Akshay Thejaswi
athejaswi@wpi.edu
CS 3013 Project 1
1/28/2014

Files:

runstat.c – utility function to run a command and then print out its statistics
runstat.h – Header file for runstat.c
doit.c – main source for solution to part 1
shell.c – main source for solution to part2
shell2.c – main source for solution to part3
Makefile – makefile for project
shell_test – test commands for shell program
shell2_test – test commands for shell2 program
doit.out – output of test with doit program (interactive commandline output)
shell.out – output of test with shell program ("./shell < shell_test")
shell2.out – output of test with shell2 program ("./shell2 < shell2_test")

Building:

The makefile make all projects by default. It also supports clean, and building each project individually.

make – build all parts
make doit – build doit (part 1)
make shell – build shell (part 2)
make shell2 – build shell2 (part 3)
make clean – remove compiler output files

Running:

doit - "./doit [command]"
shell - "./shell"
shell2 - "./shell2"

Runstat function (*fork and exec)*:

I abstracted the functionality of running a process and printing statistics into a function called "runstat()", in runstat.c , so that all three programs can call it to run a process. It takes the argv of the command as an argument. The runstat function forks itself, then the child call execvp() to run the given command. The parent waits for the child to finish and prints out statistics.

In the shell and shell2 programs, a new process is forked just for runstat(), and then runstat() forks again. This is very convenient for statistics collection, because the getrusage() call in runstat() only returns the statistics for the command that was run by it. Shell and shell2 do not need to keep track of child process statistics because the command process is not a direct child of it. This also makes the source more readable and less error-prone since this one function does not need to be duplicated across three programs.

Part 3 Explanation:

I used a simple linked list structure to keep track of background processes. The format is a struct type bg_proc_t, which has field for the PID of the process and the first command token (the program run), as well as the two adjacent bg_proc_t's in the list. When a background process is started, a bg_proc_t struct is constructed and inserted into the end of the list. There is also a counter counting the number of background processes in the list.

Whenever a command is executed, shell2, goes into a loop with wait3() to get all finished processes. When an exited PID is received from wait3(), it is simple removed from the linked list of background processes. The statistics do not need to be printed out, because the runstat() function would have done it as soon as the process finished. When a non-background process is run, the shell blocks while it completes first before checking for finished background tasks. On exit, the background process counter is checked to see if there are still background processes, and they program goes into a wait3() loop until all processes have completed.