

Laboratory Exercise 7

cpe 357 Fall 2020

```
"...one of the main causes of the fall of the Roman Empire was that,  
lacking zero, they had no way to indicate successful termination of  
their C programs."  
-- Robert Firth
```

— /usr/games/fortune

Due by (or before) 11:59pm, Monday, November 13th.
The Laboratory Exercises are to be done individually.

Note: When developing these programs, work on the workstations in the lab or on personal workstations. A runaway process repeatedly `fork()`ing can bring a large server to its knees. Your classmates will not love you if you crash `unix1` or the other shared machines.

Laboratory Exercises

Programs: `fork()` and `exec()` in two lessons

1. **forkit** This program demonstrates the process. `forkit` starts up, announces itself, then splits into two processes. The child announces itself and exits. The parent identifies itself, waits for the child, then signs off. All this is shown in Figure 1.

Parent	Child
— <i>before the fork</i> —	
greet the world fork()	— —
— <i>after the fork</i> —	
print out its pid wait() for child	print out its pid exit
— <i>after the exit</i> —	
say goodbye exit	— —

Figure 1: The expected behavior of `forkit`

Sample Run

Note, the order of execution of parent and child is nondeterministic. Try it on different systems and you will see different results. This example is from a linux machine.

```
% forkit  
Hello, world!  
This is the child, pid 2112.  
This is the parent, pid 2111.  
This is the parent, pid 2111, signing off.  
%
```

2. **tryit** takes a command-line argument of the path to a program (absolute or relative), forks a child that tries to `exec()` the given program, and reports on its success or failure. A child that exits with a status of 0 is assumed to be a success, non-zero is a failure. If the `exec()` fails, the child should print why (via `perror()`), and exit with a non-zero status.

This program does not have to support command line arguments to the other program. That will be part of Asgn 7's shell.

The process is shown in Figure 2.

Parent	Child
— <i>before the fork</i> —	
check the command line args fork()	— —
— <i>after the fork</i> —	
wait() for child	exec the given program
— <i>after the exit</i> —	
report on child's success exit with child's status	— —

Figure 2: The expected behavior of **tryit**

Sample Runs

```
% tryit
usage: tryit command
% tryit command with args
usage: tryit command
% tryit non-existent
non-existent: No such file or directory
Process 2359 exited with an error value.
% tryit ls
ls: No such file or directory
Process 2361 exited with an error value.
% tryit /bin/ls
Makefile RCS forkit forkit.c tryit tryit.c
Process 2369 succeeded.
% tryit /bin/false
Process 2371 exited with an error value.
% tryit /bin/true
Process 2373 succeeded.
```

Tricks and Tools

- Be sure to familiarize yourself with the process-management system-calls: `fork(2)`, the execs (`execl()`, `execvp(3)`, `execle(3)`, `execv(3)`, `execvp(3)`, and `execve(2)`), and `wait(2)`.
- Also look into `getpid(2)` for finding out one's own process-id.
- As seen in the example above, `/bin/true` always exits with a successful exit code, and `/bin/false` always exits with an unsuccessful one.

- The `ps` command (`/bin/ps`) will show you processes belonging to you. Be sure you know how to use it so you can clean up after yourself.

What to turn in

For the Laboratory Exercises: Submit via handin in the CSL, to the `lab07` directory, your two programs, `forkit.c` and `tryit.c`

A makefile (called `Makefile`) that will build your programs when given either the target “`all`”.

A README file that contains:

- Your name(s). In addition to your names, please include your Cal Poly login names with it, in parentheses. E.g. (`pnico`)
- Any special instructions for running your program.
- Any other thing you want me to know while I am grading it.

The README file should be **plain text**, i.e, **not a Word document**, and should be named “README”, all capitals with no extension.