

EE461S - Operating Systems

Project 1: yet another shell (yash)

Due Date: Tuesday September 19th at 23:59

Objective

In this lab you will be introduced to both the command line interface and the Unix programming environment. You will write a command line interpreter known as a **shell** that takes commands from standard input and executes the commands by creating processes.

Features

A standard shell like bash/tcsh/csh etc. has a rich set of features that it supports. We picked a subset of these features for you to implement.

Preparing to Code: First, you should exercise all of these features in a shell like bash. Once you understand how to use them you will get a sense of how you can implement them.

Here is the complete list of features you must implement:

- File redirection
 - with creation of files if they don't exist for output redirection
 - fail command if input redirection (a file) does not exist
 - `<` will replace stdin with the file that is the next token
 - `>` will replace stdout with the file that is the next token
 - `2>` will replace stderr with the file that is the next token
 - A command can have all three (or a subset) of the redirection symbols
- Piping
 - `|` separates two commands
 - The left command will have `stdout` replaced with the input to a pipe
 - The right command will have `stdin` replaced with the output from the same pipe
 - Children within the same pipeline will be in one process group for the pipeline
 - Children within the same pipeline will be started and stopped simultaneously
 - **Only one `|` must be present in each pipeline**
- Signals (`SIGINT`, `SIGTSTP`, `SIGCHLD`)
 - `Ctrl-c` must quit current foreground process (if one exists) and not the shell and should not print the process (unlike bash)

- `Ctrl-z` must send `SIGTSTP` to the current foreground process and should not print the process (unlike `bash`)
- The shell will not be stopped on `SIGTSTP`
- Job control
 - Background jobs using `&`
 - `fg` must send `SIGCONT` to the most recent background or stopped process, print the process to `stdout`, and wait for completion
 - `bg` must send `SIGCONT` to the most recent stopped process, print the process to `stdout` in the jobs format, and not wait for completion (as if `&`)
 - `jobs` will print the job control table similar to `bash`:
 - with a [`<jobnum>`]
 - a `+` or `-` indicating the current job. Which job would be run with an `fg`, is indicated with a `+` and, all others with a `-`
 - a “Stopped” or “Running” indicating the status of the process
 - and finally the original command
 - e.g. [1] - Running `sleep 5 &`
 [2] - Stopped `sleep 5 &`
 [3] + Running `long_running_command | grep > output.txt &`
 - Terminated background jobs will be printed after the newline character sent on `stdin` with a `Done` in place of the `Stopped` or `Running`.
 - A Job is all children within one pipeline as defined above
- Misc
 - Children must inherit the environment from the parent
 - Must search the `PATH` environment variable for every executable
 - All child processes will be dead on exit
 - The prompt must be printed as a `#` (hashtag-sign with a space after it) before accepting user input.

Restrictions on the input

These restrictions will help you simplify the parsing the command line:

- Everything that can be a token (`<`, `>`, `2>`, etc.) will have a space before and after it. Also, any redirections will follow the command after all its args.
- `&` will always be the last token in a line (only one `&` makes sense)
- Each line contains one command or two commands in one pipeline
- Lines will not exceed 2000 characters
- All characters will be ASCII
- `Ctrl-d` will exit the shell

Restrictions on programming environment

- All code will be in C (ANSI, C99, GNU99, etc.)
- Code may only include headers from the operating system and the GNU C stdlib. Use of the `system` library call is not allowed.

- Must create a Makefile, so grader only executes "make" in your unarchived project directory and expects the executable to be named "yash"
- All code will run on GNU/Linux (it will be tested on x86-64)

Submission

You will submit a single file named **yash.tgz** that contains all of files in a folder by the same name (**yash**). Make sure there is a **Makefile** that will build your shell with a single command (make). The executable that your makefile should create must also be called **yash**. If you have any doubts about submission please ask the TAs or Dr.Y. It is important that everybody follows these instructions so we can automate the grading process.