COSC 3360 - Fundamentals of Operating Systems

≡ Description   ⊟ Submission view

Programming Assignment 2

🗓 **Due date**: Friday, 1 April 2022, 11:59 PM
🛡 **Requested files**: client.cpp, server.cpp (⬇ Download)
Type of work: 👤 Individual work

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 parallel fixed-length code decompressor you created for programming assignment 1.

These programs are:

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 file with a string representing the compressed message (sequence of bits). The hostname and the port number will be available to the client as command-line arguments.

This program receives the number of bits of the fixed-length codes from the server program (using sockets). Then, your solution creates m child threads (where m is the number of characters in the decompressed message). Each child thread will use the parallel solution from assignment 1 to store the decompressed message into a memory location accessible by the main thread.

Each child thread determines a character of the decompressed message following the steps presented below:

1. Create a socket to communicate with the server program.
2. Send the binary code of the symbol to decompress to the server program using sockets.
3. Wait for the decompressed character from the server program.
4. Write the received information into a memory location accessible by the main thread.
5. Finish its execution.

Input Format: Your program should read its input from STDIN (C++ cin) (Moodle uses input redirection to send the information from the input file to STDIN).

The input file has a string representing the compressed message (sequence of bits).

Example Input File:

```
010010010100101101101
```

Finally, after receiving the characters of the decompressed message from the child threads, the main thread prints the decompressed message. Given the previous input file, the expected output is:

```
Decompressed message: aaabccc
```

The server program:

The user will execute this program using the following syntax:

./exec_filename port_no < input_filename

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

The server program calculates the number of bits of the fixed-length codes based on the value of the greatest base 10 code in the alphabet. For decimal values greater than zero, you can use the following formula to calculate the number of bits of the fixed-length codes: ceiling(log_2(greatest_base_10_code_in_the_alphabet +1)). Additionally, this program determines the binary representation of each of the symbols in the alphabet (based on the decimal value from the input file).

This program receives two types of requests from the client program using sockets:

1. A request from the main thread of the client program asking for the number of bits of the fixed-length codes.
2. One request per child thread of the client program. These threads send the binary code of a symbol in the compressed message, and the server returns the character from the alphabet assigned to that binary sequence.

To handle multiple requests at the same time, the server program creates a child process per request. For this reason, the parent process needs to handle zombies processes implementing the fireman() call in the primer. Each child process will determine the type of request received from the client program, returning the expected value according to the type of request.

Input Format: Your program should read its input from STDIN (C++ cin) (Moodle uses input redirection to send the information from the input file to STDIN).

The input file has the information about the alphabet used to compress the message. The format of the input file is as follow:

- An integer value representing the number of symbols in the alphabet
- n lines (where n is the number of symbols in the alphabet) with a char representing the value of the symbol and an integer value representing the symbol's code (in decimal notation).

Example Input File:

```
3
a 2
b 4
c 5
```

The server program will not print any information to STDOUT.

Notes:

- **You can safely assume that the input files 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 socket. A penalty of 100% will be applied to submissions not using POSIX Threads and Stream Sockets.**
- **For the server program, you must use multiple processes (fork) and stream socket. A penalty of 100% will be applied to submissions not using multiple processes and Stream Sockets.**
- **The Moodle server will kill your server program after is done executing each test case.**

Requested files

client.cpp

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

server.cpp

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

Jump to...