IC221 Lab: Fork+Exec

AY22 Spring

In this lab, you will complete two tasks that require the parent process to fork a child process during execution.

**Compilation**

A Makefile is provided for this lab. Each program has its own target. You can use it as follows:

make timer

make mini-sh

make # both targets

make clean

When compiling mini-sh, you will need to link against the readline library. The compilation command requires the addition of -lreadline at the end:

gcc -g -Wall mini-sh.c -o mini-sh -lreadline

This is specified in the Makefile, but you’ll need to add it if compiling manually.

**Submission**

mini-sh.c

timer.c

Note: if using the test script test.sh, you may want to comment out tests you don’t need yet.

**Part 1: timer (50 points)**

Your task is to complete the program timer.c. The timer program will take the name of *another* program as a command line argument, fork and execute that program, record the amount of time it takes to execute, then print the result.

Use the gettimeofday() API call to retrieve the current time. There is a function provided for computing the difference between two time values. The folowing statement is commented out in the code; uncomment it to print out the execution time in the proper format:

printf("Run Time: %ld.%04ld (s)\n", diff.tv\_sec, diff.tv\_usec/1000);

Sample output. Actual runtimes will vary.

$ ./timer

Run Time: 0.0084 (s)

$ ./timer ls

Makefile timer timer.c

Run Time: 0.0002 (s)

$ ./timer ls -l

total 20

-rw-r----- 1 door scs 89 Feb 17 11:04 Makefile

-rwxr-x--- 1 door scs 10898 Feb 24 17:08 timer

-rw-r----- 1 door scs 1595 Feb 24 17:08 timer.c

Run Time: 0.0003 (s)

$ ./timer sleep 1

Run Time: 1.0000 (s)

$ ./timer sleep 2

Run Time: 2.0000 (s)

Hint: You will need to construct an argv array for exec using the command line arguments to the timer program. The difference between the argv array to timer and the argv array you will need to pass to exec is just one index. For example, consider the argv for this run of the timer:

.-----.

argv -> | .--+--> "./timer"

|-----|

| .--+--> "ls"

|-----|

| .--+--> "-l"

|-----|

| .--+--> NULL

'-----'

You can set the argv that gets sent to exec to start one element later in the array:

.-----.

| .--+--> "./timer"

|-----|

argv+1 -> | .--+--> "ls"

|-----|

| .--+--> "-l"

|-----|

| .--+--> NULL

'-----'

**Part 2: mini-shell (50 points)**

We have all the techniques necessary to implement a basic command shell. A shell is just a fork-exec-wait loop. It prompts the user for input, forks a new process, tries to execute the input as a command with exec, then waits for the command to finish.

For this task, you will write a mini-shell, mini-sh, that will continually prompt the user for a command, execute the command, time the length of execution, and include the time length in the next prompt. It is very much just the timer program, in a loop, except now it's nececssary to build an argv from each round of user input. Here is some sample execution:

$ ./mini-sh

mini-sh (0.0000) #> ls

Makefile mini-sh mini-sh.c

mini-sh (0.0001) #> ls -a -l

total 32

drwxr-x--- 2 door scs 4096 Feb 24 16:32 .

drwxr-x--- 5 door scs 4096 Feb 24 16:31 ..

-rw-r----- 1 door scs 111 Feb 24 16:32 Makefile

-rwxr-x--- 1 door scs 15539 Feb 24 16:32 mini-sh

-rw-r----- 1 door scs 2970 Feb 17 11:04 mini-sh.c

mini-sh (0.0004) #> head -c 10M /dev/zero

mini-sh (0.0807) #> bad command

./mini-sh: No such file or directory

mini-sh (0.0000) #>

In mini-sh.c, the looping, prompt, and argument parsing code are provided. Your task is:

* Execute a fork-exec-wait sequence and record the time of execution in diff.
* Use execvp to execute the input command in a child process
* wait() in the parent before continuing the loop.

Exit Note: Always call \_exit() after an exec command in the child process. The exec call may fail -- if it does, it's important to for the child process to exit immediately.