COMP 7005
Computer Networks & Protocols

Assignment-02

Design

Anmol Mittal
A01397754
February 9th, 2025
Course Reference Number (CRN): 92522

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

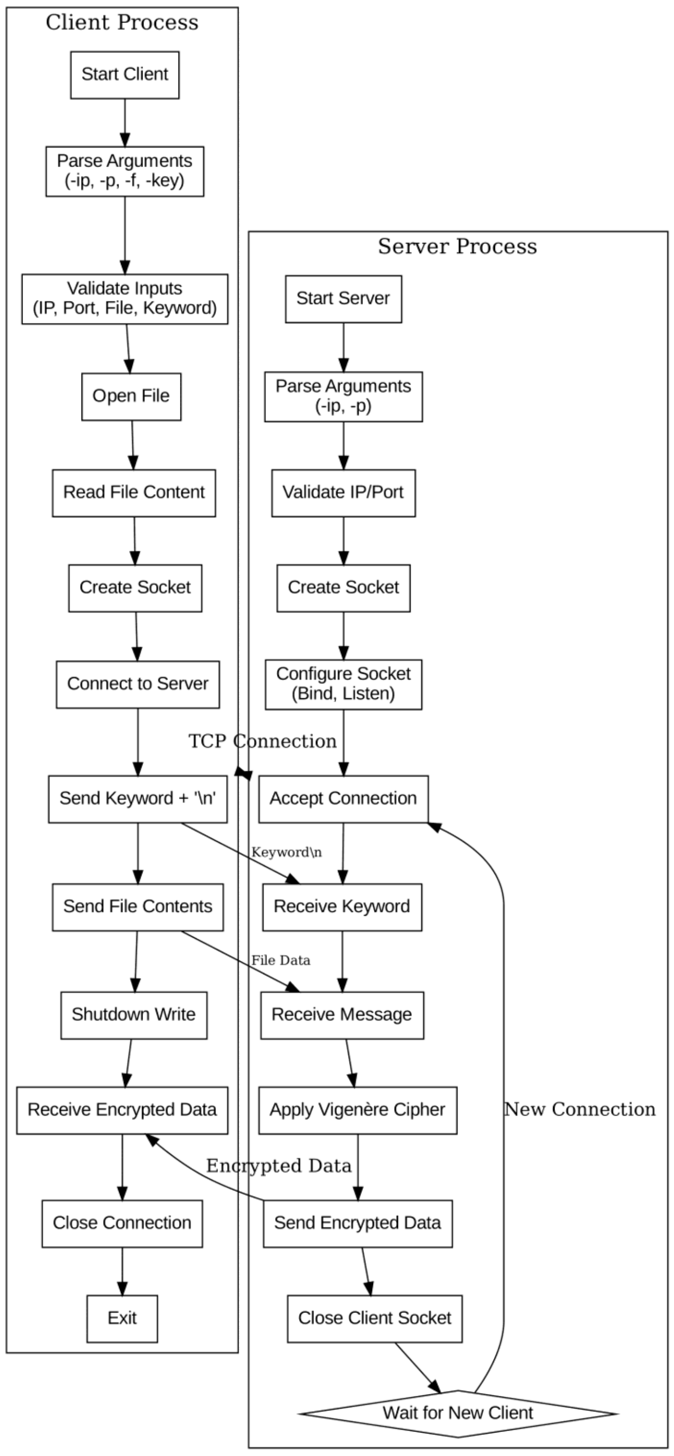
From State	To State	Function
START	VALIDATE	validate_arguments
VALIDATE	OPEN FILE	prepare_filename (parsing filename)
OPEN FILE	READ	open_file (open the file)
READ	INIT_SOCKET	create_client_socket
INIT_SOCKET	CONNECT	connect_to_server
CONNECT	SEND	send_message_to_server
SEND	RECEIVE RESPONSE	receive_server_response

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

1. 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()
```

2. 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()
```

3. 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")
```

4. 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
```

5. 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
```

6. 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
```

7. 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
```

8. 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
```

9. 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
```

10. 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
```

11. 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")
```

12. 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
```

13. 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

1. 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()
```

2. 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()
```

3. 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")
```

4. 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
```

5. 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
```

6. 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()
```

7. 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
```

8. 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()
```

9. 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.")
```

10. 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
```

11. 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
```

12. 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.")
```