CSCI 4061: Introduction to Operating Systems, Fall 2023
Project #4: Client-Server image processing

Instructor: Abhishek Chandra

Intermediate submission due: **11:59pm (CDT), Dec. 5, 2023**

Final submission due: **11:59pm (CDT), Dec. 12, 2023**

1. Background

Image rotation is the process of changing the orientation of an image by turning it around a central point. In the previous assignment, we have implemented a parallel image processing program using multi-threading. In this assignment, we will implement a client-server version of image processing using socket programming. The application will involve a server capable of handling multiple client connections concurrently and performing image processing tasks. Clients will send images to the server, specifying a rotation angle, and the server will process these images accordingly and send them back.

The purpose of this programming assignment is to get you started with socket programming. You will need to be familiar with TCP sockets and client-server communication.

2. Project Overview

Your project will be composed of two types of program: the **server** program and the **client** program. The server program will listen on a specific port for any incoming client connection. A client can connect to the server at any time. Once a client connection is established, a server thread will be spawned to handle all the requests from this client.

2.1 Server program

Server Functionality:

- Initialize a TCP Socket: Listen for incoming connections on a specific port, the **port number** should be the **last 4 digits** of the student who will be submitting this assignment for the group.
- Accept Client Connections: For each client, create a new **client handling thread** to handle the connection.

2.1.1 Client Handling thread

The server will listen for clients and spawn a client handling thread when a client connection is accepted. The client handling thread will handle all the image processing requests from a given client connection. The detailed functionality is shown below.

- Repeat following steps when receiving the IMG_OP_ROTATE operation message.
    - Receive request packet with Image Size, Data and Rotation Angle:
        - First, check the flags to determine the rotation. (180 or 270)
        - Then, receive the image file, storing it temporarily in memory.
    - Process the Image: Perform the requested rotation (e.g., flip left-to-right).
        - Send back the package with an IMG_OP_NAK message if an error happens. Skip the rest of the job for this image.
    - Send the Processed Image Back: Once the image is processed, send it back to the respective client following the server response packet protocol.
- Close connection when client terminates the session.
    - Close the Connection: After receiving IMG_OP_EXIT operation message from client, close the connection and clean up resources.

2.2 Client program

Client program will take an input directory as a command line argument.

- Connect to the Server: Establish a TCP connection to the server.

    - The port number should be the **last 4 digits** of the student who will be submitting this assignment for the group.
- Read all the files from the input directory and put them in a **queue**, along with rotation angle.
- While the **queue** is not empty.
    - Pop a file from the queue.
    - Send a packet with the IMG_FLAG_ROTATE_XXX message header desired rotation Angle, Image size, and data.

    - Receive and Save the Processed Image:
        - Receive the response packet containing the processed image from the server.
        - Save the image to a specified directory (e.g., 'output').
- When **queue** is empty
    - Send ‘terminate’ message through socket.
    - Close the Connection.

