**Programming Assignment 2**

**Due Friday, September 23, 2016 at 11:59 p.m. Uploaded to Canvas**

**Part 1. Sleepy**

Write a program named sleepy that gets a loop count from the command line as in

**sleepy n**

where **n** is the number of seconds for which the program should run. Implement this timing by using a loop that iterates **n**times.  The body of the loop will contain the statement **sleep(1)**. This will put the program to sleep for one second **n**times before exiting.  If **n**is not provided use a default value of **5**.  Here is the code needed to pass an argument to a program from the command line:

**int main(int argc, char \*argv[ ]) {**

**int count;**

**if (argc == 2)**

**count = atoi(argv[1]);**

**else**

**count = 5;**

**}**

**#include <unistd.h>  
...  
unsigned int sleep(unsigned int seconds);**

This function causes the calling process to be suspended until either

1.   the amount of wall clock time specified by **seconds** has elapsed, or

2.   a signal is caught by the process and the signal handler returns.

The return value from **sleep** is **0**, or the number of unslept seconds if the sleep function has been interrupted by a signal interrupt.  You may assume that the program always sleeps the number of seconds requested.

In each iteration print out the process ID and the loop count so that that particular process can be identified.

The process ID can be obtained from the **getpid**function:

**#include <sys/types.h>  
#include <unistd.h>  
...  
pid\_t getpid();**

This function returns the process ID of the calling process.

The process ID is returned as a type **pid\_t,** which is actually an integer so it can be treated as such in a **printf** format statement (use an **int** cast to avoid a compile warning from **gcc**).

Using the **-o** compiler option will allow you override the default **a.out**executable name.

Here is an example run of sleepy.  
   
**sleepy 4**

**176987; START**

**176987; TICK 1**

**176987; TICK 2**

**176987; TICK 3**

**176987; TICK 4**

If you do not wish to override the default **a.out** executable name an example run might look like this:

**./a.out 4**

**176987; START**

**176987; TICK 1**

**176987; TICK 2**

**176987; TICK 3**

**176987; TICK 4**

Run your program several times.  Does it always have the same process id?  What state do you think the process transitions to after a call to the sleep function? Include your answers as comments in your source code.

**Part 2. Zombie**

Write a C program that forks a child process that ultimately becomes a zombie process. This zombie process must remain in the system for at least 10 seconds. Process states can be obtained from the command

**ps -l**

The process states are shown below the S column; processes with the state of **Z** are zombies. The process identifier (pid) of the child process is listed in the PID column, and that of the parent is listed in the PPID column.

The easiest way to determine that the child process is indeed a zombie is to run your program in the background using the **&** (**./a.out &**), and then run the command

**ps -l**

to determine whether the child is a zombie process. Because you do not want too many zombie processes existing in the system, you will need to remove the one that you have created. The easiest way to do this is to terminate the parent process using the **kill** command. For example, if the process id of the parent is 4884, you would enter

**kill -9 4884**

**Part 3. Name**

Create a text file <yourname\_ID>.txt in the programming directory and put your full name and ID number in there. Include this file when you turn in your programming assignment. Do not turn in your assignment as separate files. Move your assignment to your local machine and zip the folder. Attach the zipped folder to your assignment submission on Blackboard.

Reminder: Although most or all of this programming assignment should "just work" in many other environments (cygwin, OSX, solaris, etc.), note that (1) I will not be able to assist in setting up or debugging problems caused by differences in the environment and (2) statements like "it worked on my home machine" will not be considered in the grading process. If you choose to do development in an unsupported environment, it is *your responsibility* to leave adequate time to port your solution to the supported environment, test it there, and fix any problems that manifest.