CSC4303 Assignment-2: Socket Programming Tutorial

> This tutorial will guide you step by step through the Socket programming assignment

📋 Assignment Overview

Part	Content	Points
Part 1	Basic Socket Communication	6 pts
Part 2	Simple File Transfer	3 pts
Part 3	Large File Transfer (Bonus)	1 pt

Deadline: February 6, 2026 23:59

🛠️ Environment Setup

System Requirements

- **macOS**: Use the terminal directly
- **Linux**: Use the terminal directly
- **Windows**: Must install WSL2

```
# Run in PowerShell (Administrator)
wsl --install -d Ubuntu-22.04
```

Install Dependencies

```
# Ubuntu/WSL
sudo apt update
sudo apt install build-essential cmake

# macOS
xcode-select --install
brew install cmake
```

Build the Project

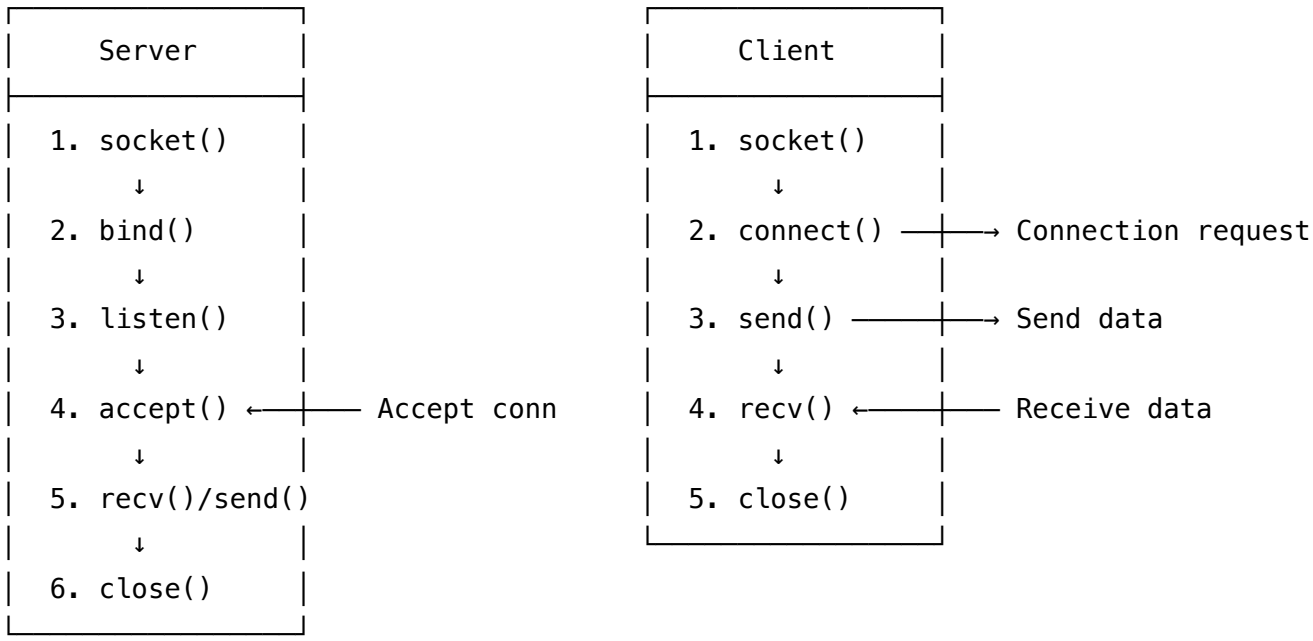
```
mkdir build
cd build
cmake ..
make
```

🌐 Part 1: Basic Socket Programming

1.1 What is a Socket?

A socket is an endpoint for network communication. Think of it as a "network outlet." Two programs establish a connection through their respective sockets, like connecting two phones with a telephone line.

