CPSC 351 - Operating Systems Concepts

Project 1, Fall 2021

due Friday, September 17 at 9:45 pm PDT

Last updated Monday September 8, 2:00 pm PDT

In this project you will use the POSIX API to create three command line utilities:

- **`thrice`** - runs a given command three times
- **`stderr`** - runs a command and saves the contents of its standard error stream to a file
- **`foreach`** - repeats a command with a list of arguments

The project may be completed individually, or in a team of up to three students.

Platform, Compiler, and Tools

As specified in the Syllabus, your programs should be written in C++ and run on Tuffix 2020. You may use either the GCC or Clang compiler. Your code must compile cleanly (with no errors or warnings) with the following compiler flags:

```
-std=c++17 -Wall -Wextra -Wpedantic -Werror
```

You must supply a `Makefile` with the following targets:

- one target for each executable (`thrice`, `stderr`, `foreach`)
- **`all`** - build all three executables. This target should be the default.
- **`clean`** - remove intermediate files and executables so that the project can be re-built from scratch

Targets for testing each utility are encouraged, but not required.

Libraries and Code

This project must be implemented in C++17 with the POSIX API (i.e. `<unistd.h>` and associated header files). Code from the textbook, the man pages, the GitHub repositories for the textbook, and examples provided by the instructor may be reused. All other code must be your own original work or the original work of other members of your team.

Command-Line Tools

If no arguments are specified to any of the following tools, write a message describing the usage of the program to `cerr` and exit with `EXIT_FAILURE`.

thrice

The command-line arguments for this program should form a command and its arguments to be executed in a child process. Repeat the command three times in sequence, waiting for the previous instance to complete before running the next instance.

```
prompt> ./thrice
Usage: ./thrice PROG [ARGS]...
```

```
prompt> ./thrice ps
  PID TTY          TIME CMD
 2393 pts/0    00:00:00 bash
 2405 pts/0    00:00:00 ps
  PID TTY          TIME CMD
 2393 pts/0    00:00:00 bash
 2405 pts/0    00:00:00 ps
  PID TTY          TIME CMD
 2393 pts/0    00:00:00 bash
 2405 pts/0    00:00:00 ps
```

```
prompt> ./thrice echo hello world
hello world
hello world
hello world
```

---

stderr

Use the technique shown in Figure 5.4 of the textbook with `STDERR_FILENO` to write the contents of a command's standard error stream to a file.

The first command-line argument should be the name of the file to which the contents of the standard error stream should be written. The remaining command-line arguments are the command to be run.

```
prompt> ./stderr
Usage: ./stderr FILE PROG [ARGS]...
```

```
prompt> ./stderr output.txt strace ls -l
total 56
-rw-r--r-- 1 student student 11770 Sep 18 16:18 output.txt
-rwxr-xr-x 1 student student 17672 Sep 18 16:17 stderr
-rwxr-xr-x 1 student student 17672 Sep 18 16:18 stderr.cpp

prompt> cat output.txt
execve("/usr/bin/ls", ["ls", "-l"], 0x7fffc7f8c518 /* 19 vars */) = 0
brk(NULL)                               = 0x7fffbf7b1000
arch_prctl(0x3001 /* ARCH_??? */, 0x7fffc7b73fc0) = -1 EINVAL (Invalid argument)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
...
```

If you are familiar with Bash or the POSIX shell, you will recognize that effect of this program is similar to the following shell command:

```
prompt> strace ls -l 2> output.txt
```

If you are familiar with strace(1), you will also know that the effect is similar to the following command:

```
prompt> strace -o output.txt ls -l
```

foreach

The command-line arguments for this program are a list of arguments to be supplied one-at-a-time to a command to be executed repeatedly:

```
prompt> ./foreach
Usage: ./foreach [-p] [-n NUM] [ARGS]... PROG

prompt> ./foreach hello world echo
hello
world
prompt> ./foreach www.fullerton.edu www.ecs.fullerton.edu host
www.fullerton.edu has address 137.151.127.120
www.fullerton.edu mail is handled by 10 mail.fullerton.edu.
www.ecs.fullerton.edu is an alias for ecs.fullerton.edu.
ecs.fullerton.edu has address 137.151.27.1
ecs.fullerton.edu mail is handled by 10 mxa-00039101.gslb.pphosted.com.
ecs.fullerton.edu mail is handled by 20 mxb-00039101.gslb.pphosted.com.
```

There should be two optional arguments, retrieved by `getopt(3)`:

- **`-p`** - run each instance of the command in parallel (i.e. do not wait for one child process to finish before running the next)
- **`-n`** `NUM` - run the command the specified number of times

---

```
prompt> ./foreach -p www.fullerton.edu www.ecs.fullerton.edu host
www.ecs.fullerton.edu is an alias for ecs.fullerton.edu.
ecs.fullerton.edu has address 137.151.27.1
www.fullerton.edu has address 137.151.127.120
ecs.fullerton.edu mail is handled by 10 mxa-00039101.gslb.pphosted.com.
ecs.fullerton.edu mail is handled by 20 mxb-00039101.gslb.pphosted.com.
www.fullerton.edu mail is handled by 10 mail.fullerton.edu.

prompt> ./foreach -n 2 hello world echo
hello
hello
world
world

prompt> ./foreach -n 3 'hello world' echo
hello world
hello world
hello world
```

Submission

Your project should include summary information in a README file as described in the Syllabus. Only one submission is required. Be certain to identify the names of all team members at the top of the README.

Submit a tarball (`.tar.gz`, `.tgz`, `.tar.Z`, `.tar.bz2`, or `.tar.xz`) file containing the following through Canvas before 9:45 pm PDT on the due date:

1. A README file as described in the Syllabus
2. The source code for each utility in C++17
3. A `Makefile` as described above.

Do **not** include compiled executables or other binary files. If you use git, this includes the contents of the `.git/` directory. See Git Archive: How to export a git project for details.

If you include other files in your tarball, I will not examine them unless your README states explicitly that they should be included in the evaluation of your project.

The Canvas submission deadline includes a grace period of an hour. Canvas will mark submissions after the first submission deadline as late, but your grade will not be penalized. If you miss the second deadline, you will not be able to submit and will not receive credit for the project.

Note: do not attempt to submit projects via email. Projects must be submitted via Canvas, and instructors cannot submit projects on students' behalf.

See the following sections of the Canvas documentation for instructions on group submission:

- How do I join a group as a student?
- How do I submit an assignment on behalf of a group?

Grading

The grade for the project will be assigned on the following five-point scale:

Exemplary (5 points)

Results are correct; code and documentation are clearly written; organization makes it easy to review.

Basically Correct (4 points)

Results are (mostly) correct, but the code or documentation is not easy to follow or is not clear.

Right Idea (3 points)

The approach is appropriate, but the work has mistakes in code or documentation that undermine the correctness of the results.

Solid Start (2 points)

The work makes a good start, but does not include documentation or has fundamental conceptual problems in code that do not produce legitimate results.

Did Something (1 point)

The solution began an attempt, but is either insufficiently to assess correctness or is on entirely the wrong track.

Did Nothing (0 points)

Project was not submitted, or submission was of such low quality that there is nothing to assess.

Acknowledgements: this grading scale is drawn from the general rubric used by Professor Michael Ekstrand at Boise State University.