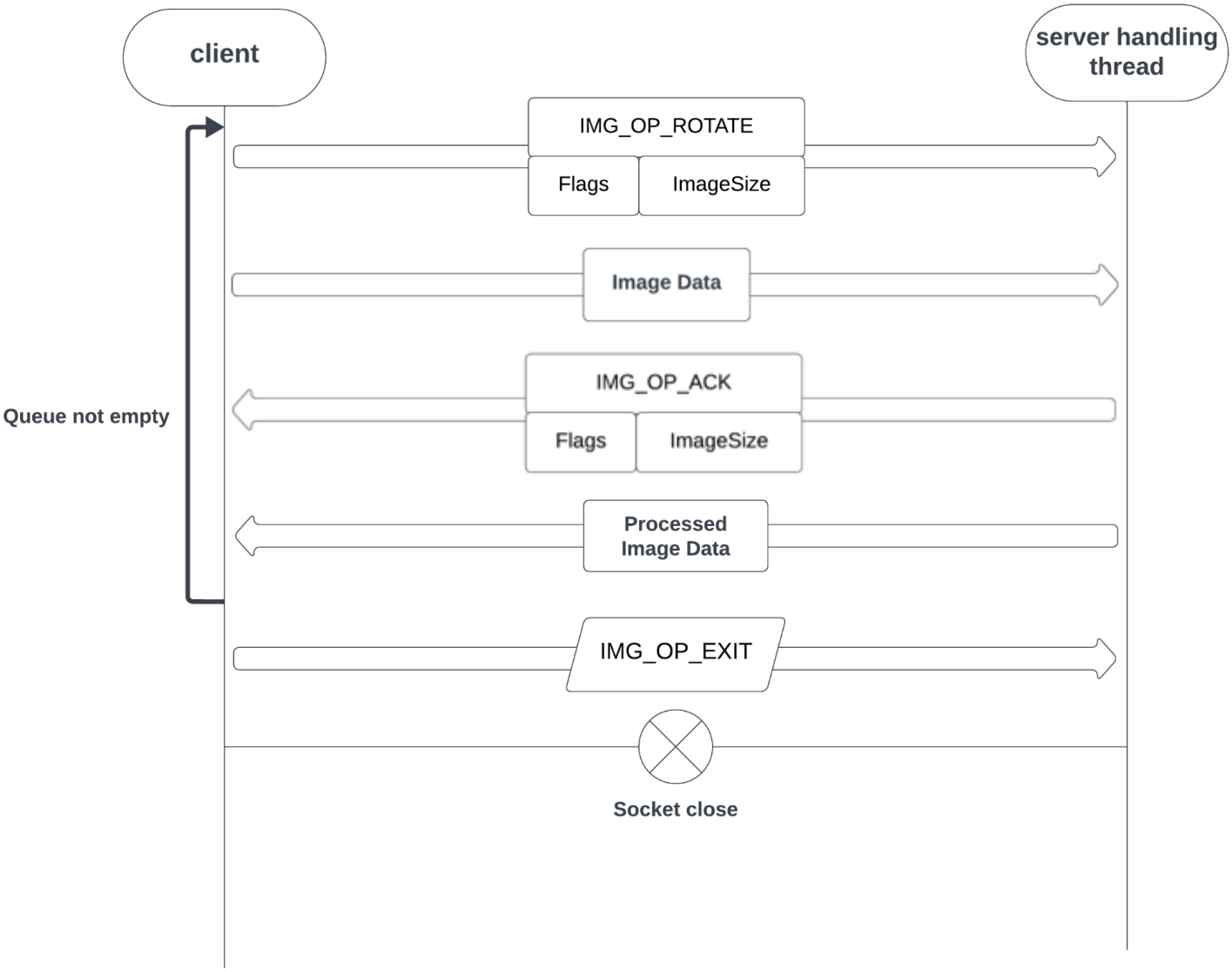

Figure 1. Communication process between the client and server handling thread.

2.3 Communication protocol

You will implement the following communication protocol between the server and the client to make your application work. The packet format is shown below and the different operations are described below.

PACKET_T		
Operation (int)	**Flags** (int)	**Image Size** (int)
Checksum (Char, 32 bytes, extra credit)		
Image Data		

Operation:

IMG_OP_ACK (1):

IMG_OP_NAK (2):

Server responds to the clients image processing request with negative acknowledgement based on the contents of the request.

IMG_OP_ROTATE (3):

Client requests to perform a rotation operation on an image.

IMG_OP_EXIT (4):

Client notifies the server to terminate the connection.

Flags:

IMG_FLAG_ROTATE_180 (1):

The processed image should be a 180 degree rotation,

IMG_FLAG_ROTATE_270 (2):

The processed image should be a 270 degree rotation,

IMG_FLAG_ENCRYPTED (3):

Indicates that the message is encrypted.

IMG_FLAG_CHECKSUM (4):

Indicates that the packet contains a checksum for the image. The checksum will be a 32 bytes Char digest.

Image Size:

The size of the image indicates how big the image file is.

Image Data:

The image is sent as a stream of bytes. The server reads the exact number of bytes as indicated by the image size.

Processed Image:

The server sends back the processed image using a similar format - with acknowledgement on success and non-acknowledgement otherwise. In negative acknowledgement cases, send back the package received with IMG_OP_NAK operation message.

