CMPUT 379 - Assignment #1 (10%)

A Simple Shell Program (first draft)

Due: Monday, October 2, 2017, 09:00 PM
(electronic submission)

Objectives

This programming assignment is intended to give you experience in using Unix function calls that manage processes (e.g., fork(), waitpid(), and execl()).

Program Specifications

Shell programs like Bourne shell (sh), C shell (csh), Bash, and Korn shell (ksh) provide powerful interactive programming environments that allow users to utilize many of the services provided by complex multiprocessing operating systems. In this assignment, you are asked to write a simple shell program, called `"a1shell"`, in C/C++ that runs the commands described below. To implement your shell program, you need to familiarize yourself with some Unix functions including the functions mentioned in the following table.

function	Reference in Advanced Programming in the Unix Environment [SR 3/E]
Time values	Section 1.10
umask()	Section 4.8 (see also, Fig. 4.11 in Sec. 4.9, and the summary of file access permission bits in Sec. 4.25)
chdir() and getcwd()	Section 4.23
Time and date routines	Section 6.10
getenv()	Section 7.9
getrlimit() and setrlimit()	Section 7.11
fork(), waitpid(), execl()	Chapter 8: Process Control
times()	Section 8.17
kill()	Section 10.9

The `a1shell` program is invoked by the command line: `"a1shell interval"` where `interval` is an integer number of seconds. After invocation, the program

1. Uses `setrlimit()` to set a limit on its CPU time (e.g., 10 minutes). The goal is to provide some safeguard against a buggy process that may run forever.

2. Forks a child process, referred to as the *a1monitor* process in the explanation given below.

3. Runs the parent process, referred to as the *a1shell* process (explained below) until the user quits the program.

The a1monitor Process

This process gets the interval argument specified on the command line, and runs in a loop until the parent terminates. Each iteration of the loop displays on the stdout the following system status information:

a) The current date and time,

b) The average system load for the last 1, 5, and 15 minutes, and

c) The number of running processes, and the total number of processes in the system.

Use an output format of your choice. For example, you may display the output as

```
a1monitor:  Mon, Sep 18, 2017 12:00:00 PM
            Load average:  1.09, 1.08, 1.03
            Processes:  2/291
```

After displaying the information, the process sleeps for the specified interval of seconds. A new iteration begins when the process wakes up.

To obtain and format the current date and time, you should use suitable system calls from Section 6.10. To obtain information for (b) and (c), the process should read file ``/proc/loadavg'' maintained by the kernel.

The a1shell Process

When started, a1shell prints a prompt "a1shell% " to the stdout, and waits for the user to enter a command line on the stdin (at most 80 characters). After executing each command, the shell prints its prompt string again and waits for the next command line. The shell processes each command as described below.

1. cd pathname: Change the current working directory to the directory specified by the absolute or relative pathname. The specified pathname may begin with a shell environment variable that needs expansion. For example, the pathname "$HOME/c379" requires the expansion of environment variable HOME (the '$' is not part of the variable name). Implement this command using function chdir() (see the above table).

2. pwd: Print the current working directory. Implement this command using function getcwd() (see the above table).

3. umask: Print the current file mode creation mask in **octal** (see function umask() above). In addition, print in octal the value of the constants S_IRWXU, S_IRWXG, S_IRWXO (see Fig. 4.11 in [SR 3/E]).

4. done: Exit the a1shell process.

5. cmd arg1 arg2 ...: If cmd is not one of the above keywords then a1shell should try to execute cmd by passing the entire command line to a child process that executes Bash (program "/bin/bash" on lab machines). In addition, a1shell should report the real time, user CPU time, and system CPU time required to execute cmd. More specifically, a1shell should perform the following steps:

(a) Call function `times()` (see the table above) to record the user and CPU times for the current process (and its terminated children).

(b) Call `fork()` to create a child process. `a1shell` process should then call `waitpid()` to wait for the child process to terminate.

(c) The child process should overlay itself with `Bash` by executing:

```
execl("/bin/bash", "bash", "-c",
      "cmd arg1 arg2 ...", (char *) 0);
```

(d) Upon termination of the child process, process `a1shell` should resume execution and call function `times()` again to obtain the user and system CPU times for itself and its terminated children.

(e) Using a setup and output format similar to the program in Figure 8.31 of [SR 3/E], `a1shell` should use the timing information recorded in steps (a) and (d) to compute and print the following times in **seconds**:

   i. the total time elapsed between steps (a) and (d),

   ii. the **user** and **system** CPU times spent by `a1shell` in executing steps (b) and (c), and

   iii. the **user** and **system** CPU times spent by the child process in executing step (c).

More Details

1. This is an individual assignment. Do not work in groups.

2. Only standard include files and libraries provided when you compile the program using `gcc` or `g++` should be used.

3. **Important:** you **cannot** use `system()` or `popen()` to implement any of the above functionalities.

4. Although many details about this assignment are given in this description, there are many other design decisions that are left for you to make. In such cases, you should make reasonable design decisions that do not contradict what we have said and do not significantly change the purpose of the assignment. Document such design decisions in your source code, and discuss them in your report. Of course, you may ask questions about this assignment (e.g., in the Discussion Forum) and we may choose to provide more information or provide some clarification. However, the basic requirements of this assignment will not change.

5. When developing and testing your program, **make sure you clean up all processes before you logout of a workstation.** Marks will be deducted for processes left on workstations.

Deliverables

1. All programs should compile and run on Linux lab machines (e.g., ug[00 to 34].cs.ualberta.ca)

2. Make sure your programs compile and run in a fresh directory.

3. Your work (including a Makefile) should be combined into a single tar archive 'submit.tar'.

   (a) Executing 'make' should produce the `a1shell` executable

   (b) Executing 'make clean' should remove unneeded files produced in compilation.

   (c) Executing 'make tar' should produce the 'submit.tar' archive.

   (d) Your code should include suitable internal documentation of the key functions.

   (e) Typeset a project report (e.g., one to three pages either in HTML or PDF) with the following (minimal set of) sections:

   - **Objectives:** state the project objectives and value from your point of view (which may be different from the one mentioned above)
   - **Design Overview:** highlight in point-form the important features of your design
   - **Project Status:** describe the status of your project; mention difficulties encountered in the implementation
   - **Testing and Results:** comment on how you tested your implementation, and discuss the obtained timing results
   - **Acknowledgments:** acknowledge sources of assistance

4. Upload your tar archive using the **Assignment #1 submission/feedback** link on the course's web page. Late submission (through the above link) is available for 24 hours for a penalty of 10%.

5. It is strongly suggested that you **submit early and submit often**. Only your **last successful submission** will be used for grading.

Marking

Roughly speaking, the breakdown of marks is as follows:

15% : successful compilation of a reasonably complete program that is: modular, logically organized, easy to read and understand, and includes error checking after important function calls

05% : ease of managing the project using the makefile

25% : `a1monitor`: correctness of starting the monitor process, printing information specified in (a), (b), and (c), and terminating the process

30% : `a1shell`: correctness of executing the built-in features and commands (`setrlimit`, `pwd`, `cd`, expanding environment variables, and `umask`)

15% : `a1shell`: correctness of executing `"/bin/bash -c commandline"` using fork/execute calls for handling non built-in commands, and reporting the required user and system CPU times

10% : quality of the information provided in the project report