CSC 209 Assignment 3, Fall 2013: Processes

Due by the end of Friday November 8, 2013; no late assignments without written explanation.

Part 1: A trivial shell

As an initial exercise, you will examine the basic operations needed to execute commands in a shell with execve(). In /u/ajr/209/a3 you will find files named babyshell.c, parse.h, and parse.c. Note how babyshell.c reads an input line in a loop, and parses it using parse() from parse.c. Modify babyshell.c by adding the execute() function and submit only babyshell.c (you don't modify parse.c, and you don't have to understand its insides, just its interface from the outside).

Note that the parsing is entirely trivial; whitespace characters are the only special characters. For example, "cat file" works as expected, but "echo hello; echo goodbye" will not (the semicolon will just be considered to be part of the arguments to 'echo').

The first thing your execute() function has to do is to find the command. If argv[0] does not contain a slash, look for it in /bin, /usr/bin, /usr/local/bin, and the current directory, in that order. (Note: Do not opendir() the directories; just put together the path name and stat() it to see whether it exists. A real shell would then use the returned "struct stat" to check whether it's executable, but we won't do that for this assignment.) If the command is not found in any of these directories, print the usual error message: "%s: Command not found\n".

However, if argv[0] does contain a slash, then it is a path name (absolute or relative) and should be attempted to be executed directly, without searching /bin and such. For example, the user can type /bin/cat, and this doesn't mean /bin/bin/cat, or /usr/bin/bin/cat, or even ./bin/cat—it just means /bin/cat as typed. Also, "./cat" means to run cat in the current directory, even though "/bin/./cat" would be a valid name for /bin/cat. So just like in a real shell, the user can use "./" to evade the search algorithm.

The argv value passed to your execute() function differs from the standard argv parameter to main() in C, in that the argument list is terminated with a NULL pointer; thus you do not need an argc parameter. This is the format which execve() expects. The third parameter to execve() will be the global variable "environ". You can declare it with "extern char **environ;".

When you wait() for the command, assign its exit status to the global variable "laststatus". Since main() returns laststatus, when you reach end-of-file on the input (e.g. you press ^D), the exit status of the shell is the exit status of the last command.

Part 2: The telephone game

The children's game known as the "telephone game" involves whispering a message person-to-person around a circle, and seeing how it gets distorted in the relaying. In this part of this assignment, you will relay messages in a ring amongst different processes.

Your program will be called "ring.c" and it will fork a number of processes connected by pipes—each process is connected to the next, and the last process is connected to the first. That is to say, if there are n processes (the value of n is a command-line argument), then process number i sends its message on to process number $(i+1)\%n$.

Process number zero reads a line from the standard input. It then transmits it to process number one. Each other process reads the line, increments the first byte by one as a distortion, and then relays the line to the next process. As it relays, it prints a status message which looks something like this:

```
process #1 (14279) relaying message: Iello, how are you?
```

so that we can see the message travelling around the ring. When the message gets back to

(over)

process number zero, it is output to the standard output as well.

When a process receives end-of-file (whether from its pipe, in the case of processes other than process zero, or from standard input, in the case of process zero), it prints a message and exits. This process termination causes the closing of the next pipe, so they will all exit.

Other notes

Your programs must be in standard C. They must compile on the CDF machines with "gcc –Wall" with no errors or warning messages, and may not use linux-specific or GNU-specific features.

Submission commands are as in previous assignments and the labs. You need to submit babyshell.c and ring.c. Your babyshell.c will be compiled with the command "gcc –Wall babyshell.c parse.c", with the distributed parse.h and parse.c present in the current directory.

Please see the assignment Q&A web page at

`http://www.cdf.toronto.edu/~ajr/209/a3/qna.html`

for other reminders, and answers to common questions.

And as always, your submission must be your own.