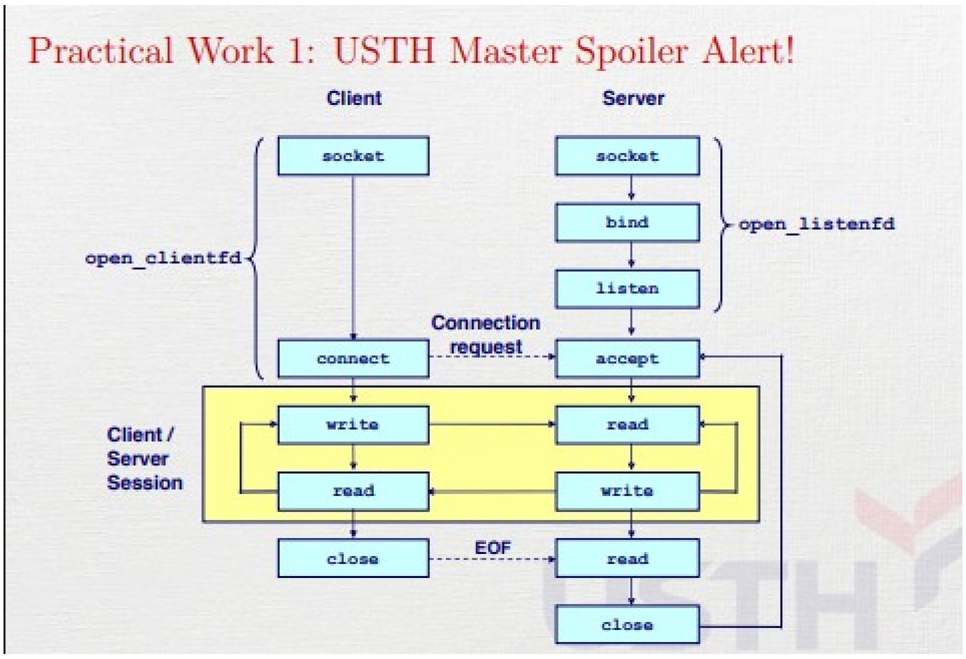
Distributed System

Labwork 1

Practical Work 1: USTH Master Spoiler Alert!



• socket(): Creates a socket, a communications endpoint
• setsockopt(): Set options on a socket
• bind(): Associate a socket with an address
• gethostbyname(): Get the the address of the machine with a given name
• listen(): Listen for machines trying to connect to this machine
• connect(): Establish a connection with another machine
• accept(): Accept a connection
• send(): Send data over a connection
• recv(): Read data from a connection

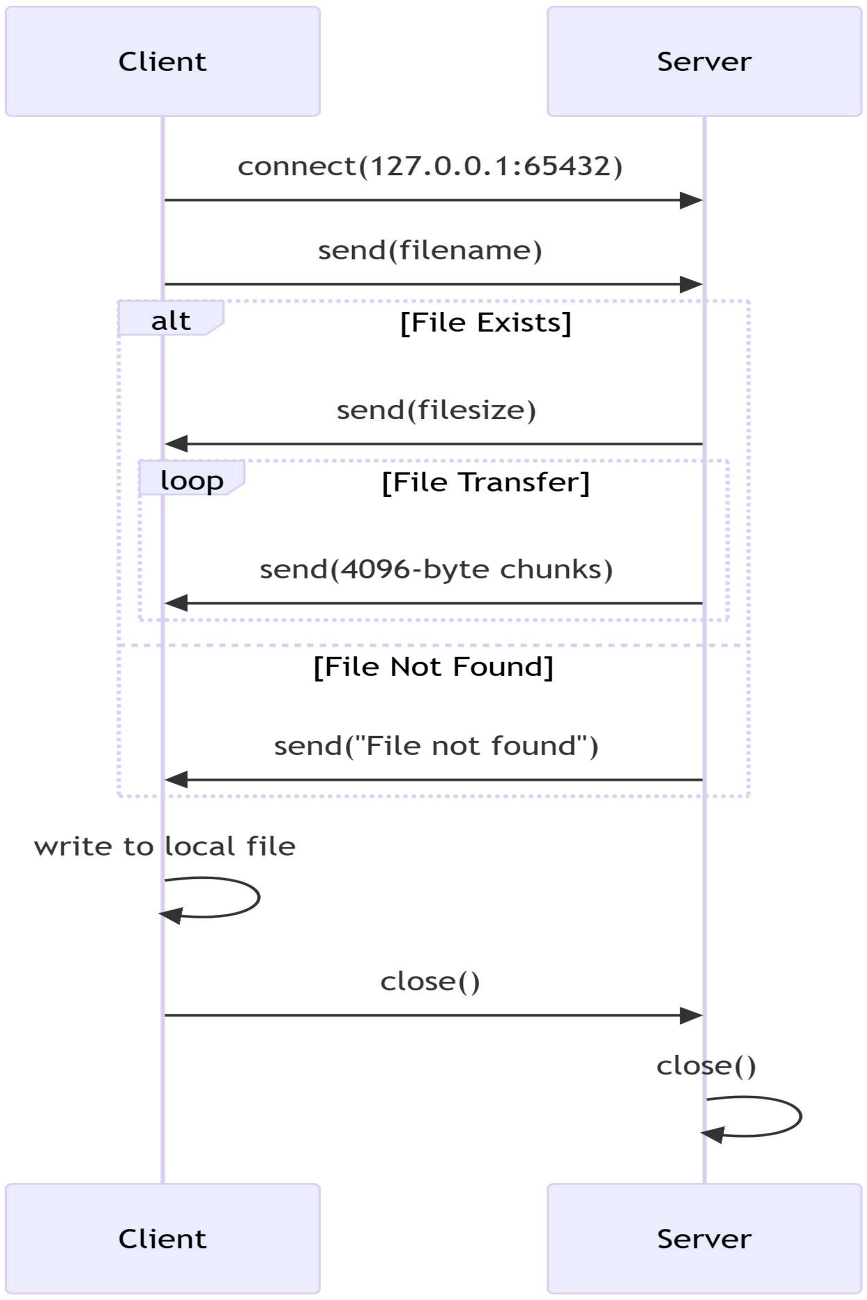
I. Protocol Design :

