COSC 3360 (16770) - COSC 6310 (11209) - Operating Systems

Available from: Thursday, 20 February 2025, 12:00 AM
Due date: Sunday, 30 March 2025, 11:59 PM
Requested files: client.cpp, server.cpp (Download)
Type of work: Individual work

This programming assignment closes at 11:58:59 PM on 03/30/2025.

Similarity Threshold: 90%

Objective:
This assignment will introduce you to interprocess communication mechanisms in UNIX using sockets.

Problem:
You must write two programs to implement a distributed version of the program to draw a 2D image with printable ASCII characters (based on the Encoding Method for Sparse Binary Data presented by Weiwei Duan et al.) you wrote for programming assignment 1.

These programs are:

The server program:

The user will execute this program using the following syntax:

./exec_filename port_no

where exec_filename is the name of your executable file, and port_no is the port number to create the socket. The port number will be available to the server program as a command-line argument.

The server program does not receive any information from STDIN and does not print any messages to STDOUT.

The server program executes the following task:

- Receive multiple requests from the client program using sockets. Therefore, the server program creates a child process per request to handle these requests simultaneously. For this reason, the parent process must handle zombie processes by implementing the fireman() function call (unless you can determine the number of requests the server program receives from the client program).

Each child process executes the following tasks:

1. Receive the image size, the output characters, the headPos array, the dataPos array, and the row number to decode.

2. Decode the assigned row using the Encoding Method for Sparse Binary Data.
3. Send the decoded row to the client program.

The client program:

The user will execute this program using the following syntax:

```
./exec_filename hostname port_no < input_filename
```

where exec_filename is the name of your executable file, hostname is the address where the server program is located, port_no is the port number used by the server program, and input_filename is the name of the input file. The hostname and the port number will be available to the client as command-line arguments.

The client program receives a multiline input representing the image to decode (based on programming assignment 1 specifications).

Example Input:

```
26 7
U 0 10,H 15 25
0 4 8 12 25 29 33
0 10 15 25 0 10 15 25 0 10 15 25 0 10 15 16 17 18 19 20 21 22 23 24 25 0 10 15 25 1 9 15 25 2 3 4 5 6
7 8 15 25
```

After reading the information from STDIN, the program must create n child threads (where n is the number of lines in the image (width)). Each child thread executes the following tasks:

1. Receives the image size, the output characters, the headPos array, the dataPos array, the row number to decode, and the memory location to store the decoding process results from the main thread.
2. Create a socket to communicate with the server program.
3. Send the required information to decode the assigned row to the server program using sockets.
4. Wait for the decoded row from the server program.
5. Write the received information into a memory location accessible by the main thread.

Finally, after receiving the decoded rows, the main thread prints this information to STDOUT. Given the previous input, the expected output is:

```
U         U    H         H
U         U    H         H
U         U    H         H
U         U    HHHHHHHHHHH
U         U    H         H
 U       U     H         H
  UUUUUUU      H         H
```

Notes:

- You can safely assume that the input will always be in the proper format.
- You must use the output statement format based on the example above.

    - For the client program, you must use POSIX Threads and stream sockets. A penalty of 100% will be applied to submissions not using POSIX Threads and Stream Sockets.
    - You must use multiple processes (fork) and stream sockets for the server program. Submissions that do not use multiple processes and Stream Sockets will be penalized 100%.
    - The Moodle server will kill your server program after executing each test case.
    - You must present code that is readable and has comments explaining the logic of your solution. A 10% penalty will be applied to submissions not following this guideline.
    - You cannot use global variables. A 100% penalty will be applied to submissions using global variables.
    - A penalty of 100% will be applied to solutions that do not compile.
    - A penalty of 100% will be applied to solutions hardcoding the output.

RUBRIC:

- 10 points for each test case (20 points total)
- 10 points for presenting clean and readable code.
- 10 points for using the fireman function correctly or using a for loop based on the number of requests.
- 10 points for reading information from STDIN (client).
- 25 points if each request to the server:
    - Receives information from the client program and creates a child process per request (5 points).
    - Uses the Encoding Method for Sparse Binary Data presented by Weiwei Duan et al. (10 points).
    - Return the decoded row to the client program (5 points).
    - Closes the socket (5 points).
- 25 points if each child thread:
    - Creates a socket to communicate with the server (5 points).
    - Sends and receives information to/from the server (10 points).
    - Stores the information on a memory location accessible by the main thread (5 points).
    - Closes the socket (5 points).

ADDITIONAL PENALTIES:

100 points off for a solution not compiling (in addition to test cases not passing)

100 points off for lack of generalized solution (hard coding, no server file, etc.)

100 points off for not using POSIX threads, fork(), or Sockets.

100 points for using global variables.

Requested files

client.cpp

```
// Write your code here
```

server.cpp

```
// Write your code here
```