1.2 TCP Communication Flow



1.3 Key Function Descriptions

Function	Purpose	Return Value
`socket()`	Create a socket	Success: fd, Failure: -1
`bind()`	Bind address and port	Success: 0, Failure: -1
`listen()`	Start listening	Success: 0, Failure: -1
`accept()`	Accept connection	Success: new fd, Failure: -1
`connect()`	Connect to server	Success: 0, Failure: -1
`send()/recv()`	Send/Receive data	Returns bytes, Failure: -1

1.4 Implementing socket_client.c (5 TODOs)

Open `socket_client.c` and complete the following steps:

TODO 1: Create Socket

```
// Hint: Use socket(AF_INET, SOCK_STREAM, 0)
// AF_INET = IPv4, SOCK_STREAM = TCP
// Check return value, use perror() on failure
```

TODO 2: Convert IP Address

```
// Hint: Use inet_pton(AF_INET, "127.0.0.1", &serv_addr.sin_addr)
// Converts string IP to binary format
// Return value <= 0 indicates failure
```

TODO 3: Connect to Server

```
// Hint: Use connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr))
// Check return value
```

TODO 4: Read Server Response

```
// Hint: Use read(sock, buffer, BUFFER_SIZE)
```

TODO 5: Close Connection

```
// Hint: Use close(sock)
```

1.5 Implementing socket_server.c (7 TODOs)

TODO 1-4: Basic Socket Setup

Similar to client, you need to:

- Create socket
- Set `SO_REUSEADDR` option (avoid "Address already in use" error)
- Bind address with `bind()`
- Start listening with `listen()`, set backlog to 3

TODO 5: Prepare poll_fds Array ⭐Key Point

`poll()` is an I/O multiplexing technique that allows monitoring multiple connections simultaneously:

```
// Iterate through all client slots
for (int i = 0; i < max_clients; i++) {
    int sd = client_socket[i];
    // Set the file descriptor to monitor
    poll_fds[i + 1].fd = sd;
    // If slot is valid (sd > 0), monitor POLLIN event
    poll_fds[i + 1].events = (sd > 0) ? POLLIN : 0;
}
```

TODO 6: Accept New Connection

```
// Hint: Use accept(master_socket, ...)
```

TODO 7: Add New Client to Array

```
// Iterate through client_socket array, find empty slot (value 0)
// Store new_socket in that slot
```

1.6 Testing Part 1

```
# Terminal 1: Start server
cd build
./socket_server

# Terminal 2: Run client
./socket_client
```

Expected Output:

```
# Server side
Listener on port 8080
Waiting for connections ...
New connection, socket fd is 4, ip is: 127.0.0.1, port: xxxxx

# Client side
Hello message sent
Message from server: Hello from client
```

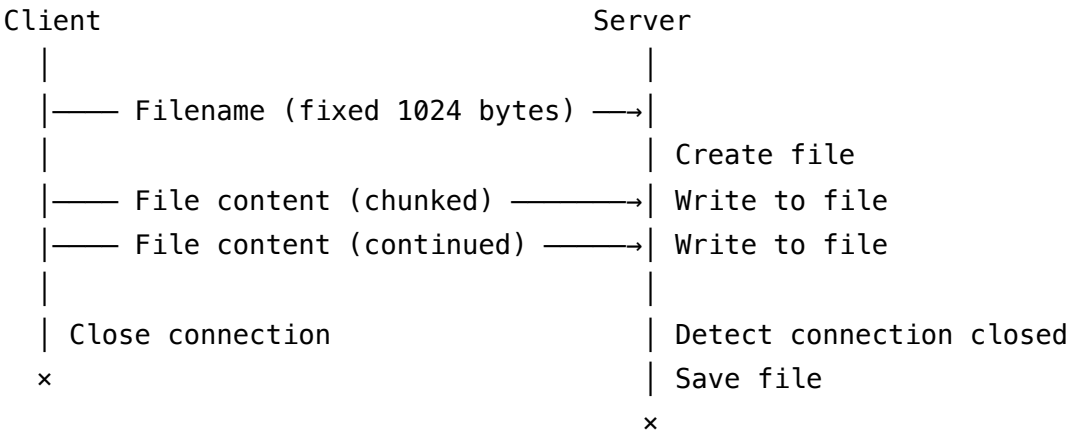
Concurrency Test

```
cp ../con.sh .
chmod +x con.sh
./con.sh
```

You should see all 30 clients successfully sending and receiving messages.