Simple and reliable request-response protocol over TCP:

1.1 Protocol Flow:

1.Client connects to server at 127.0.0.1:65432
2.Client sends requested filename as string
3Server checks file existence:
• If not exists → sends "FILE_NOT_FOUND"
• If exists → sends file size, then file content in 4096-byte chunks
4.Client receives and writes data to local file
5.Connection closes automatically after transfer completion

- 1.2 Protocol diagram:



II.System Organization

2.1 System Structure Explanation:

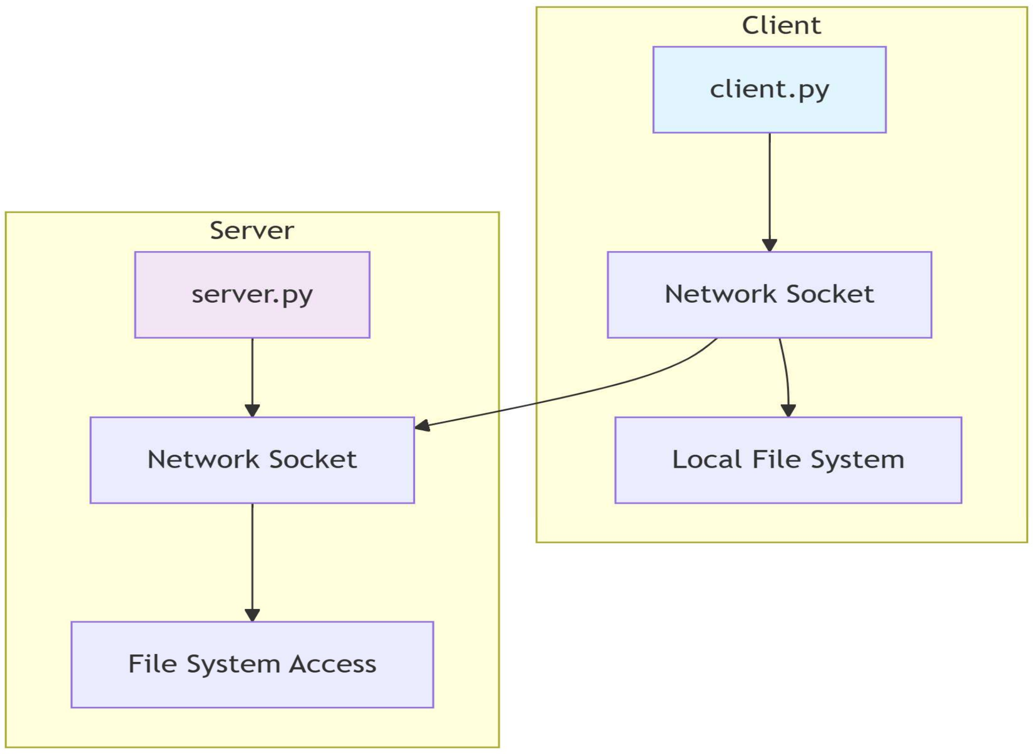
•**Client Program (client.c)**: Active component that initiates connection and requests files

•**Server Program (server.c)**: Passive component that listens for connections and serves files

•**Network Socket**: Communication endpoint using TCP/IP

•**File System**: Local storage for both original and received files

2.2 System diagram:



III.**File Transfer Implementation**

3.1 Server Implementation (server.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <sys/stat.h>

#define PORT 65432
#define BUFFER_SIZE 4096
#define FILENAME_SIZE 256

void send_file(int client_socket, const char *filename) {
    FILE *file = fopen(filename, "rb");
    if (file == NULL) {
        send(client_socket, "FILE_NOT_FOUND", 14, 0);
        return;
    }

