CMPUT 379 Assgn. 1 **Dhruba Chowdhury**
CMPUT 379
StudentID: 1466117
CCID: konishky

Objectives

This assignment was intended to formalize UNIX programming by utilizing function calls that manage processes. This approach to low level programming allowed for a greater appreciation for the C language, and gave a general overview of the process life cycle.

Design Overview

- a1jobs
  - Utilized a struct known as jobInfo which holds information pertaining to a job (command, index, and pid)
  - Use of memory allocation to create an array of struct jobInfo
  - created functions to parse arguments when the user enters ”run prgm arg1....”
  - Use of execvp instead of execlp to since it was easier to utilize an array or arguments
  - Use of the getline function to ease reading from stdin - with the tradeoff of extra mem allocation
  - Use of the APUE [3/E]. In particular I used figure 8.31 to utilize the times function
- a1mon
  - Used recursion to set a list of childs
  - Use of a struct known as childInfo that stores information related to a child of a parent process (command, pid, ppid)
  - Use of the popen listed in the assignment description to display the requried stdout
  - Used other ps commands to minimize parsing with the cost of more popen and pclose
  - Used the command ”ps -o ppid,pid,cmd –ppid pid” in order to grab all children of a process by easing the process of string tokenization
  - Use of the ”ps pid”command to monitor to see if the process is terminated
    * If the process is terminated, then the process would not be listed using the command (contributing to only one row in the terminal - ”PID TT STAT TIME COMMAND ”)
    * I returned a bool to check if the number rows from the ouput are greater than 1 (if false then the process has been killed, else otherwise).

Project Status

The project is completed, though, I did find one issue which arises when a1jobs is called with the standard list of instructions as mentioned in the assignment. The program a1mon works perfectly when targeting the pid of a1jobs. However, when the the pid of a ”myclock” program that was running through a1jobs is targeted and killed, a1mon does not detect that the process has been killed and thus still continues with a zombie sleep process (from myclock). It was an unusual occurrence since when ”myclock outfile” was run outside of a1jobs, a1mon successfully monitored its termination and took action in accordance to the assignment specifications. I believe that this was due to the implementation I used to monitor the state of the process.”Ps pid” does not take into account when zombies occur. When called, it would list the zombie effectively making row count greater than 1, which, according to my implementation of the monitor process function, returns a false bool which states that the process has not been killed - which is wrong.

Testing and Results

Implemented unit tests in my functions. Whenever I wrote some code, I always tested it with some ”dummy” input such that I would get the desired output. This was particularly used in the implementation of the strtok functions. As I kept deviating from my desired results, I had to re-factor my implementation in order to get the desired output.

Acknowledgements

APUE 3/E Ta office hours
Lab sessions
Eclass Discussion forum
C Programming: A Modern Approach, 2nd Edition - K.N.King