CS 392 Spring 2024 Systems Programming — Homework 5

1 Objective

For this assignment, you will be creating a program that can sort the output of `ls -ai <DIRECTORY>` command by their inode numbers. In `ls -ai`, the flag `-a` is to show all the files under a directory, while `-i` is to show the files inode numbers.

Assume after compilation, your executable is called `sl`. The program is invoked as follows:

```
$ ./sl  <DIRECTORY>
```

The program then will create two child processes: one will execute `ls -ai <DIRECTORY>`, and send its output to the other process, which will then execute `sort` command, and print the output to the terminal.

2 Tasks

2.1 Task 1: Validating Command-Line Arguments

When invoking program `sl`, you need to check the first command-line argument is a directory that can be read. If it's not a directory, you should print

```
The first argument has to be a directory.
```

If it is a directory but can't be read, you should print

```
Permission denied. <DIRECTORY> cannot be read.
```

where `<DIRECTORY>` should be replaced by the string passed to the terminal.

All error messages should be printed to `stderr` with an exit code of `1`.

2.2 Task 2: Sorted File List

Task 2 is the main task.

You will create two child processes, one for `ls -1ai <DIRECTORY>`, and one for `sort`. You will connect the `stdout` of `ls` to the `stdin` of `sort` using one pipe, then you will connect a second pipe to the `stdout` of `sort`, which your parent process will read from. Last, in the parent process, you will print the total number of files.

For example, when the command line is:

```
$ ./sl /home/
```

The output might look like this:

```
     2 ..
  1497 .
259329 ubuntu
Total files: 3
```

You will be working with pipes (with raw file descriptors), using `fork()` and `exec*()`, using `dup2()` to duplicate file descriptors, and making sure you close all unused file descriptors. You may also use `read()` and `write()`.

You should check the return code of all the system and function calls you use in your program. If any of them fails, print an error message and return `EXIT_FAILURE`. We will look through your code to verify you check return codes and take the proper action.

We will, however, check for specific output for failures that occur when `exec()`-ing `ls` and `sort`. If an error occurs, we expect to see `Error: ls failed.` or `Error: sort failed.` written to `stderr`.

This is a short assignment in terms of lines of code, but the lines are a bit tricky to write and hard to debug. Start early, and ask questions early and often.

A few common pitfalls are as follows:

- *My program hangs!*
  Make sure you have closed all file descriptors that you do not need. If the write end of a pipe is still unclosed, the process on the read end of it will think it is still open;

- *My pipe doesn't work!*
  Check the return value of the call to `dup2()`, and make sure it's succeeding. It could be that you're using it wrong. Also, start small. Try a toy program with one child process which just writes some string to the pipe for the parent to read. Get comfortable working with that and then build up to the full solution.

3 Grading

The homework will be graded based on a total of 100 points:

- Task 1 & 2 (100 pts): 10 test cases in total, **10** points each.

After accumulating points from the testing above, we will inspect your code and apply deductibles listed below. The lowest score is 0, so no negative scores.

- **-100:** the code does not compile, or executes with run-time error;
- **-100:** the code is generated by AI, and/or through reverse engineering on the tester;
- **-100:** did not use raw file descriptors and pipes (*i.e.,* `pipe()` function call);
- **-100:** used `system()` function instead of `exec*()`;
- **-100:** used `dup()` instead of `dup2()`;
- **-100:** did not create multiple child processes;
- **-100:** did not invoke `sort` command; used `qsort()` or any sorting algorithms to sort;
- **-100:** used `readdir()` or related structs/functions to count the number of files;
- **-10:** no pledge and/or name in C file.

Earlybird Extra Credit: 2% of extra credit will be given if the homework is finished two days before the deadline.

For specific policy, see syllabus.

Deliverable

Submit a single `sl.c`.