Goals

By completing this assignment, you will know how to develop a program in the Linux platform and to make system calls related to process management.

Introduction

You need to implement a C/C++ program called `BP` that allows user to run **multiple processes (up to three *at the same time*)** at the background. When three processes are currently running, further execution request will be pended (the process state is changed to “stopped”) and wait until **another** process is stopped or terminated. While processes are running at the background, the user can input command to display information of the background processes, stop or kill a background process.

Requirements

1. Your BP needs to show a prompt for user input as follows.

```
$ ./BP
BP >
```

2. `BP` accepts the following commands from the user and takes the corresponding action.

```
BP >bg [name of executable file] [a list of arguments]
```

Action: `BP` runs the executable file with a list of arguments at the background and continues to accept input from the user. If there are already 3 running processes, the process is stopped.

Example: `BP` runs the executable file `demo1` with a list of [arguments]: `running 2 5` at the background and continues to accept input from the user.

```
BP >bg demo1 running 2 5
BP >
```

```
BP >bglist
```

Action: Display the process id(s), name(s) and the state(s) of ALL background processes.

Example:

```
BP >bglist
16529: demo1(running)
16605: demo2(stopped)
16613: demo3(terminated)
```

```
BP >bgstop [pid]
```

Action: Stop the process with process id `pid` and display a message. If there exists *__another__* stopped process *(e.g. stopped earlier because there’re already 4 running processes)*, the earliest process in stopped state *(creation order, not runtime order)* should be automatically restarted.

Example:

```
BP >bgstop 16529
16529 stopped
16624 automatically restarted
```

```
BP >bgkill [pid]
```

Action: Terminate the process with process id `pid` and display a message. Similar to `bgstop`, if there exists *another* stopped process, the earliest process *(creation order, not runtime order)* in stopped state should be automatically restarted.
Example:

```
BP >bgkill 16529
16529 killed
```

```
BP >exit
```

Action: `BP` executes `bgkill` to terminate all background processes, if any, and exits.
Example:

```
BP >exit
16605 killed
16607 killed
$
```

3. BP should display a message after a background process has completed.
   Example:
   ```
   16529 completed
   ```

4. You may assume that the syntax of the input commands and pids are valid, but BP needs to handle redundant commands (e.g. `bgkill` a process which is already terminated) by displaying a message.
   Examples:
   ```
   16529 already stopped
   16529 already terminated
   16529 does not exist
   ```

Hints

- Use `fork()` and `execvp()` so that the parent process accepts user input and the child process executes the background process.
- When you use `fork()`, it is important that you do not create a *fork bomb*, which easily eats up all the 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.
- Use `waitpid()` with an option `WNOHANG` to check if a background process has completed.
- Use `kill()` to send a signal to a process, e.g., a `SIGTERM` signal to terminate a process. Do note that `kill()` can also be used to stop / resume a process, regardless of it's name.
- Study the man pages of the system calls used in your program. For example, the following command displays the man pages of `kill()` in Section 2.
  ```
  $ man 2 kill
  ```

Helper programs

demo.cpp

- This demo program can be used to act as a background process for testing your `BP` as its execution can be visualized by displaying a word every few seconds a number of times.
- This program takes three arguments, `word`, `interval` and `times`.
- The first argument `word` is a single word to be displayed repeatedly.

- The second argument `interval` is the number of seconds between two consecutive displays of the word.
- The third argument `times` is the number of times the word to be displayed.
- For example, the following command displays the word “running” 5 times in 2-second interval.

```
$ demo running 2 5
```

args.cpp

- This example program shows how to read a line from terminal, as well as parsing (cutting) the string using the `strtok()` function.
- To compile the program, use the following command.

```
$ g++ args.cpp -lreadline -o args
```

Marking

- You program will be tested on our CSLab Linux servers (cs3103-01, cs3103-02, cs3103-03). You should describe clearly how to compile and run your program in the text file. **If an executable file cannot be generated and running successfully on our Linux servers, it will be considered as unsuccessful.**
- Marking scheme (total: 100%):

o `bg`	20%
o `bglist`	10%
o `bgstop`	10%
o `bgkill`	10%
o `exit`	10%
o Correctly having 3 processes running at the same time (not 2 or 4…)	10%
o Correctly restart another process upon stop/termination	10%
o Handling of redundant calls (e.g. stop a stopped process)	10%
o programming style and in-program comments	10%

Submission

- 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 student’s work or allowing another student to copy your work, is regarded as a serious academic offence.
- Each submission consists of two files: a source program file (.cpp file) and a text file containing user guide, if necessary, and all possible outputs produced by your program (.txt file).
- Write down your name(s), eid(s) and student ID(s) in the first few lines of your program as comment.
- Use your student ID(s) to name your submitted files, such as 5xxxxxxx.cpp, 5xxxxxxx.txt for individual submission, or 5xxxxxxx_5yyyyyyy.cpp, 5xxxxxxx_5yyyyyyy.txt for group submission. Only ONE submission is required for each group.
- Submit the files to Canvas.
- The deadline is **11:00am, 28-FEB-2020 (Friday)**. No late submission will be accepted.

Questions?

- This is not a programming course. You are encouraged to debug the program on your own first.
- If you have any questions, please submit your questions to Dr. Kenneth Lee via the Discussion board "Programming Assignment #1".
- To avoid possible plagiarism, do not post your source code on the Discussion board.
- If necessary, you may also contact Dr. Kenneth Lee at kenkclee@cityu.edu.hk.