    // Get file size
    fseek(file, 0, SEEK_END);
    long file_size = ftell(file);
    fseek(file, 0, SEEK_SET);

    // Send file size
    char size_buffer[32];
    snprintf(size_buffer, sizeof(size_buffer), "%ld", file_size);
    send(client_socket, size_buffer, strlen(size_buffer), 0);

    // Send file content
    char buffer[BUFFER_SIZE];
    size_t bytes_read;
    while ((bytes_read = fread(buffer, 1, BUFFER_SIZE, file)) > 0) {
        send(client_socket, buffer, bytes_read, 0);
    }

    fclose(file);
    printf("File sent successfully: %s (%ld bytes)\n", filename, file_size);
}

int main() {
    int server_fd, client_socket;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);

    // Create socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0) {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Set socket options
    if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof(opt))) {
        perror("setsockopt");
        exit(EXIT_FAILURE);
    }

    address.sin_family = AF_INET;
    address.sin_addr.s_addr = INADDR_ANY;
    address.sin_port = htons(PORT);

    // Bind the socket to the network address and port
    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) < 0) {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    // Listen for incoming connections
    if (listen(server_fd, 3) < 0) {
        perror("listen");
        exit(EXIT_FAILURE);
    }

    printf("Server listening on port %d\n", PORT);

    while (1) {
        // Accept incoming connection
        if ((client_socket = accept(server_fd, (struct sockaddr *)&address,
                                    (socklen_t*)&addrlen)) < 0) {
            perror("accept");
            exit(EXIT_FAILURE);
        }

        printf("Client connected\n");

        // Receive filename from client
        char filename[FILENAME_SIZE];
        int bytes_received = recv(client_socket, filename, FILENAME_SIZE - 1, 0);
        if (bytes_received > 0) {
            filename[bytes_received] = '\0';
            printf("Client requested: %s\n", filename);
            send_file(client_socket, filename);
        }

        close(client_socket);
        printf("Client disconnected\n\n");
    }

    close(server_fd);
    return 0;
}
```

3.2Client Implementation (client.c)

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define PORT 65432
#define BUFFER_SIZE 4096
#define FILENAME_SIZE 256
```

```
void receive_file(int server_socket, const char *filename, const char *output_filename) {
    char buffer[BUFFER_SIZE];
    int bytes_received;

    // Receive initial response (file size or error)
    bytes_received = recv(server_socket, buffer, BUFFER_SIZE - 1, 0);
    if (bytes_received <= 0) {
        printf("Server disconnected\n");
        return;
    }
    buffer[bytes_received] = '\0';

    // Check if file was found
    if (strcmp(buffer, "FILE_NOT_FOUND") == 0) {
        printf("Server: File not found!\n");
        return;
    }

    // Parse file size and receive file content
    long file_size = atol(buffer);
    printf("Receiving %s (%ld bytes)\n", filename, file_size);

    FILE *file = fopen(output_filename, "wb");
    if (file == NULL) {
        printf("Error creating output file\n");
        return;
    }

    long total_received = 0;
    while (total_received < file_size) {
        bytes_received = recv(server_socket, buffer, BUFFER_SIZE, 0);
        if (bytes_received <= 0) {
            break;
        }
        fwrite(buffer, 1, bytes_received, file);
        total_received += bytes_received;

        // Display progress
        float progress = (float)total_received / file_size * 100;
        printf("Progress: %.1f%%\r", progress);
        fflush(stdout);
    }
    fclose(file);
    printf("\nFile saved as: %s\n", output_filename);
}

int main(int argc, char *argv[]) {
    if (argc != 3) {
        printf("Usage: %s <filename> <output_filename>\n", argv[0]);
        return 1;
    }

    int sock = 0;
    struct sockaddr_in serv_addr;

    // Create socket
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        printf("Socket creation error\n");
        return -1;
    }

    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IP address to binary form
    if (inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr) <= 0) {
        printf("Invalid address\n");
        return -1;
    }

    // Connect to server
    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0) {
        printf("Connection failed\n");
        return -1;
    }

    printf("Connected to server\n");

    // Send filename to server
    send(sock, argv[1], strlen(argv[1]), 0);

    // Receive file from server
    receive_file(sock, argv[1], argv[2]);

    close(sock);
    return 0;
```

```
}
```

IV. Role of Each Component :

4.1 Server:

– Creates and binds the socket to port 65432
– Listens and accepts client connection
– Receives filename request
– Sends file size + content in 4KB chunks
– Handles "file not found" gracefully

4.2 Client:

– Connects to server
– Sends the desired filename
– Receives file size and data
– Writes to a local file with progress tracking
– Supports command-line arguments

V. How to run :

5.1 Compile the programs:

```
gcc -o server server.c
gcc -o client client.c
```

5.2 Create a test file:

```
echo "Hello, this is a test file" > example.txt
```

5.3 Start the server:

```
./server
```

5.4 Run the client (in new terminal):

```
./client example.txt received.txt
```

5.5 Verify the file:

```
cat received.txt
```