CS3103: Operating Systems (Spring 2019)

Programming Assignment 1
P1: A Process Manager (PMan)

1 Goals

This assignment is designed to help you:

1. get familiar with Linux system,
2. get familiar with C++ programming,
3. get familiar with system calls related to process management.

You are required to implement your solution in C++ (other languages are not allowed). Your work will be tested on the Linux server `cs3103-01.cs.cityu.edu.hk`.

Be sure to study the `man` pages for the various systems calls and functions suggested in this assignment. The system calls are in Section 2 of the `man` pages, so you should type (for example):

```
$ man 2 waitpid
```

2 Requirements

2.1 Prompt for user input

Your PMan needs to show the prompt for user input. For example, when you run PMan by type in

```
$ ./PMan
```

It prompts

```
PMan:  >
```

for user input.

2.2 Background Execution of a Program

PMan allows a program to be started in the background—that is, the program is running, but PMan continues to accept input from the user. You will implement a simplified version of background execution that supports executing processes in the background.

If the user types: `bg foo`, PMan will start the program `foo` in the background and print out the process id (*pid*) of `foo`. That is, the program `foo` will execute, and PMan will first print out the *pid* of foo and then continue to execute and give the prompt to accept more commands.

To simplify your task, PMan only allows one program running in the background. For instance, if the background process `foo` is still running, PMan will not accept any further `bg` command. In other words, if there is a background process is still running, when the user types `bg test2`, PMan should print `There is a background process still running.`

Your PMan needs to support the following commands:

1. The command `bgkill` *pid* will send the `TERM` signal to the background process with process ID *pid* to terminate that process.

2. The command `bgstop` *pid* will send the `STOP` signal to the background process with process ID *pid* to stop (temporarily) that process.

3. The command `bgstart` *pid* will send the `CONT` signal to the background process with process ID *pid* to **re**-start that process (which has been previously stopped).

4. The command `exit` will execute `bgkill` *pid*, if there is any background process, and then exit.

Note: (1) If *pid* is invalid, PMan should print `The pid is not a valid process id of a background process.` (2) After you use `bgkill` to terminate the background process, PMan should able to run `bg` again because there is no background process running.

See the `man` page for the `kill()` system call for details.

To summarize, your PMan should support the following commands: `bg`, `bgkill`, `bgstop`, `bgstart`, and `exit`. If the user types an unrecognized command, an error message is given by PMan, e.g.,

```
PMan:> ttest
PMan:> ttest:  command not found
```

3 Odds and Ends

3.1 Implementation Hints

1. Use `fork()` and `execvp()` so that the parent process accepts user input and the child process executes the background process.

2. Use a variable `bpid` to record the pid of the background process, and use a variable `status` to record the status of the background process (i.e., terminated, stopped, running).

3.2 Helper programs

demo.cpp:

1. This demo program can be used to act as a background process for testing your PMan as its execution can be visualized by displaying a word every few seconds.
2. This program takes three arguments, word, interval, and times.
3. The first argument word is a single word to be displayed repeatedly.
4. The second argument interval is the number of seconds between two consecutive displays of the word.
5. The third argument times is the number of times the word to be displayed.
6. For example, the following command displays the word running 10 times in 2-second interval.

   ```
   PMan:> bg demo running 2 10
   ```

args.cpp:

1. This example program shows how to display a list of all arguments passed to it.
2. To compile the program, use the following command:

   ```
   cs3103-01:/home/lec/vlee> g++ args.cpp -lreadline -lhistory -o args
   ```

3.3 Warning

Since you will use `fork()` in your assignment, it is important that you do not create a `fork()` bomb, which easily eats up all the pid resources allocated to you.

If this happens, you can try to use the command "kill" to terminate your processes (http://cslab.cs.cityu.edu.hk/supports/unix-startup-guide). However, if you cannot log into your account any more, you need to ask CSLab for help to kill your processes.

3.4 Code Quality

We cannot specify completely the coding style that we would like to see but it includes the following:

1. Proper decomposition of a program into subroutines (and multiple source code files when necessary)—A 500 line program as a single routine won't suffice.
2. Comment—judiciously, but not profusely. Comments also serve to help a marker, in addition to yourself. To further elaborate:
   (a) Your favorite quote from Star Wars or Douglas Adams' Hitch-hiker's Guide to the Galaxy does not count as comments. In fact, they simply count as anti-comments, and will result in a loss of marks.
   (b) Comment your code in English. It is the official language of this university.
3. Proper variable names—`leia` is not a good variable name, it never was and never will be.

4. Small number of global variables, if any. Most programs need a very small number of global variables, if any. (If you have a global variable named `temp`, think again.)

5. **The return values from all system calls and function calls listed in the assignment specification should be checked and all values should be dealt with appropriately.**

4 Marking

Your program will be tested on our CSLab Linux servers (cs3103-01, cs3103-02, cs3103-03). You should tell TA how to compile and run your code in your text file. TAs are not supposed to fix the bugs, either in your source code or in your make file. If an executable file cannot be generated and running successfully on our Linux servers, it will be considered as unsuccessful.

Table 1: Marking scheme.

Components	Weight
`bg`	20%
`bgstop`	20%
`bgstart`	20%
`bgkill`	20%
`exit`	5%
`Error handling`	5%
`programming style and in-program comments`	10%

5 Submission

1. This assignment is to be done individually or by a group of two students. You are encouraged to discuss the high-level design of your solution with your classmates but you must implement the program on your own. Academic dishonesty such as copying another students work or allowing another student to copy your work, is regarded as a serious academic offence.

2. Each submission consists of two files: a source program file (.cpp file) and a text file containing all possible outputs produced by your program (.txt file).

3. Use your student ID(s) to name your submitted files, such as 5xxxxxxx.cpp, 5xxxxxxx.txt for individual submission, or 5xxxxxxx_5yyyyyyy.cpp, 5xxxxxxx_5yyyyyyyy.txt for group submission. Only ONE submission is required for each group.

4. Submit the files to Canvas.

5. The deadline is 10:00 a.m., 21-FEB-19 (Thursday). No late submission will be accepted.

Question?
Contact Miss LIANG, Yu at yliang22-c@my.cityu.edu.hk or your course lecturer

---

The End
