UVic CSC 360, Assignment 1: kapish?

A UNIX Shell

Objective

To learn about UNIX processes, and the command interpreter.

Background

A UNIX **shell** is a program that makes the facilities of the operating system available to interactive users. There are several popular UNIX shells: *sh* (the Bourne shell), *csh* (the C shell), and *bash* (the Bourne Again shell) are just a few. In this assignment, you will build *kapish*.

Your Task

You need to create a program named *kapish*. *kapish* should be a minimal but realistic **k**iller **a**pplication **i**nteractive UNIX **sh**ell.

Initialization and Termination

When first started, *kapish* should read and interpret lines from the file *.kapishrc* in your HOME directory, provided that the file exists and is readable. Note the the file name is **.kapishrc** (with the leading ".", not kapishrc), and that it resides in the user's **HOME** directory (not the **current** directory). Typically, the *.kapishrc* file contains commands to specify the terminal type and environment.

To facilitate your debugging and our testing, *kapish* should print each line that it reads from .kapishrc immediately after reading it. *kapish* should print a question mark and a space (?) before each line.

kapish should terminate when the user types Control-D or **exit**.

Interactive Operation

After startup processing, *kapish* should read lines from the terminal, prompting with a question mark and a space (*?)*. Specifically, *kapish* repeatedly should perform the these actions:

- Read a line from standard input.
- Lexically analyze the line to form an array of **tokens**.
- Syntactically analyze (i.e. parse) the token array to form **command [options] [args]** typically fed to the command interpreter.
- Execute the **command**.

Lexical Analysis

Informally, a **token** should be a word. More formally, a token should consist of a sequence of non-whitespace characters that is separated from other tokens by whitespace characters. *kapish* should assume that no line of standard input is longer than 512 characters. If a line of standard input is longer than 512 characters, then

kapish need not handle it properly; but it should not corrupt memory.

Execution

If the command is a *kapish* built-in, then *kapish* should execute it directly (i.e. without forking a child process). *kapish* should interpret four shell built-in commands:

setenv *var* [*value*]	If environment variable *var* does not exist, then *kapish* should create it. *kapish* should set the value of *var* to *value*, or to the empty string if *value* is omitted. Note: Initially, *kapish* inherits environment variables from its parent. *kapish* should be able to modify the value of an existing environment variable or create a new environment variable via the setenv command. *kapish* should be able to set the value of any environment variable; but the only environment variables that it explicitly uses are HOME and PATH.
unsetenv *var*	*kapish* should destroy the environment variable *var*.
cd [*dir*]	*kapish* should change *kapish*'s working directory to *dir*, or to the HOME directory if *dir* is omitted.
exit	*kapish* should exit.

Note that those built-in commands should neither read from standard input nor write to standard output.

If the command is not an *kapish* built-in, then *kapish* should consider the command-name to be the name of a file that contains executable binary code. *kapish* should use the PATH environment variable to locate the binary, fork a child process and pass the filename, along with its arguments, to the **execvp** system call. If the attempt to execute the file fails, then *kapish* should print an error message indicating the reason for the failure.

kapish should print its prompt for the next standard input line only when a command has finished executing.

Process Control

All child processes forked by *kapish* should run in the foreground; *kapish* need not support background process control. However, the user must be able to kill the current child processes using Control-C. Control-C should not kill *kapish* itself.

Error Handling

kapish should handle an erroneous line gracefully by rejecting the line and writing a descriptive error message to standard error. *kapish* should handle **all** user errors; it should be impossible for the user's input to cause *kapish* to crash.

Memory Management

kapish should contain no memory leaks. For every call to **malloc** or **calloc**, there should eventually be a call to **free**.

History Mechanism (for extra credit)

kapish should support a history mechanism that includes:

- A **history** built-in command. The history command should print a list of all previously issued commands. Note that the **history** command should write data to standard output.
- The ability to re-execute a previously issued command by typing a prefix of that command preceded by an exclamation point (**!*commandprefix***).

kapish need not support editing of the previously issued command.

Logistics

Develop on linux.csc.uvic.ca.

The expectation is that you will devote substantial effort to creating and running test cases.

You should submit:

- Your source code files.
- A makefile. The first dependency rule should build your entire program, compiling with the -Wall and -Werror options. The makefile should maintain object (.o) files to allow for partial builds.
- A readme file.

Your readme file should contain:

- Your name.
- A description of whatever help (if any) you received from others while doing the assignment, and the names of any individuals with whom you collaborated.
- Optionally, any information that will help us to grade your work in the most favorable light. In particular you should describe all known bugs.

Submit your work electronically via the CourseSpaces.

Grading

Your work will be graded on correctness, understandability, and design. To encourage good coding practices, we will take off points based on warning messages during compilation.