

A Tiny Unix Shell (tsh) – Part II

Introduction

This is the second part of the project on process control and signaling. This assignment asks you to complete the part of the shell that parses, interprets and executes the command line. This assignment must be completed *individually*.

Background on Shell Command Lines

A *shell* is an interactive command-line interpreter that runs programs on behalf of the user. A shell repeatedly prints a prompt, waits for a *command line* on `stdin`, and then carries out some action, as directed by the contents of the command line.

The command line is a sequence of ASCII text words delimited by whitespace. The first word in the command line is either the name of a built-in command or the pathname of an executable file. The remaining words are command-line arguments. If the first word is a built-in command, the shell immediately executes the command in the current process. Otherwise, the word is assumed to be the *pathname* of an executable program. In this case, the shell forks a child process, then loads and runs the program in the context of the child. The child processes created as a result of interpreting a single command line are known collectively as a *job*.

If the command line ends with an ampersand "&", then the job runs in the *background*, which means that the shell does not wait for the job to terminate before printing the prompt and awaiting the next command line. Otherwise, the job runs in the *foreground*, which means that the shell waits for the job to terminate before awaiting the next command line. Thus, at any point in time, at most one job can be running in the foreground. However, an arbitrary number of jobs can run in the background.

For example, typing the command line

```
tsh> jobs
```

causes the shell to execute the built-in `jobs` command. Typing the command line

```
tsh> /bin/ls -l -d
```

runs the `ls` program in the foreground. By convention, the shell ensures that when the program begins executing its main routine

```
int main(int argc, char *argv[])
```

the `argc` and `argv` arguments have the following values:

- `argc == 3`,
- `argv[0] == "/bin/ls"`,
- `argv[1] == "-l"`, and
- `argv[2] == "-d"`.

Typing the command line `tsh>/bin/ls -l -d &` runs the `ls` program in the background.

What to do

In the first part of this project, you have implemented one module `joblib.c` for manipulating jobs, and a test driver `tsh.c` for testing your code. Replace your test driver `tsh.c` with the shell driver provided below. You should not need to modify the shell driver.

```
/* File tsh.c */
#include "joblib.h"

char *prompt = "tsh> ";    /* command line prompt */

/*
 * main - The shell's main routine
 */
int main(int argc, char **argv)
{
    char cmdline[MAXLINE];

    /* Initialize the job list */
    initjobs(jobs);
    /* