytes |

#### Pseudo Code

function get\_file\_size(file):

move to file end

size = current position

reset to file start

return size

### read\_file\_content:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| file | FILE\* | File pointer |
| file\_size | long | Size of the file in bytes |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| char\* | Pointer to the file content in memory |

#### Pseudo Code

function read\_file\_content(file, size):

allocate memory buffer of size+1

read entire file into buffer

add null terminator

return buffer

1. create\_client\_fd:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| None | - | - |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| int | Socket file descriptor |

#### Pseudo Code

function create\_socket():

create TCP socket

return socket descriptor

### connect\_server:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| port | char\* | Port number |
| ip | char\* | IP address |
| client\_fd | int | Client socket file descriptor |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Exits program if connection fails |

#### Pseudo Code

function connect\_to\_server(socket, ip, port):

create address structure with ip/port

attempt connection

if failed:

print\_error("Connection failed")

### send\_message\_to\_server:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| client\_fd | int | Client socket file descriptor |
| message | const char\* | Message to send |
| size | long | Size of the message in bytes |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Exits program if connection fails |

#### Pseudo Code

function send\_message\_to\_server(client\_socket, message, size):

total\_sent = 0 // Track the total bytes sent so far

while total\_sent < size: // Loop until the entire message is sent

// Send a chunk of the message starting from the current position

sent = send(client\_socket, message + total\_sent, size - total\_sent, 0)

if sent == -1: // Check for send error

print\_error("Failed to send message")

close\_socket(client\_socket)

exit\_program()

total\_sent += sent // Update the total bytes sent

### receive\_server:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| client\_fd | int | Client socket file descriptor |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Prints server response or handles errors |

#### Pseudo Code

function receive\_server\_response(client\_socket):

buffer = Allocate BUFFER\_SIZE bytes

bytes\_received = 0

done\_receiving = False

print("Encrypted message received from the server:")

while done\_receiving = False:

bytes\_received = recv(client\_socket, buffer, BUFFER\_SIZE - 1, 0)

if bytes\_received > 0:

buffer[bytes\_received] = '\0'

print(buffer)

if bytes\_received < BUFFER\_SIZE - 1:

done\_receiving = True

else if bytes\_received = 0:

done\_receiving = True

print("\nServer closed the connection.")

else:

print\_error("ERR: Receiving error")

done\_receiving = True

print("\nDisconnected from the server.")

close\_socket:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| client\_fd | int | Client socket file descriptor |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Ensures socket is closed properly |

#### Pseudo Code

function close\_socket(client\_socket):

close(client\_socket)

## Server

### validate\_argument\_number:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| argc | int | Number of command-line arguments |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Exits program if argument count is invalid |

#### Pseudo Code

function validate\_argument\_number(argc):

if argc != 5: // Expecting 4 arguments + program name

print\_error("Usage: -ip <IP Address> -p <Port>")

exit\_program()

### parse\_arguments:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| argc | int | Number of command-line arguments |
| argv | char\*\* | Array of command-line arguments |
| ip | char\*\* | Pointer to store IP address |
| port | char\*\* | Pointer to store port number |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Exits program if any argument is missing |

#### Pseudo Code

function parse\_arguments(argc, argv, ip, port):

for each argument in argv:

if argument is "-ip": next\_arg = IP

if argument is "-p": next\_arg = Port

if IP or Port is missing:

print\_error("Missing arguments")

exit\_program()

### validate\_arguments:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| ip | char\*\* | Pointer to IP address |
| port | char\*\* | Pointer to port number |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Exits program if any argument is invalid |

#### Pseudo Code

function validate\_arguments(ip, port):

if not valid\_ip(ip):

print\_error("Invalid IP format")

if not valid\_port(port):

print\_error("Invalid port number")

### is\_valid\_ip:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| ip | const char\* | IP address to validate |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| 1 (true) | IP address is valid |
| 0 (false) | IP address is invalid |

#### Pseudo Code

function is\_valid\_ip(ip):

return ip matches IPv4 pattern

### is\_valid\_port:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| port | const char\* | Port number to validate |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| 1 (true) | Port number is valid |
| 0 (false) | Port number is invalid |

#### Pseudo Code

function is\_valid\_port(port):

convert port to number

return 1 <= port <= 65535 and no leading zeros

### handle\_signal:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| signal | int | Signal number (e.g., SIGINT) |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Exits program after cleanup |

#### Pseudo Code

function handle\_signal(signal):

if signal is SIGINT:

print("Shutting down...")

cleanup()

exit\_program()

### create\_server\_fd:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| None | - | - |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| int | Socket file descriptor |

#### Pseudo Code

function create\_server\_fd():

create TCP socket

if socket creation fails:

print\_error("Socket creation failed")

exit\_program()

return socket descriptor

### config\_server:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| ip | const char\* | IP address |
| port | const char\* | Port number |
| server\_fd | int | Server socket file descriptor |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Exits program if configuration fails |

#### Pseudo Code

function config\_server(ip, port, server\_fd):

create address structure with ip/port

set socket options (reuse address)

bind socket to address

if bind fails:

print\_error("Binding failed")

exit\_program()

listen for incoming connections

if listen fails:

print\_error("Listen error")

exit\_program()

### accept\_client\_connections:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| server\_socket | int | Server socket file descriptor |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Runs indefinitely until shutdown |

#### Pseudo Code

function accept\_client\_connections(server\_socket):

while true:

print("Waiting for a client...")

client\_socket = accept(server\_socket)

if client\_socket is invalid:

print\_error("Accept failed")

continue

print("Client connected.")

process\_client\_message(client\_socket)

close(client\_socket)

print("Client disconnected.")

### process\_client\_message:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| client\_socket | int | Client socket file descriptor |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Handles client communication |

#### Pseudo Code

function process\_client\_message(client\_socket):

buffer = empty

keyword = empty

keyword\_received = false

while not keyword\_received:

read data into buffer

if newline found:

extract keyword

extract message

keyword\_received = true

encrypt message using Vigenère cipher

send encrypted message back to client

free allocated memory

### vigenere\_cipher:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| text | char\* | Text to encrypt |
| key | const char\* | Encryption key |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Modifies the input text in-place |

#### Pseudo Code

function vigenere\_cipher(text, key):

normalize key to uppercase

for each character in text:

if character is alphabetic:

shift character using key

free normalized key

### cleanup:

#### Parameters

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Type** | **Description** |
| None | - | - |

#### Return

|  |  |
| --- | --- |
| **Value** | **Reason** |
| None | Ensures resources are released |

#### Pseudo Code

function cleanup():

if server\_fd is valid:

close(server\_fd)

print("Server socket closed.")