# CSCI 4220 Lab 2

## Lab 2: fork() and SIGCHLD

In this lab you will practice using `fork()` and handling `SIGCHLD` calls. You may want to make use of **unp.h** since it has many common includes and provides the **Signal()** function. To compile `lab2.c` inside the `unpv13` directory, you would write `gcc -o lab2.out lab2.c lib/libunp.a` and would `#include "lib/unp.h"` inside your code. You can do this from any directory by adjusting the path in the `#include` and the compile line.

In order to receive credit, write a C program that asks for a number of children to spawn from **STDIN**. Your program should immediate create that many children using `fork()`. All children have a common parent, the original program has no grandchildren. When a child is made, the parent should report that a child has been made and provide the PID. The child should immediately pick a random amount of time between 0 and 5 seconds to wait, and should print its PID and the amount of time it will wait. After waiting that amount of time, the child should print that it is terminating (and include its PID), and then exit. If all your processes pick the same amount of random time, and this happens every time you run your program, you should read the hint - this is not correct behavior.

The parent should use `SIGCHLD` to detect when children terminate and immediately print the PID of the child that terminated. Once all children have terminated, the parent should also terminate.

*Hint*: You may want to make use of **getpid()**, **rand()**, **srand()**, **time()**, and **sleep()**. You should also remember that your computer will probably spawn all the processes within 1 second, so you may want to add something unique to each child when using **srand()**.

Sample program execution:

```
./a.out
Number of children to spawn: 5
Parent spawned child PID 16738
Child PID 16738 dying in 1 seconds.
Parent spawned child PID 16739
Child PID 16739 dying in 1 seconds.
Parent spawned child PID 16740
Child PID 16740 dying in 3 seconds.
Parent spawned child PID 16741
Child PID 16741 dying in 3 seconds.
Parent spawned child PID 16742
Child PID 16742 dying in 2 seconds.
Child PID 16738 terminating.
Parent sees child PID 16738 has terminated.
Child PID 16739 terminating.
Parent sees child PID 16739 has terminated.
Child PID 16742 terminating.
Parent sees child PID 16742 has terminated.
Child PID 16740 terminating.
Parent sees child PID 16740 has terminated.
Child PID 16741 terminating.
Parent sees child PID 16741 has terminated.
```

## Extra Credit

For an extra lab point, implement the above program but with an efficient signal handler. There should be no printing or busy waiting inside the signal handler. The parent should still report when a process is terminated as soon as possible. You are allowed to use global variables (a common practice with signal handlers) and may want to use a busy waiting loop (a common practice with servers). If you're worried about wasting CPU cycles, you can make use of **sleep()** while waiting for childrent to terminate.