CS 253: Introduction to Systems Programming
Smash - Part II

March 31, 2020

Learning Objectives

- Debugging memory leaks with `valgrind`
- Specifying functional test cases
- Using fork/exec/wait to run a program in a child process
- Interaction of I/O APIs with fork and exec
- Retrieving a child process's exit status

Overview

This project adds new functionality to smash for executing external commands and recording their exit status in the history. Your shell will begin to actually "do something."

Smash II Features

1. Smash parses a command and its arguments from each line read from stdin. E.g. `ls -l`
2. Smash issues appropriate error messages (similar to bash) for illegal/invalid commands.
3. Smash executes an external command in a child process and retrieves its exit status.
4. The `history` command displays the exit status retrieved from external commands.
5. External commands may reside in any directory identified by the PATH environment variable.
6. External commands inherit stdin, stdout and stderr from smash.

Note: Because this version of smash implements support for external commands, it no longer displays the contents of their argument array to the console as did Smash I.

Not Implemented

The following features are explicitly not included in Smash Part II:

- Support for I/O redirection (e.g. `ls >allMyFiles`)
- Support for pipelined commands (e.g. `ls | wc`)

- Support for wildcards (e.g. `ls *.o`)
- Support for quotation marks (e.g. `ls -l "my file"`) enclosing arguments with embedded spaces
- Support for shell variables (e.g. `$PATH`, `$?`, etc).

Code Health and Memory Leaks

Your code must compile with `-Wall` and no warning messages on onyx. Smash II must pass `valgrind` on onyx without reports of "Definitely lost" memory.

Makefile

The `Makefile` must include targets for `all`, `smash` and `clean`. Additional targets are encouraged but not required nor graded. The `Makefile` must rebuild only the affected binaries when a header or source file is updated.

Procedure

1. Create a new directory called, `p5`, for this version of the smash shell.

2. Copy everything from P4 into your new project directory.

3. Modify the P4 quick test, `testSmashOne`, to comment-out (disable) the test of the parsed external command tokens by placing a `#` symbol at the beginning of each line of the test that appears below the comment labeled, "Smash should be able to parse an external command into tokens..." near lines 351...357. Run the test to confirm that the modified test and the P4 code are still working[1] as expected in the new directory.

4. Implement a new function, `executeExternalCommand()`, invoked from `executeCommand()` where it recognizes it needs to execute an external command.

5. Invoke the `fork()` API in `executeExternalCommand()`. Arrange for the parent to `wait()` for the child's exit status. Arrange for the child to execute the external command using the `execvp()` API. Don't forget to include the NULL pointer following the last argument pointer in the args array.

6. Thoroughly test external commands; they should be functional now.

7. Re-run the P4 regression test to confirm that you didn't break anything that previously worked!

8. Download the P5 quick test (it will be made available on Blackboard) and verify that the most essential features of Smash II are working before beginning Smash III.

9. Run valgrind and repair the problems it uncovers. Rerun the P4 and P5 regression tests to ensure the valgrind repairs didn't introduce a new bug. The P5 grader will run valgrind and check for "definitely lost" allocations – those are the worst kind.

[1] A test to verify previously working features are still working is known as a *regression test*.

Hints and References

While an external command must execute in a child process, internal commands always execute in the parent. Consider the `cd` command. If executed in the child, it would change the directory of the child process, not the smash parent.

Recall that issuing the `fork` API while streams (e.g. stdin, stdout, stderr...) contain buffered data associated with a `FILE` structure may result in unexpected bugs as the child process inherits a copy of its parent's buffers[2]. Buffering problems are extremely difficult to test and analyze. You can avoid many buffering problems in P5 by disabling buffering in stdout and stderr. Here's an example of how to disable buffering of the stdout stream:

```
#include <stdio.h>;
...
int main() {
   setvbuf(stdout,NULL,_IONBF,0);  //Disable buffering in the stdout stream
   ...
}
```

The `execvp` API automatically searches your PATH for the executable program.

You may modify the function prototypes used in earlier versions of smash as you find necessary. The examples in these assignments are merely suggestions that might simplify later versions of smash.

References

- ExampleProcesses/forkAndWait.c illustrates the use of fork and wait
- https://www.geeksforgeeks.org/fork-system-call provides a tutorial covering the use of fork
- https://www.geeksforgeeks.org/wait-system-call-c provides a tutorial covering the use of wait
- https://www.geeksforgeeks.org/exec-family-of-functions-in-c covers use of execvp
- man 3 setvbuf
- man 2 fork
- man 3 execvp
- man 2 wait

The `man` pages for fork, exec and wait include operating system-specific details you may need.

If you are not programming on onyx, be aware that different distros have minor variances in their support for the fork, wait, and exec APIs. When consulting man pages for the APIs, be sure to use those on onyx.

Practice thinking about the corner-cases as well as the "happy path" through the code. Is the history command properly recording non-zero exit status (it may not if you haven't read the `wait`

[2]The Standard I/O stream's buffering implementation varies depending upon whether a stream is writing to a disk file or to the console.

documentation)? What happens if an external command's program does not exist? Will smash crash if the user enters a ridiculously long command line? Can smash handle "lots" of arguments for a command? Is smash working correctly when reading commands from stdin redirected to a disk file as well as from the keyboard? Try something ridiculously simple... like... a blank line.

Submission

Artifacts that must be submitted include:

- Makefile
- smash.c
- Additional source and header files required to build Smash II

The onyx command, `submit username cs253 p5`, will submit the contents of the current working directory to the specified instructor's username.