COMP 7005
Assignment 1

Design

Daryush Balsara
A01265967
Sept, 20 2024

Purpose

- To implement a client server app using unix sockets. The client sends files the server receives the files and prints out the file name and sends back the contents of the file which gets printed out on the client.

Functions

client.py

parse_args	This is so that i can pass arguments to the program through the command line
connect_to_server	This creates a unix domain socket and connects to the server. If the connection fails, it prints an error message and exits the program.
send_file_request	This encodes a file request then sends it to the socket. If it doesn't work it prints an error
get_server_response	This reads the data on the server from the socket. The data that it gets is saved as a string. If there is an error then print it out
main	This is the way the program will be ran and where the functions will be called.

server.py

check_socket_path	This checks to see if there already is a socket file that exists. If there is then if trys to remove the socket if it can’t or it is not a socket file it prints a error
setup_server_socket	This creates a unix domain socket and tries to bind it to a socket file. It then listens to incoming connections from 1 host. If it is successful it prints a

	message and returns the socket if not it prints error and exits
wait_for_connection	This waits for the client to connect on the server. It then accepts the connection and returns the connection of the client if there is an error then print error
process_request	This handles a file request by checking if it exists it reads permission then returns the file contents or an error
send_reply	Sends response back to the client through the given

Variables

SOCKET_PATH	Path to were the socket is stored
LINE_LEN	Max number of bytes to read from socket at once

Pseudo Code

client.py

Parameters

Parameter	Type	Description
na	na	na

Return

Value	Reason
parse_args	Namespace containing the parsed command-line arguments, allowing the rest of the program to access the file paths provided by the user

parse_args

```
FUNCTION parse_args():
    CREATE argument parser with description "Unix domain socket assignment 1"
    ADD argument for file paths (one or more)
    RETURN parsed arguments
```

Parameters

Parameter	Type	Description
na	na	na

Return

Value	Reason
sock	Connected socket object after successfully establishing a connection to the server.

connect_to_server

```
FUNCTION connect_to_server():
    TRY:
        CREATE a Unix domain socket
        CONNECT the socket to SOCKET_PATH
        RETURN the socket
    CATCH exception AS e:
        PRINT "Error: Unable to connect to server socket: {e}"
        EXIT program with status 1
```

Parameters

Parameter	Type	Description
sock	Socket object	The socket through which the request will be sent.
file_request	string	The file path requested by the user

Return

Value	Reason
na	na

send_file_request

```
FUNCTION send_file_request(sock, file_request):
    TRY:
        SEND file_request encoded as UTF-8 through the socket
    CATCH exception AS e:
        PRINT "Error: Unable to send request: {e}"
        CLOSE the socket
        EXIT program with status 1
```

Parameters

Parameter	Type	Description
sock	Socket object	The socket from which the response will be received.

Return

Value	Reason
data	Returns the received data from the server as a string.

get_server_response

```
FUNCTION get_server_response(sock):
    TRY:
        SET data to an empty string
        WHILE true:
            RECEIVE part of data from the socket and decode it as UTF-8
            IF part is empty:
                BREAK
            APPEND part to data

        RETURN data
    CATCH exception AS e:
        PRINT "Error: Unable to receive response: {e}"
        RETURN None
```

main

```
FUNCTION main():
    CALL parse_args() to get arguments
    SET file_requests to the file_paths attribute from arguments

    FOR each file_request IN file_requests:
        CALL connect_to_server() to get the socket
        CALL send_file_request(sock, file_request stripped of whitespace)

        SET data to the result of get_server_response(sock)
        IF data is not None:
            PRINT "Contents in {file_request}:\n{data}"
        CLOSE the socket
```

server.py

Parameters

Parameter	Type	Description
na	na	na

Return

Value	Reason
na	na

check_socket_path

```
FUNCTION check_socket_path():
    IF SOCKET_PATH exists:
        TRY:
            IF SOCKET_PATH is a socket file:
                REMOVE SOCKET_PATH
                PRINT "Removed existing socket file: {SOCKET_PATH}"
            ELSE:
                PRINT "Warning: '{SOCKET_PATH}' exists but is not a socket file."
        CATCH exception AS e:
            PRINT "Error checking socket file: {e}"
    ELSE:
        PRINT "No existing socket file to remove at: {SOCKET_PATH}"
```

Parameters

Parameter	Type	Description
na	na	na

Return

Value	Reason
server_sock	The created and bound server socket so that it can be used later in the program to accept connections and handle incoming requests.

socket _server_socket

```
FUNCTION setup_server_socket():
    TRY:
        CREATE a Unix domain socket (server_sock)
        BIND server_sock to SOCKET_PATH
        LISTEN for incoming connections on server_sock
        PRINT "Server is listening on {SOCKET_PATH}"
        RETURN server_sock
    CATCH exception AS e:
        PRINT "Error: Unable to create server socket: {e}"
        EXIT program with status 1
```

Parameters

Parameter	Type	Description
server_sock	na	Required to accept a connection from clients. The function uses the passed server socket to listen for incoming connections

Return

Value	Reason
conn	Connection object if a client successfully connects to the server.

wait_for_connection

```
FUNCTION wait_for_connection(server_sock):
    TRY:
        ACCEPT a connection from server_sock and store in conn
        RETURN conn
    CATCH exception AS e:
        PRINT "Error: Unable to accept connection: {e}"
        RETURN None
```

Parameters

Parameter	Type	Description
server_sock	string	Necessary to identify which file the client is requesting.

Return

Value	Reason
A string	Returns file requests and needs to return either the requested data or an error message to inform the client of the result of their request.

process_request

```
FUNCTION process_request(requested_file):
    TRY:
        IF requested_file is a valid file:
            IF requested_file is not readable:
                RETURN "Error: Permission denied for file '{requested_file}'"
            OPEN requested_file in read mode with UTF-8 encoding AS file:
                RETURN contents of file
        ELSE:
            RETURN "Error: File '{requested_file}' not found."
    CATCH exception AS e:
        RETURN "Error: Unable to read file '{requested_file}': {e}"
```

Parameters

Parameter	Type	Description

conn	Socket object	Needed to specify the connection through which the reply will be sent back to the client,
reply	string	Maintains the message to be sent, whether it's the file content or an error message

Return

Value	**Reason**
na	na

send_reply

```
FUNCTION send_reply(conn, reply):
    TRY:
        SEND reply encoded as UTF-8 to conn
    CATCH BrokenPipeError:
        PRINT "Error: Client disconnected unexpectedly."
    CATCH exception AS e:
        PRINT "Error: Unable to send response: {e}"
```

main

```
FUNCTION main():
    CALL setup_server_socket() to create server_sock
    PRINT "Ctrl+C to shutdown and exit"

    TRY:
        WHILE true:
            CALL wait_for_connection(server_sock) to get conn
            IF conn is None:
                CONTINUE

            TRY:
                RECEIVE file request from conn and decode as UTF-8
                IF file request is empty:
                    BREAK

                PRINT "Received file request: {file_req}"
                CALL process_request(file_req) to get reply
                CALL send_reply(conn, reply)
```

```
            CATCH exception AS e:
                PRINT "Error handling request: {e}"
            FINALLY:
                CLOSE conn

    CATCH KeyboardInterrupt:
        PRINT "\nShutting down and closing the connection"
    FINALLY:
        CLOSE server_sock
        IF SOCKET_PATH exists:
            REMOVE SOCKET_PATH
        PRINT "Server socket removed"

IF this script is run as the main program:
    CALL main()
```