National University of Singapore
School of Computing

CS2106 # Laboratory 4 Semester 1 11/12

Deadline

16 September, 2011, Friday, 11:59pm.

Platform

This exercise should be done on a Linux machines in the OS Lab.

If you want to use your own Linux machines to work on the assignment, make sure you have the GNU Readline library installed.

If you want to use sunfire to work on the assignment, make sure to test it on a Linux machine in the OS Lab before submission (the OS and system calls have slightly different behaviour). Further, to compile on sunfire, you need to use a different Makefile (`Makefile.sunfire`). Rename `Makefile.sunfire` to `Makefile` then run `make` as usual.

Submission

Your working directory should be named `lab04-A000000X` where `A000000X` is your matriculation number. Create a tar ball named `lab04-A000000X.tar` containing your version of the source code by:

```
tar cvf lab04-A000000X.tar lab04-A000000X/*
```

and submit the tar file into the IVLE workbin named Lab 4.

Tips

`man` and Google are your best friends. Use them effectively to learn about the functions and system calls used in this lab exercise.

Start early!

Marks

This exercise is worth 10 marks.

Marks are deducted for the following:

- 3 marks for not naming your file properly.
- 0.1 marks for every minute late after deadline.

After this lab exercise, you should be more familiar with how to use signals and and the process management system calls fork(), waitpid(), and execve() in a program.

Your task in this lab exercise is to implement your own simple little shell called `bush`.

The shell prints a prompt for user to enter the name of a command and its arguments. Upon receiving the command, the shell looks for the command at the system's standard location (`/bin` and `/usr/bin`) and executes the command with the given arguments.

If the last argument is the "`&`" character, the command is executed in the *background*, i.e., the shell returns to the prompt immediately without waiting for the command to complete. Otherwise, the shell waits until the command has finished execution before it returns to the prompt and waits for the next command. In the latter case, we say that the command is executed in the *foreground*. Note that there must be a white space before `&`.

User may enter a special command `exit` or the EOF character (Control-D) at the prompt, upon which the shell exits normally.

A code skeleton has been given to you. You may download it from the URL `http://www.comp.nus.edu.sg/~cs2106/lab04-A000000X.tar`. Untar the downloaded file using the command:

```
tar xvf lab04-A000000X.tar
```

A subdirectory `lab04-A000000X` is now created. As usual, rename the directory by replacing the string "A000000X" with your matriculation number.

Inside the directory are a `Makefile` and a skeleton code, `bush.c`. You can compile the code using the `Makefile` provided.

Skeleton Code

Read `bush.c` and understand the code provided. Note that the code uses the following functions:

readline(). `bush.c` uses the GNU readline library to simplify reading of commands and arguments from the prompt. To use readline, we have included the header file `readline/readline.h` and linked with the library `libreadline` during compilation.

string functions. We use two string functions in `bush.c`. `strcmp()` compares two strings to see if they are equal, while `strtok()` tokenizes a string into substrings.

Your Tasks

Your tasks in this exercise are:

(a) (2 points) Create a child process to execute the command with the given arguments. You should use `execvp` for this purpose. If `execvp` encounters an error (for instance, if a command cannot be found), you should use the `perror` function to print out a message.

(b) (2 points) If the command is to be executed in the foreground (the variable `run_in_bg` is false), wait for the command to finish execution before return to the prompt and prompt the user for the next command. Otherwise, return to the prompt immediately.

(c) (3 points) Make sure that the child process's entry in the process table is removed after the child process exits. Details on how to do this will be explained in the section "Preventing Zombies"

(d) (3 points) Make sure that when `bush` exits normally, all child processes are killed. Details on how to do this will be explained in the section "No Child Left Behind".

Preventing Zombies

To prevent child processes running in the background from turning into zombies, the the parent process needs to call `waitpid` *asynchronously* after the child process has exited. Such asynchronous execution can be accomplished with signals – the child sends the signal `SIGCHLD` to the parent process when the child exits. The parent process can catch the `SIGCHLD` signal and clean up by calling `waitpid` *after* the child process has exited.

There is one more complication. What if there are multiple child processes running in the background, and not all have exited? The exit of one child causes the signal handler to call `waitpid`, which causes the parent to block waiting for other child processes to finish. To prevent the parent from blocking when multiple child processes are running, the parent should call `waitpid` with `WNOHANG` as its third argument. the `WNOHANG` flag causes the caller to continue without blocking. In other words, calling `waitpid` cleans up the process table for any child process that has exited, but will not wait for child processes that are still running.

In summary, you should write and install a signal handler for `SIGCHLD` that calls `waitpid` with `WNOHANG` as an option to prevent the background child processes from becoming zombies.

No Child Left Behind

A perculiar behaviour for `bush` is its "no child left behind" policy. When the shell exits normally, its child processes running in the background have to terminate too.

A simple way to achieve this is to send the signal `SIGHUP`, which is a signal used to indicate the termination of the controlling process, to the process group lead by `bush`.

There is one more minor detail we need to take care of. If the parent process sends a `SIGHUP` signal to the process group, it will receive the signal too, causing it to terminate itself! To prevent such suicidal result, the parent process should ignore the `SIGHUP` signal by calling `signal`. Beware that `signal` needs to be called at the right moment to allow the parent process to response to normal `SIGHUP` signal.

Missing Features

`bush` is a little brother of `bash` – many functionalities are not available. Notably, there are no builtin commands except for `exit`. Commands such as `cd`, `pwd` etc. does not work. There is no `redirection` nor `pipe`. You cannot run `bush` recursively (i.e., execute `bush` within `bush`). Certain commands that deal with managing terminals would not work correctly (e.g., `vim &` may not work). You may also see garbled screens when you run a command in the background `ls -l &` due to race conditions.

THE END