Aidan Lambrecht
Advanced Operating Systems
Project 2
January 29, 2021

Assignment 2 Report

Project 2 required students to produce a program that utilized pipes to pass data between two processes, ultimately resulting in the reporting of the word count of a user-specified file. The first step of the project was to create two processes.

README

All files are contained in a tarball called ‘project2.tgz.’ Run the program by typing ‘./pwordcount <filename.txt>’ in the console, or just type ‘./pwordcount’ and follow the on-screen prompts. Text files included in the tarball are ‘test.txt,’ notes.txt,’ and ‘long.txt.’ Their lengths are 8, 1170, and 1528 words respectively.

Creating a Child Process

Since a parent process exists by default when running a program, it is only necessary to work on creating the child process. This is done through the ‘fork()’ command in C. If ‘fork()’ returns a negative number, the program has failed to create a child process, and this is checked in the code.

Creating Pipes

After making the child process, one needs to create a way for the two to communicate. This is done through pipes, unidirectional data transfer busses, created in C with the ‘pipe()’ command. Like with ‘fork(),’ if ‘pipe()’ returns a negative number (-1 to be specific), the pipe creation failed. This is checked for in the code. Ultimately two pipes are created since both of the processes need to send data to the other.

Open and Read File

The next step was to open the text file. Because I misread the instructions, I coded my program to take user input through command line arguments or separate user input, e.g. one can enter the program name and text file name simultaneously or subsequently. An exit command is included in case the program must be terminated prematurely. The ‘openFile()’ function takes a string representing the filename and returns a file object pointer.

The actual reading of the file is done through a while loop in the main function, reading the file in batches of 255 characters until the file is fully read.

Word Count

To count the words in the given text file, a ‘wordCount()’ function was used which takes a string. If the string begins with a character, the count starts at 1. Otherwise, the count begins at 0. While there are more characters to read, the program checks for whitespace of any kind. If it encounters any, a flag is tripped. If a non-whitespace character follows, then the word count is incremented. The final count is returned.

Passing Data

Two pipes are used to transfer data between the two processes. The parent-to-child pipe is used to send data to the child in batches of 255 characters. The batches are then read continuously by the child process until the whole file has been transferred. When each batch is read in by the child, the word count of that batch is found and then added to the total word count of the document. This number is then sent back via the child-to-parent pipe.

Tests

First I needed to test the ability of my program to open and read in files. This was done and proven by having the text file's content written to the console upon reading. Next, the word count function was tested by comparing results of three passages with the word count as found by MicroSoft Word. Then the ability to transfer data via pipes was tested by printing the text file's content to the console in the child process. Finally the ability to transfer the word count back to the parent was tested and found to succeed by once again printing to the console.

Problems Encountered

For a while, I was struggling to send all the data to the child process, and the amount of data that was transferred was inconsistent between runs. This was remedied by making the size of the data transfer a constant instead of a variable amount. This constant was defined at 255.

I also had trouble getting my environment to correctly read in a file to test the program. This was due to using relative paths to define the debugging environment instead of the absolute program paths and was fixed appropriately.