ECE 362 – Embedded Operating Systems

Process system calls

This assignment will provide the opportunity to develop programs that use the fork, exec, and pipe system calls. Please use the guidelines described in previous assignments. Programs should only use read and write system calls for obtaining input and producing output.

Be careful! Often students unintentionally create extra processes that don't terminate. ***Please regularly run the following command, especially before logging off.***

```
ps -ef | grep $USER | less
```

Reviewing the list of your processes will help you determine if you've created "runaway" processes. If you have, use the command `kill -9 pid`, where *`pid`* is the id of the process you want to terminate.

1. Executable name: timer

Timer should run a specified program with its arguments and then print out the amount of time (in seconds) the program needed to execute (use the `time()` function). For example, "`timer ls -l /bin`" would run the command `ls` and pass the arguments `-l` and `/bin`

2. Executable name: piper

Write a program that creates two child processes. The parent should then read input from STDIN that it sends to both children using pipes.

- The program input should be a sequence of ASCII strings, separated by spaces or newlines. The children will need to covert each string into an integer value.
- One child should add the integers and output the sum. The other child should multiply the numbers and output the product. Be sure the results are converted into ASCII strings. You can assume the computations will not produce underflows/overflows.
- Your solution should be insensitive to the number of input characters (i.e. input from STDIN should terminate with a Control-D). Be sure that every process ``cleans-up'' before terminating.

For example: if the program receives the input:

```
3 4 -1
50
```

The output should be the two lines (in either order):

```
Sum: 56
Product: -600
```

Turn in: a single Makefile and your source code. The Makefile must include targets "timer" and "piper" that will compile individually the programs and a target "all" that compiles both programs.

A few suggestions:

- Start by first writing an outline that can be used to create comments in your program.
- Build your program in stages where not all requirements need to be addressed in each stage. For example, integers may be positive or negative, but you might start with a stage where they can be only positive and a single digit.
- Break down program development into many small steps (some ideas below). It can be very helpful to write small programs that experiment with system calls individually or manipulate data structures.
- After each small step is successful, save the code (perhaps using Git).
- Check return codes from all system calls.
- When using new functions and constructs, confirm each block of code for each comment is working. During development, this might be done using the gdb debugger or by adding print statements.