LE/EECS 3221 – Operating System Fundamentals

Summer 2020

Programming Assignment 1

Submission Deadline: June 07, 2020 before 23:59

Objectives

In this assignment we will try to practice the concept of parent-child process, inter-process communication and some related system calls.

General Assignment Notes

When writing and submitting your assignments follow these requirements:

- Name your source code file as: your YorkU student number, an underscore, 'a' (for 'assignment', then the assignment number in two digits. For example, if the user 100131001 submits Assignment 1, the name should be: `100131001_a01.c.txt`. No other file name format will be accepted. We require the .txt extension in the end because Moodle does not allow .c extension.
- For this assignment you must use C99 language syntax. Your code must compile using make **without errors and warnings.** You will be provided with a makefile and instructions on how to use it. If you use that makefile to compile the code, then you don't have to do anything special to select C99.
- **Test your program thoroughly with the `gcc` compiler (version 5.4.0) in a Linux environment.** This can be verified by running the command "gcc --version" in Linux shell.
- If your code does not compile, **then you will get zero**. Therefore, make sure that you have removed all syntax errors from your code.

Marks will be deducted from any question(s) where these requirements are not met.

WARNING

Follow the assignment instructions to the letter in terms of the file names and function names, as this assignment will be auto graded. If anything is not as per description, the auto grading will fail, and your assignment will be given a mark of 0.

Synopsis

In this assignment, you will have a parent process that will create children processes to perform tasks and will collect the output from these children processes. There are three tasks that should be performed:

- Read the input file that contains Linux shell commands. Parent process will create a child process to read that file and child process will return the content of that file in the form of a string **using a shared memory area**.
- Execute the Linux shell commands read from the input file and execute them one by one. A child process will be created to execute these commands and the output will be returned by the child process in the form of string **using pipe**.
- The parent process will write the output of commands to the screen.

Description

Write a C program that includes the code for following tasks. Write all the code in single file:

1. Write the parent program as per the description in the "Synopsis". The parent program must use `fork` system call to create children processes when required.
2. In order to create/use/clear shared memory buffers, use the POSIX API system calls mentioned in the book in chapter 3.
3. Read the sample input file, sample_in.txt. This file must be read by a child process and not the main/parent process. The name of the file must be given to the main/parent process through command line arguments.
    a) This file contains one shell command per line.
    b) The child process will write the content of the file to a shared memory area so that the parent process can read from there.
    c) Once file reading is done, the child process will be terminated.
4. Copy the commands from the shared memory area to a dynamically allocated array.
5. Execute the commands one by one with the help of a child process:
    a) Use a suitable version of `exec` system call to execute shell commands.
    b) The child process must write the output of the command to a pipe from where the parent process will read it and display to the screen.
    c) You may use only one child process by doing fork once to execute all the commands or fork again and again.
    d) The parent process **must use** the provided output function, `writeOutput()`, to write the output to the screen. Invoke the output function iteratively, for each command. **Warning: If you will not use this function to display the output then your outcome will not match with our outcome and you will be awarded zero.**
6. The other implementation details are on your discretion and you are free to explore.

7. In order to test your code, use the `sample_in.txt` and compare the output of your program for this input file with the content of `sample_out.txt`. However, keep in mind that due to the nature of the commands, the output is specific to the content of the directory in which you run your program.
8. The following flow chart describes the flow of the program.