ICS 53, Winter 2021

Assignment 2: A Simple Shell

A shell is a program which allows a user to send commands to the operating system (OS), and allows the OS to respond to the user by printing output to the screen. The shell allows a simple character-oriented interface in which the user types a string of characters (terminated by pressing the Enter(\n)) and the OS responds by printing lines of characters back to the screen.

Typical Shell Interaction

The shell executes the following basic steps in a loop.

1. The shell prints a "prompt>" to indicate that it is waiting for instructions.

prompt>

2. The user types a command, terminated with an <ENTER> character ('\n'). All commands are of the form COMMAND [arg1] [arg2] ... [argn].

prompt> ls

3. The shell executes the chosen command and passes any arguments to the command. The command prints results to the screen. Typical printed output for an ls command is shown below.

hello.c hello testprog.c testprog

Types of commands that your shell must support

There are two types of commands, **built-in commands** which are performed directly by the shell, and **general commands** which indicate compiled programs which the shell should cause to be executed. Your shell will support five built-in commands: jobs, bg, fg, kill, and quit. You shell must also support general commands.

General commands can indicate any compiled executable. We will assume that any compiled executable used as a general command must exist in the current directory. The general command typed at the shell prompt is the name of the compiled executable, just like it would be for a normal shell. For example, to execute an executable called hello the user would type the following at the prompt:

prompt> hello

Built-in commands are to be executed directly by the shell process and general commands should be executed in a child process which is spawned by the shell process using a fork command. Be sure to reap all terminated child processes.

Job Control

A job is a process that is created (forked) from the shell process. Each job is assigned a sequential job ID (JID). Because a job is also a process, each job has an associated process ID (PID). There are three types of job statuses:

- Foreground: When you enter a command in a terminal window, the command occupies that terminal window until it completes. This is a foreground job.

Prompt > hello

- Background: When you enter an ampersand (&) symbol at the end of a command line, the command runs without blocking the terminal window. The shell prompt is displayed immediately after you press Return. This is an example of a background job.

Prompt > hello &

- Stopped: If you press ctrl-Z while a foreground job is executing, the job stops. This job is called a stopped job and it can be restarted later by the receipt of a SIGCONT signal.

Any built-in command is always executed in the foreground. When a command is executed in the foreground, the shell process must wait for the child process to complete. Please note that only one job can run in the foreground while many can run in the background.

Unix shells support the notion of job control, which allows users to move jobs back and forth between background and foreground, and to change the process state (running, stopped, or terminated) of the processes in a job. Typing ctrl-C causes a SIGINT signal to be delivered to the process which is the foreground job. The default action for SIGINT is to terminate the process. Similarly, typing ctrl-Z causes a SIGTSTP signal to be delivered to the process which is the foreground job. The default action for SIGTSTP is to place a process in the stopped state, where it remains until it is awakened by the receipt of a SIGCONT signal.

Built-In Commands

- jobs: List the running and stopped background jobs. Status can be “Running”, “Foreground”, and “Stopped”. Format is following.

[<job_id>] (<pid>) <status> <command_line>
prompt> jobs
[1] (30522) Running hello &
[2] (30527) Stopped sleep

- fg <job_id|pid>: Change a stopped or running background job to a running in the foreground. There can only be one foreground job at a time, so the previous foreground job should be stopped. A user may use either job_id or pid. In case job_id is used, the JID must be preceded by the “%” character.

prompt> fg %1

prompt> fg 30522

- bg <job_id|pid>: Change a stopped background job to a running background job.
- kill <job_id|pid>: Terminate a job by sending it a SIGINT signal. Be sure to reap a terminated process.
- quit: Ends the shell process.

I/O redirection

Your shell must support I/O redirection.

Most command line programs that display their results do so by sending their results to standard output (display). However, before a command is executed, its input and output may be redirected using a special notation interpreted by the shell. To redirect standard output to a file, the ">" character is used like this:

prompt> ls > file_list.txt

In this example, the ls command is executed and the results are written in a file named file_list.txt. Since the output of ls was redirected to the file, no results appear on the display. Each time the command above is repeated, file_list.txt is overwritten from the beginning with the output of the command ls. To redirect standard input from a file instead of the keyboard, the "<" character is used like this:

prompt> sort < file_list.txt

In the example above, we used the sort command to process the contents of file_list.txt. The results are output on the display since the standard output was not redirected. We could redirect standard output to another file like this:

prompt> sort < file_list.txt > sorted_file_list.txt

I/O redirection Authorization

We should add permission bit when we do I/O Redirection. Permission bits control who can read or write the file. https://www.gnu.org/software/libc/manual/html_node/Permission-Bits.html

- **Input redirection to “input.txt” (Read)**
  mode_t mode = S_IRWXU | S_IRWXG | S_IRWXO;
  inFileID = open (“input.txt”, O_RDONLY, mode);
  dup2(inFileID, STDIN_FILENO);
- **Output redirection to “out.txt” (Create or Write)**
  outFileID = open (“out.txt”, O_CREAT|O_WRONLY|O_TRUNC, mode);
  dup2(outFileID, STDOUT_FILENO);

Submission Instructions

Your source code should be a single c file named 'hw2.c'. Submissions will be done through Gradescope. You have already been added on the Gradescope course for ICS53. Please login to with your school (UCI) email to access it. Please remember that each C program should compile and execute properly on openlab.ics.uci.edu when it is compiled using the gcc compiler version 4.8.5. The only compiler switch which will be used is -o (to change the name of the executable).