3. Project structure

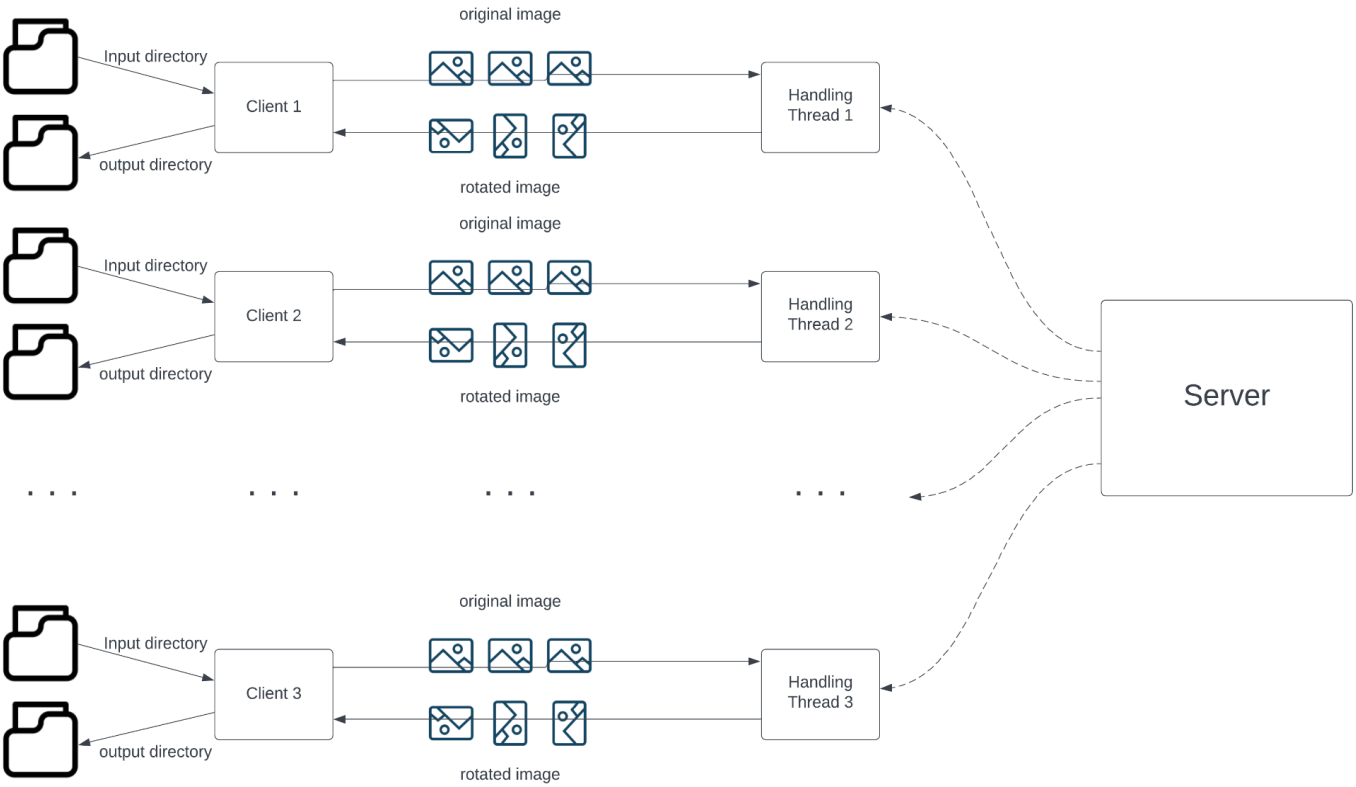

Figure 2. Structure of the project

4. Compilation Instructions

You can compile the **server** solely with

Command Line
`$ make server`

You can compile the **client** solely with

Command Line
`$ make client`

Or you can compile the **client** and the **server** with

Command Line
`$ make all`

You can run the **server** with

Command Line
`$ ./server`

You can run the **client** with

Command Line
`$ ./client <input_dir> <output_dir> <rotate_angle>`

You can clean out the **compiled server**, **compiled client** and reset **input/output directory** with

Command Line
`$ make clean`

5. Project folder structure

Please strictly conform to the folder structure that is provided to you. Your

conformance will be graded.

Directory	Content
include/	.h header files (utils.h, server.h, client.h)
lib/	.o library files (utils.o)
src/	.c source files (server.c, client.c)
img/	Contain multiple Directories that contain different images
Expected/	Expected output
Makefile	File containing build information and used for testing/compiling

6. Assumption / Notes

1. The Encrypt library, which is used for extra credit, is deprecated. For this assignment, you can assume the library is safe for practice purposes only. Please use an alternative encryption library in the future to avoid security issues.

