COMPSCI 453/552: Operating Systems

Programming Assignment 1

Shell Project Part 1 (100 points)

1 Setup

All the programming assignments for Linux will be graded under the Fedora Linux operating system. The machine `onyx.boisestate.edu` will be used for submission and grading of programs. Make a directory for the class (say `cs453` or `cs552`). Make subdirectories (say `p1, p2, ...`) underneath the class directory. Make sure the directory is protected from others.

2 Introduction

In this project, we will be extending the functionality of the simple shell example that was discussed in class (`ch2/shell1.c`). We will call our shell "dash" (for Dead Again SHell). We will add the capability to accept arguments to commands and the ability to run background jobs. We will add some basic built-in commands like `exit`, `cd` and `logout`.

We will be adding a job descriptor data structure to keep track of jobs running in the background. Now your shell will assign job numbers to commands started in the background and store them in the job list while they execute. We will also add a new built-in command `jobs`, that allows the user to query the status of background jobs.

There will be more parts to the dash project in the rest of the semester. So design your project so that it is easy to extend.

3 DASH: Specifications

4 The Job List

How do we design a data structure for a job list? We need a data structure that is flexible (that is, can grow as needed but not use much memory when not needed), easy to add/remove jobs, relatively easy to search for a job (although this requirement is not very important for our shell project). A doubly-linked list is a good choice to satisfy our needs. You can also use a Hashed Linked List, which allows for easy traversal combined with fast search.

Your mini shell should support the following functionality:

- **Filename Completion and command history**. (5 points) The GNU `readline` and `ncurses` library allow a program control over the input line. The `readline()` function allows the user to edit the input line, use TAB key for filename completion, the use of $\uparrow, \downarrow, \leftarrow, \rightarrow$ keys to access the history of commands typed in by the user. The following code sample shows the usage of the readline library.

```
/*
 *  Filename: test-readline.c
 *  Compile:  gcc -Wall test-readline.c -lreadline -lncurses
 */
#include <stdio.h>
#include <stdlib.h>
#include <readline/readline.h>
#include <readline/history.h>

int main() {

    char *line;
    char *prompt = "dash>";

    using_history(); /* enable readline history mechanism */
    while ((line=readline(prompt)))
    {
        printf("%s\n",line);
        add_history(line); /* add current line to history list */
        free(line);
    }
    exit(0);
}
```

- **Normal command execution**. (10 points) The mini-shell accepts one command per line with arguments. It should accept at least 2048 arguments to any command. The shell will parse each command that is entered and then execute any valid command typed in with arguments (using the `execvp()` system call). The `execvp()` system call performs a search for the command using the `PATH` environment variable. This will simplify your programming since you do not have to search for the location of the command. You will need to consult the `man` pages for the `fork()`, `exec()`, `wait()` and other related system calls.
- **Handling empty commands**. (5 points) If the user just presses the `Enter` key, then the shell displays another prompt.
- **Handing EOF**. (5 points) The mini-shell terminates normally on receiving the end of input (Under Linux and bash, this would normally be Ctl-D for you to test your mini-shell).
- **Exit/Logout commands**. (5 points) Include two built-in commands `exit` and `logout` (built-in to the shell) that also terminate the shell normally.
- **Change Directory Command**. (10 points) Add a built-in command `cd` to allow an user to change directories. You will need to use the `chdir()` system call. The `cd` command without any arguments should change the working directory to the user's home directory. You can use the system calls `getuid()` and `getpwuid()` to find out the home directory of the user.

- **Prompt**. (5 points) The default prompt for dash may be anything you like. However the shell checks for an environment variable `DASH_PROMPT`. If the environment variable is set, then it uses the value as the prompt. The environment variable would be set using the following commands in the terminal.

  ```
  DASH_PROMPT="myshell>"
  ```

  ```
  export DASH_PROMPT
  ```

  Now any program can inquire the value of the above environment variable using the system call `getenv()`. See the man page for `getenv()` for more details.