Specific directions

- Headers:
  stdio.h, string.h, unistd.h, stdlib.h, sys/stat.h, sys/types.h, sys/wait.h, ctype.h, signal.h, fcntl.h

* As long as you can compile it on openlab (gcc 4.8.5) without any additional compiler flags, you can use any headers other than those specified.

- MaxLine: 80, MaxArgc: 80, MaxJob: 5
- Use both execvp() and execv() to allow either case.
  execvp() : Linux commands {ls, cat, sort, ./hellp, ./slp}.
  execv() : Linux commands {/bin/ls, /bin/cat, /bin/sort, hello, slp}.

Example 1

1. Create 2 programs, add.c and counter.c, and compile them.
   < add.c >

```
int main(int argc, char * argv[]){
   int n = atoi(argv[1]);
   printf("%d \n", n+2);  // print n+2
   return 0;
}
```

   <counter.c>

```
int main() {
   unsigned int i = 0;
   while(1)
   {
      printf("Counter: %d\n", i);
      i++;
      sleep(1);
   }
}
```

$gcc add.c -o add

$gcc counter.c -o counter

Now we have compiled executables, “add” and “counter”. ^Z in below == ctrl-Z

```
prompt> add 4
6
prompt> add 4 > out.txt
prompt> cat out.txt
6
prompt> counter
Counter: 0
Counter: 1
^Zprompt> jobs
[1] (2744) Stopped counter
prompt> fg %1
Counter: 2
Counter: 3
Counter: 4
^Zprompt> jobs
[1] (2744) Stopped counter
prompt> fg 2744
Counter: 5
Counter: 6
Counter: 7
^Zprompt> jobs
[1] (2744) Stopped counter
prompt> kill %1
prompt> jobs
prompt> quit
```

Example 2

Prepare two terminals on the same server.

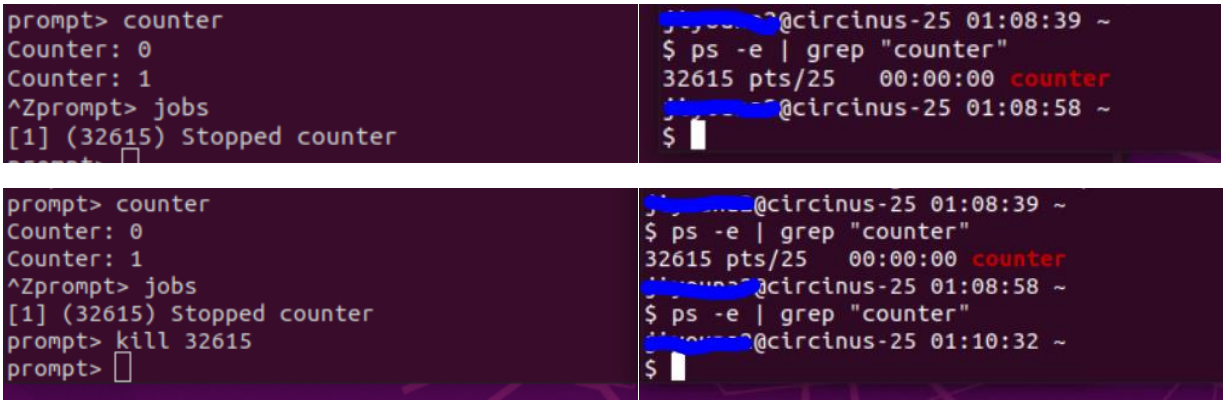
[Terminal1]
ssh <your_id>@openlab.ics.uci.edu
You will see your server name e.g. <your_id>@circinus-14
circinus-14 is your current server name in this case.
[Terminal2]
ssh <your_id>@<server_name>.ics.uci.edu
e.g. ssh <your_id>@circinus-14.ics.uci.edu
In this way you can access to the same server using two terminals.

1. [Terminal1] Run your shell
2. [Terminal1] prompt> counter
3. [Terminal1] ctrl-z
4. [Terminal1] jobs
   [1] (<a_pid>) Stopped counter
5. [Terminal2] ps -e | grep “counter”

<a_pid> pts/0 00:00:01 counter

6. [Terminal1] kill <a_pid>
7. [Terminal1] jobs
   No output
8. [Terminal2] ps -e | grep “counter”
   No output

< Terminal 1 > <Terminal 2 >



If you can still see “<a_pid> pts/0 00:00:01 counter” after kill,
it means that you did not properly terminate a child process.

References

“Understanding the job control commands in Linux – bg, fg and CTRL+Z“, Understanding the job control commands in Linux – bg, fg and CTRL+Z”, accessed Jan 10, https://www.thegeekdiary.com/understanding-the-job-control-commands-in-linux-bg-fg-and-ctrlz/