COSC 3360 - 24967 - Fundamentals of Operating Systems

Dashboard / My courses / COSC3360SP2023-01 / PROGRAMMING ASSIGNMENTS / Programming Assignment 2

≣ Description   ⊟ Submission view

📅 **Available from**: Saturday, 25 February 2023, 12:00 PM
📅 **Due date**: Sunday, 2 April 2023, 11:59 PM
🛡 **Requested files**: client.cpp, server.cpp, huffmanTree.h, huffmanTree.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 multithreaded Huffman decompressor you created for programming assignment 1.

These programs are:

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 alphabet's information. The port number will be available to the server program as a command-line argument.

The server program receives from STDIN the alphabet's information (using input redirection). The input file has multiple lines, where each line contains information (character and frequency) about a symbol from the alphabet. The input file format is as follows:

- A char representing the symbol.
- An integer representing the frequency of the symbol.

Example Input File:

```
A 3
C 3
B 1
D 2
```

To generate the Huffman tree, you must execute the following steps:

The server program executes the following tasks:

- Read the alphabet information from STDIN.
- Arrange the symbols based on their frequencies. If two or more symbols have the same frequency, they will be sorted based on their ASCII value.
- Execute the Huffman algorithm to generate the tree. To guarantee that your solution generates the expected output from the test cases, every time you generate an internal node of the tree, you must insert this node into the queue of nodes as the lowest node based on its

frequency. Additionally, you must label the edge to the left child as 0 and the edge to the right child as 1.
   - Print the Huffman codes from the generated tree. The symbols' information (leaf nodes) must be printed from left to right.
   - 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. First, receive the binary code from the client program.
2. Next, use the generated Huffman tree to decode the binary code.
3. Finally, return the character to the client program using sockets.

Given the previous input file, the expected output for the server program is:

```
Symbol: C, Frequency: 3, Code: 0
Symbol: B, Frequency: 1, Code: 100
Symbol: D, Frequency: 2, Code: 101
Symbol: A, Frequency: 3, Code: 11
```

The client program:

The user will execute this program using the following syntax:

./exec_filename hostname port_no < compressed_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 compressed_filename is the name of the compressed file. The hostname and the port number will be available to the client as command-line arguments.

The client program receives from STDIN (using input redirection) m lines (where m is the number of symbols in the alphabet). Each line of the compressed file has the following format:

1. A string representing the binary code of the symbol.
2. A list of n integers (where n is the frequency of the symbol) representing the positions where the symbol appears in the message.

Example Compressed File:

```
11 1 3 5
0 0 2 4
101 6 8
100 7
```

After reading the information from STDIN, this program creates m child threads (where m is the size of the alphabet). Each child thread executes the following tasks:

1. Receives the information about the symbol to decompress (binary code and list of positions) from the main thread.
2. Create a socket to communicate with the server program.
3. Send the binary code to the server program using sockets.
4. Wait for the decoded representation of the binary code (character) from the server program.
5. Write the received information into a memory location accessible by the main thread.

Finally, after receiving the binary codes from the child threads, the main thread prints the original message. Given the previous compressed, the expected output is:

```
Original message: CACACADBD
```

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 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. 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 it is done executing each test case.**

Requested files

client.cpp

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

server.cpp

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

huffmanTree.h

```
1   // Optional file to implement an OOP solution for the Huffman Tree
```

huffmanTree.cpp

```
1   // Optional file to implement an OOP solution for the Huffman Tree
```

VPL

Data retention summary

Get the mobile app