April 1st, 2020

CSE344 – System Programming - Homework #2 - v5
Signal handling

The program that you must develop, will consist of two processes: the parent process P1 and its child process P2 **that will run concurrently**. We want the entire program to finish as soon as possible. That means, the processes must spend the minimum possible time as idle. In other words, P2 must process the data produced by P1 concurrently and ASAP without waiting for P2 to finish.

P1 will admit two command line parameters as paths to regular files (either ASCII or BINARY)

```
./program –i inputPath –o outputPath
```

P1 will read the contents of the file denoted by `inputPath`, and every couple of unsigned bytes it reads, will be interpreted as a 2D coordinate (x,y). For every 10 coordinates (i.e. every 20 bytes) it reads, it'll apply the **least squares method** to the corresponding 2D coordinates and calculate the line equation (ax+b) that fits them. This calculation will be considered a critical section, and is not to be interrupted by SIGINT/SIGSTOP. Then, P1 will write in a comma separated line the 10 coordinates followed by the line equation as a new line of a temporary file created via `mkstemp`.

```
(x_1, y_1),….,(x_10, y_10), ax+b
```

and proceed to the next 20 bytes and repeat.

After finishing processing the contents of the input file, P1 will terminate gracefully by closing open files and printing on screen how many bytes it has read as input, how many line equations it has estimated, and which signals where sent to P1 while it was in a critical section.

P2 will be forked early by P1, and will read the contents of the temporary file created by P1, line by line, and **for every line it reads**, it will calculate the mean absolute error (MAE), mean squared error (MSE) and root mean squared error (RMSE) between the coordinates and the estimated line. This calculation will be considered a critical section, and is not to be interrupted by SIGINT/SIGSTOP (are they blockable??). It will then remove the line it read from the file and write its own output to the file denoted by `outputPath` as 10 coordinates, the line equation (ax+b) and the three error estimates in a comma separated form:

```
(x_1, y_1),….,(x_10, y_10), ax+b, MAE, MSE, RMSE
```

After processing all the contents of the temporary file and making sure that no more input will arrive, P2 will terminate and print on screen for each error metric, its mean and standard deviation.

Attention: P2 will remove/delete the file denoted by `inputPath` after P1 is done with it. How will P2 know whether P1 has finished writing to it? P1 will send SIGUSR1 to P2 with this purpose.

Attention again, if P2 gets more chance to execute than P1, it might not find anything to read in the temporary file. Use `sigsuspend` to make sure P2 waits until there is some input available in the file, if P1 is not yet done with it. **Don't use busy waiting or any form of sleep.**

Attention some more, both processes will be writing/reading to the same file...be extra careful with locks and offsets...

Attention (sigh), in case of SIGTERM, in either P1 or P2, your processes must catch it and clean up after them before exiting gracefully, by closing open files, and removing the input and temporary files from disk.

Rules:

- use the `getopt()` library method for parsing commandline arguments.
- Use system calls for all file I/O purposes, don't use standard C library functions.
- Your programs are not allowed to crash due to any foreseeable reason. In case of an error, exit by printing to stderr a nicely formatted informative errno based message.
- If the command line arguments are missing/invalid your program must print usage information and exit.
- Don't forget to lock the files while reading and writing into them.
- **You are not allowed to create/use any additional files besides those explicitly provided and the one temporary file created via mkstemp**
- All mathematical operations will be realized with an accuracy down to 3 decimal points.
- Do your best to produce a robust program that will not crash even with the most evil user executing it.
- Compilation must be warning-free.
- You will provide a demonstration of your program to the course assistants. They will provide the details.
- No late submissions will be allowed.
- Close all files and free all resources explicitly.

Submission:

- your source files, your makefile and a report on how you solved the issues in this homework.

Grading:

- Does not compile: fail
- Compiles but crashes despite normal input: fail
- Crashes for n reasons: -10 * n
- Violating the homework rules: fail
- No report: -30

Good luck.