

Assignment 1

Creating and Destroying a Process Hierarchy

Due Feb 15th midnight

Objectives

- Learn how to create and terminate processes.
- Learn about process hierarchy.
- Learning to use system calls `fork()`, `exec()`, `wait()`, `getpid()`, `getppid()`
- Practice using command-line arguments and recursion.

Description

You are being asked to write a program that will recursively create a process hierarchy tree that is H levels deep, print out the process tree information as the tree is being created, and then systematically terminate all the processes in the tree. Here are the detailed requirements:

1. Your program should accept two command-line arguments **H** and **C** that describe the structure of the process tree. The argument H is the height of the tree and C is the number of child processes belonging to each internal node of the process tree.
2. Upon starting up, your program should first print the following information:

```
(pid): Process starting
(pid): Parent's id = (ppid)
(pid): Height in the tree = (value_of_H_from_command_line_argument)
(pid): Creating (value_of_C_from_command_line) children at height (H-1)
```

In the above output, you should replace `pid` by the process id of the current process and `ppid` by the process id of the parent process.

3. Next, if the height H is greater than 1, your program should create C child processes using **fork()**, and wait for all of the children to complete using the **wait()** system call.
4. Once all the child processes (if any) have terminated, your program should quit by printing.

```
(pid): Terminating at height (H).
```

Parent must not quit before ALL child processes terminate.

Also, parent must call `wait()` ONLY AFTER CREATING ALL CHILDREN. (Think why?)

5. What should each child process do while the parent waits? Recursion! Each of the child processes should use the **exec()** system call to run exactly the same program image as the parent. The only difference should be that the command-line argument received by the child processes from the parent (via the `exec()` system call) should be **H-1** for height and **C** for number of children.
(NOTE: Recursion in this step can also be done without using `exec()`, via straightforward function calls. But you are REQUIRED to use `exec()` to start the child program image and pass arguments to it. Learning about `exec()` is one of the goals of this assignment.)
6. Make the output more readable by neatly indenting the print statements above to match height of each process in the process hierarchy. (Do this at the end. Its worth only 5 points, but makes the TA's life easier while grading).

Grading Guidelines

This is how we will grade your assignment during the demo. So please prioritize your work accordingly.

5 - README, Makefile, Compilation without errors

5 - Creating and terminating a single process ($H = 1$, $C = 1$) and printing the correct output

5 - Correct use of `getpid()` and `getppid()`

40 - Correctly creating and terminating a hierarchy of processes with arbitrary H and C using `fork()`, `exec()`, and `wait()`

10 - Demonstrating that recursion is implemented using `exec()` (and NOT using usual function calls)

10 - Demonstrating that a parent process `wait()`s for child processes ONLY after it creates ALL child processes.

10 - Demonstrating that a parent process `wait()`s for ALL child processes before terminating itself.

20 - Explaining the code during demo.

You will be asked to explain the logic of the code you wrote.

10 - Error Handling

Most important part here is to make sure that you check and handle the errors returned by ALL systems calls used in your program. Also check for other common error conditions in your program. But don't go overboard with error checking. We will NOT try to intentionally break your code with bad input that's irrelevant to the assignment's goal.

5 - Indented formatting for output of processes at different heights in the hierarchy

Total points = 120

Hints

1. The entire program is likely very small, no more than a few tens of lines of code at most, if you think through it carefully.
2. Use manpages to check the usage details of different system calls.
3. Do this assignment as a regular user, NOT as root, to avoid corrupting your system accidentally.
4. Use the command "kill -9 (pid)" to kill one process. Use "killall (program_name)" to kill all processes starting with a certain name. Use "kill -9 -1" to kill all processes owned by the current user (do this ONLY as a normal user, NEVER as root).
5. Remember to terminate **all** the processes you create. If implemented incorrectly, you can easily end up writing a program that forks indefinitely. That's called a "fork bomb", and you don't want to handle cleaning up the mess.
6. If at all possible, try NOT to do this assignment on bingsuns machines. If you do something slightly wrong, you can end up with a fork bomb on the bingsuns server. The system administrators there won't like it and they will automatically disable your account. It will be a royal pain for you to get it reactivated. (This has happened before). Use a regular user account in your own virtual system image in the G-7 lab, so that you have root access to kill your processes, or to reboot the machine, if needed.