📁 Part 2: Simple File Transfer

2.1 File Transfer Protocol Design



2.2 Implementing file_client.c

The socket connection part is the same as `socket_client.c` , you can reuse the code.

Key Steps:

1. Open File

```
FILE *file = fopen(argv[1], "rb");  // "rb" = binary read
```

2. Extract Filename

```
// "../file1.zip" → "file1.zip"
char *base_filename = strrchr(argv[1], '/');
if (base_filename) {
    base_filename++;  // Skip '/'
} else {
    base_filename = argv[1];
}
```

3. Send Filename

```
memset(buffer, 0, BUFFER_SIZE);
strncpy(buffer, base_filename, BUFFER_SIZE - 1);
send(sock, buffer, BUFFER_SIZE, 0);  // Send fixed size
```

4. Send File Content

```
size_t bytes_read;
while ((bytes_read = fread(buffer, 1, BUFFER_SIZE, file)) > 0) {
    send(sock, buffer, bytes_read, 0);
}
```

5. Close Resources

```
fclose(file);
close(sock);
```

2.3 Implementing file_server.c

The socket setup part is the same as `socket_server.c` .

File Receiving Logic:

```
// 1. Receive filename
char filename[BUFFER_SIZE];
valread = read(sd, filename, BUFFER_SIZE);

// 2. Handle disconnection
if (valread == 0) {
    close(sd);
    client_socket[i] = 0;
    continue;
}

// 3. Create file
FILE *file = fopen(filename, "wb");  // "wb" = binary write

// 4. Receive and write file content
while ((valread = read(sd, buffer, BUFFER_SIZE)) > 0) {
    fwrite(buffer, 1, valread, file);
}

// 5. Cleanup
fclose(file);
close(sd);
client_socket[i] = 0;
```

2.4 Testing Part 2

```
# Generate test files
cd ..
chmod +x generator.sh
./generator.sh

# Start server
cd build
./file_server

# In another terminal, send file
./file_client ../file1.zip

# Verify file integrity
md5 ../file1.zip ./file1.zip      # macOS
md5sum ../file1.zip ./file1.zip   # Linux
```

Success Indicator: Both files have the same MD5 checksum.

🚀 Part 3: Large File Transfer (Bonus)

3.1 Problem Analysis

The current implementation may have issues with large files (>32MB). Consider:

- How does the server know when file transfer is complete?
- What mechanism does it currently rely on to determine this?

3.2 Hints

Consider one of the following approaches:

Approach A: Send file size first

```
// Client: Get and send file size
fseek(file, 0, SEEK_END);
long file_size = ftell(file);
fseek(file, 0, SEEK_SET);
// Send file_size...

// Server: Receive based on file size
// Loop reading until file_size bytes received
```

Approach B: Use `shutdown()`

```
// Client: Close write end after sending
shutdown(sock, SHUT_WR);
```

3.3 Testing

```
./file_client ../bigfile1.zip
md5 ../bigfile1.zip ./bigfile1.zip
```

❓ Common Issues

Error Message	Cause	Solution
"Address already in use"	Port is occupied	`pkill socket_server` or `pkill file_server`
"Connection refused"	Server not running	Start the server first
"Permission denied"	Script lacks execute permission	`chmod +x *.sh`
File MD5 mismatch	Incomplete transfer	Check read/write loop logic

📚 References

- Beej's Guide to Network Programming
- `man socket` , `man bind` , `man listen` , `man accept`
- `man poll` - I/O Multiplexing

✅ Checklist

Before submitting, confirm all these tests pass:

- [ ] `socket_client` can connect to `socket_server` and receive echo
- [ ] 30 concurrent clients test passes (`con.sh`)
- [ ] `file_client` can transfer files to `file_server`
- [ ] Transferred file MD5 matches original file
- [ ] (Bonus) 50MB large file transfer succeeds

Good luck! 🎓