7. Submission Details

There will be two submission periods. The intermediate submission is due 1 week before the final submission deadline. The first submission is mainly intended to make sure you are on pace to finish the project on time. The final submission is due ~2 weeks after the project is released.

7.1 Intermediate Submission

For the Intermediate submission, you are required to implement 1) directory traversal and add the directory content into a request queue. 2) Establishing TCP connection between Server and Client. 3)Send a package with IMG_OP_ROTATE to the server and receive the package on the server side.

One student from each group should upload a **.zip file** to Gradescope containing all of your project files. We'll be primarily focusing on *.c and your README, which should contain the following information:

- Project group number
- Group member names and x500s
- The name of the CSELabs computer that you tested your code on
    - e.g. csel-kh1250-01.cselabs.umn.edu
- Any changes you made to the Makefile or existing files that would affect grading
- Plan outlining individual contributions for each member of your group
- Plan on how you are going to construct the client handling thread and how you plan to send the package according to the protocol.

The member of the group who uploads the .zip file to Gradescope should add the other members to their group after submitting. Only one member in a group should upload.

7.2. Final submission

One student from each group should upload a **.zip** file to Gradescope containing all of the project files. The README should include the following details:

- Project group number
- Group member names and x500s
- The name of the CSELabs computer that you tested your code on
    - e.g. csel-kh1250-01.cselabs.umn.edu
- Members' individual contributions

- Any changes you made to the Makefile or existing files that would affect grading
- Any assumptions that you made that weren't outlined in section 7
- How you designed your program for Package sending and processing (again, high-level pseudocode would be acceptable/preferred for this part)
- If your original design (intermediate submission) was different, explain how it changed
- Any code that you used AI helper tools to write with a clear justification and explanation of the code (Please see below for the AI tools acceptable use policy)
- How you implement error detection/correction and the package encryption (extra credit, optional)

The member of the group who uploads the .zip file to Gradescope should add the other members to their group after submitting. Only one member in a group should upload.

Your project folder should include all of the folders that were in the original template. You can add additional files to those folders and edit the Makefile, **but make sure everything still works**. Before submitting your final project, run "make clean" to remove any existing output/ data and manually remove any erroneous files.

8. Reiteration of AI Tool Policy

Artificial intelligence (AI) language models, such as ChatGPT, may be used in a **limited manner for programming assignments** with appropriate attribution and citation. For programming assignments, you may use AI tools to help with some basic helper function code (similar to library functions). You must not use these tools to design your solution, or to generate a significant part of your assignment code. You must design and implement the code yourself. You must clearly identify parts of the code that you used AI tools for, providing a justification for doing so. You must understand such code and be prepared to

explain its functionality. Note that the goal of these assignments is to provide you with a deeper and hands-on understanding of the concepts you learn in the class, and not merely to produce "something that works".

If you are in doubt as to whether you are using AI language models appropriately in this course, I encourage you to discuss your situation with me. Examples of citing AI language models are available at: libguides.umn.edu/chatgpt. You are responsible for fact checking statements and correctness of code composed by AI language models.

For this assignment: The use of AI tools for generating code related to the primary concepts being applied in the assignment, such as thread management and synchronization, socket management, I/O and communication, is prohibited.

9. Rubric (tentative)

- [10%] README
- [10%] Intermediate submission
- [20%] Coding style: indentations, readability, comments where appropriate
- [20%] Test cases
- [30%] Correct use of socket(), connect(), bind(), listen(), accept(), send(), recv().
- [10%] Error handling — should handle system call errors and terminate gracefully
- [10%] Extra credit - Message encryption and data checksum

Additional notes:

- We will use the GCC version installed on the CSELabs machines to compile your code. Make sure your code compiles and runs on CSELabs.
- A list of CSELabs machines can be found at https://cse.umn.edu/cseit/classrooms-labs
- Try to stick with the Keller Hall computers since those are what we'll use to test your code

- Helpful GDB manual. From Huang: GDB Tutorial From Kauffman: Quick Guide to gdb