- Your mini-shell should return a status of 0 when it terminates normally and a non-zero status otherwise.

- **Show version. (5 points)** The shell has a command line option `-v`, which prints out the subversion id and then the shell exits.

- **Background jobs**. (5 points) Your shell can start a command in the background if an & is the last character on the line. For each background job that is started, it prints the job id (see section on Simple Job Control for job ids) and process id of the background process and the full command after starting the command in the background. After starting a background job, the mini-shell comes back with a prompt for the next command without waiting for the background command to finish. The user should **not** be required to separate the & from the command by a space. For example, the commands `date &`, and `date&` and `date &` (blanks after the ampersand) are both valid.

- **Job List. (10 points)** Use a job list to keep track of background jobs. For each job, we want to keep track of the original command, the process id of the background job, exit status if any and the state of the job (running or done).

- **Simple Job Control. (10 points)** The mini shell should keep track of commands running in the background. Whenever the user starts a job in the background, it should assign and display the job number, the associated process id and the full command (including the ampersand)

  ```
  [n] process-id command
  ```

  It should also display an informative message when a background job gets done. That is, every time the user presses the ENTER key, the shell should report all the background jobs that have finished since the last time the user pressed the ENTER key. The message should be of the following form:

  ```
  [n] Done command
  ```

  where `n` is the job number and command is the actual command that was typed. You should use the `WNOHANG` option with the `wait` system call to determine the status of background jobs.

- **The `jobs` command. (10 points)** Add a new built-in command called `jobs`, that prints out all the background commands that are running or are done but whose status has not yet been reported. Here is a sample output:

```
[1] Running sleep 100 &
[2] Done    sleep 5 &
[3] Running gargantuan &
```

- **How to assign job numbers?** The first job should be assigned the job number 1. Each additional job should be assigned one higher number. If lower numbered jobs finish, then we do not reuse them unless all jobs numbered higher than that number have also finished.

5 Source Code Management

You must use subversion for source code management in this project. Create your subversion repository under your home directory in the folder `svn` as follows.

```
svnadmin create ~/svn
chmod o-rwx,g-rwx ~/svn
```

The `chmod` command protects your repository from other people accessing it directly. You should keep your linked list also in the subversion system. You may keep it as a separate project (especially if you are using the shared library version) or in the same project.

Please submit a working checked out copy. Please also include the log file from your subversion repository. You can do that by using the following command:

```
svn log > svn.log
```

6 Design and Documentation (15 points)

Read the entire specification before designing your shell project. Write down the rationale for your design choices. Here are some issues to decide:

- How are you going to organize your functions or classes? In general, you want each function to be relatively short. Each file could be a collection of related functions or you can even have one function per file. Jot down what functionality will be in each file.
- How do you plan to test the shell? Make a comprehensive list of test cases that you will be trying.
- You must have a `README` file that contains your name, date, assignment number on top. Typical things to include are an explanation of what files contain what, how to build the project and run it, known features and bugs, other personal observations about the assignment etc.

Your README file would typically also contain your design ideas as well as testing information. If you wish, you can have a separate design document. But you should still have a README file as described above.

7 Submitting the Assignment

Please observe the following guidelines.

- The name of your executable must be `dash`. The executable must be found in the top level of your submission folder.
- Any shared library that you have must have a copy at the top level of your submission folder.
- You must have a `Makefile` that the `make` command can use to compile and link your shell.
- You must have a `README` file that contains your name, date, assignment number and observations about the assignment.
- Prepare your directory for submission by removing all executables and object files. You should only have the source code checked out of your repository, README files, Makefiles, test scripts and test files before submitting.
- Place all these files in a directory (say `~/cs453/p1`). Change directory to this directory and execute the following command (on `onyx`) to submit the assignment.

  ```
  submit amit cs453 p1
  ```

  or

  ```
  submit amit cs552 p1
  ```

  This command will pick up all files in the current directory (as well as any subdirectories recursively) and time-stamp them before transferring the combined files to the instructor's account.