COSC 3360 - 15772 - Fundamentals of Operating Systems

Programming Assignment 2

Available from: Friday, 30 September 2022, 12:00 AM
Due date: Sunday, 30 October 2022, 12:00 AM
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 Shannon-Fano-Elias code generator 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

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. Instead, this program receives 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() call in the primer.

Each child process executes the following tasks:

1. First, receive the symbol's information from the client program.
2. Next, use the Shannon-Fannon-Elias encoding algorithm to generate the binary code of the symbol.
3. Finally, return the binary code to the client program using sockets.

The server program will not print any information to STDOUT.

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 the message (string) to be codified. The hostname and the port number will be available to the client as command-line arguments.

The client program receives from STDIN a message (string or char array). The input file has a single line with the message.

Example Input File:

```
AAAAAABBBBBCCCCDDDEE
```

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).

This program determines the alphabet of the message and the frequency of each symbol in the alphabet, sorting the alphabet in decreasing order based on the frequency (if two or more symbols have the same frequency, they will be sorted using the lexicographic order). Then, it creates m child threads (where m is the size of the alphabet). Each child thread determines the binary code for the assigned symbol executing the following steps:

1. Create a socket to communicate with the server program.
2. Send the symbol's information to the server program using sockets.
3. Wait for the binary code from the server program.
4. 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 list of codes. Given the previous input file, the expected output is:

```
SHANNON-FANO-ELIAS Codes:

Symbol A, Code: 001
Symbol B, Code: 011
Symbol C, Code: 1010
Symbol D, Code: 1101
Symbol E, Code: 11110
```

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
```

VPL