Computer Science 311

Assignment 4 Write-up

Austin Dubina

1 Deviations From the Assignment

- The program does not currently post timings.
- The program does not suppress duplicate words.

2 Assumptions

- The user will use the follow command to execute program, cat (filename) ”pipe symbol” ./uniqify (number of processes)
- The program only has to support alpha characters
- Words in file are separated with a single white space

3 Program Design

Overview:

The uniqify program reads an alphabetic text file delimited by spaces and parses these words into a given number of sorting processes specified by the user. Once these words have been sorted, they are then passed to a separate merge function which collects, suppresses, and prints them in alphabetic order to the terminal screen. All the previously mentioned processes are connected via pipes. It was worth noting that two times the number of unique processes is needed to plumb all the sorting processes to the parent and merging processes.

The program starts by creating a number of pipes equal to two times the number of specified children though the command line. Once the pipes have been created, fork is called to create parallel processes known as the child process in my program. These children sort the parsed words from the parent process (also known as the parser process in my program) using the Unix sort utility program. Once words have been sorted, they are then piped to my merger process which collects all the words and prints them to the terminal through stdout. As previously mentioned, I simply did not have enough time to implement the suppresser in the merger process, however I hardly think this was the main objective of the assignment.

The following is a detailed description of each function and its purpose.

Program Functions:

`int main()`
Purpose:
This function serves as the main entry point into my program

Description:
Main starts by creating a number of pipes equal to two times the number of processes specified by the user. Main is responsible for forking off each child process and cleaning up any leftover file descriptors.

`void make_pipes()`
Purpose:
To create a number of pipes equal to two times the number of processes specified by the user at the start of the program.

Description:
Each file descriptor created from the pipe function is stored in a two dimensional array. Each colum of the array holds the file descriptor for the read / write of each pipe, or row of the array. Because I did not want to create a three dimensional array, I simply called make pipes twice to create two sets of arrays. The first set of file descriptors is denoted as "parser to sort", and the second set is denoted as "sort to merger".

`void parser_process()`
Purpose:
Scan and separate words out of texted file. Send parsed words in a round robin fashion to child processes for sorting.

Description:
The parser process starts by closing all unused file descriptors. Because each process gets a copy of each file descriptor from the main process it is important to close all file descriptors belonging to the "sort to merger" set. Also, because all scanned input is being processed through stdin, all file descriptors belonging to read end of the pipe is closed as well. The function uses the fscanf function to read words delimited by whitespaces into a buffer. It is important to note that any non-alpha characters read will likely result in a program crash. Once a word has been stored into a buffer, characters are forced to lowercase then fputs is called to send the word to a sorting process in a round robin fashion. Once fscanf has reached the end of file, the function exits its loop and closes any remaining file descriptors.

`void child_process()`
Purpose:
Sorts parsed words from parent process and writes them merger process.

Description:
The child processes start by closing all unused file descriptors. Since both the "parser to sort" and the "sort to merger" set of file descriptors are needed to connect or "plumb" all the processes together, only the read end(s) of the "sort to merger" pipes; the write end(s) of the "parser to sort" pipes are closed as well. The sorting is done by calling the Unix utility program "sort" using the execlp function. However because sort typically reads from stdin and writes to stdout, these pipes need to be re-directed, this is done by calling the sort function in my program (confusing I know).

`void sort()`
Purpose:
Redirect the read and write ends of the Unix sort utility program.

Description:
The sort function redirects both the read and write end(s) of the Unix sort utility program by calling the dup2 function which duplicates one file descriptor, making them aliases, and then deleting the old one.

`void merger_process()`
Purpose:
Reads from all sorting processes, suppresses duplicate words, and prints each word in order to the terminal.

Description:
The merger process starts like every other process by closing all unused file descriptors. However, unlike the parent process for example, the "parser to sort" set of file descriptors are unused. Also, instead of writing to each pipe the merger reads from the "sort to merger" set of pipes. As noted in the "Deviations From the Assignment" section, I was unfortunately unable to suppress duplicate words from the child. Currently the merger process and just reads from the sorting process and prints to the terminal in a round robin fashion. Given more time, I am sure this can be added fairly easily.

4 Additional Questions

1. what do you think the main point of this assignment is?
   The main purpose of this assignment is to begin thinking about parallel processes using fork and how connect these processes though the use of pipes.

2. how did you ensure your solution was correct? Testing details, for instance.
   I systematically went through each function to test every imaginable test case. As part of my check, I put error checking on most system calls and functions. I also test my program with very large text files. I also listed my assumptions at the top of this document for anything I was not sure about.

3. what did you learn?
   I have painfully learned a lot during this assignment. Before this assignment, I have never written any kind of program that executes several processes independent of one another. Getting my head wrapped around this concept alone was a major challenge. Once I accepted that several processes were executing simultaneously in the same code segment, I then had to take another leap of faith and being sending data to and from these processes using pipes. I cannot tell you how ecstatic I was the first time my program successfully sent a word from the parser to the merger without a glitch. I also learned that working with strings in c is a total pain which is partially why I was unable to suppress duplicate words in my merger.

5 Difficulties

The biggest difficulty I had with this assignment was keeping track and properly closing all unused file descriptors. Not properly closing file descriptors caused my program to hang for the longest time, however once I figured out a process for managing and closing these file descriptors, the assignment became much more bearable.