Wack a Mole!

### Due Date: November 14, 2014 @ midnight Submission Subject: “Daemon”

## General Submission Criteria:

* See Lab 0 for the General Submission Criteria!
* Make a directory in your repository: lab6
* Include all of your Lab6 work within the lab6 directory

## Overview:

In this lab, you will create a daemon and a child process . Daemons are process that live for a long time. They are often started when the system is bootstrapped and terminate only when the system is shutdown. Because they don’t have a controlling terminal, we say that they run in the background.

The daemon that you create will spawn and terminate a child process upon receiving a signal. For this daemon, you need to handle three signals:

* SIG\_TERM: shut down any and all child of the daemon process, and then exit
* SIG\_USER1: terminate child process #1, and spawn a new child process
* SIG\_USER2: terminate child process #2, and spawn a new child process

When your system is running properly, you will have at most **three** process running: the daemon and one child.

## Executable Names:

**spiritd**: a program that is intended to run as a daemon, and based upon signals will either create or destroy child processes.

**mole:** a program that emits the following messages to ~/lab6.log

- fprintf(logfile, “Pop %s\n”, argv[0]);

## Makefile Targets:

all: (default)

spiritd:

moles:

clean:

## Part 1 Description: Daemon Preparation

In this software project, you are to write a C program that acts as a daemon. To create this daemon, there is a number of coding standards you should use.

See Chapter 13 in [Advanced Programming in the Unix Environment](http://proquest.safaribooksonline.com.libproxy.csun.edu/book/programming/unix/9780321638014/praise-for-advanced-programming-in-the-unix-environment-second-edition/pref01_html?query=((Advanced+Programming+in+the+UNIX®+Environment%2c+Third+Edition))#X2ludGVybmFsX0h0bWxWaWV3P3htbGlkPTk3ODAzMjE2MzgwMTQlMkZjaDEzX2h0bWwmcXVlcnk9KChBZHZhbmNlZCUyMFByb2dyYW1taW5nJTIwaW4lMjB0aGUlMjBVTklYJUMyJUFFJTIwRW52aXJvbm1lbnQlMkMlMjBUaGlyZCUyMEVkaXRpb24pKQ==) for more information.

These coding standards include:

1. Set the file creation mask to 0 (see umask(2))
2. Fork and have the parent exit (see fork(2))
3. Create a new session (see setsid(2))
   1. create a new process group
4. Change the current working directory to be “/” (see chdir(2))
5. Close all unneeded file descriptors (see getrlimit(2))
6. Reopen the standard file descriptors to map to /dev/null (see dup(2))

When the program is running, it will respond to external stimuli (i.e., signals).

## Part 2 Description: Signal Handling and Child Process

In this part of the software project, you are to modify your daemon to handle three signals, and based upon these signals the daemon will either create or destroy a child process. These signals should ONLY be handled by the daemon process and not the child process.

* The daemon handles: SIG\_TERM, SIG\_USR1, and SIG\_USR2 (see signal(2))
* Upon SIG\_TERM, the program kills all child processes and  
  shutdowns the daemon gracefully. (see kill(2))
* Upon SIG\_USR1, the program will
  + kill child process #1 (mole1)
  + randomly create either mole1 or mole 2 if it does not already exists
* Upon SIG\_USER2, the program will
  + kill child process #1 (mole2)
  + randomly create either mole1 or mole 2 if it does not already exists
* When a new mole is created the following steps are followed:
  + fork a new process (see fork(2))
  + randomly determine the child process number (either 1 or 2)
  + exec the program mole,   
    with the value of argv[0] set to either (see execv(2))
    - mole1
    - mole2

In this part of the software project, you will also create a trivial program called “mole”. This program simply writes a signal line to a well-known log file (~/lab6.log). The program writes either the string:

* Pop mole1
* Pop mole2

The mole program knows what to write based upon the value of argv[0]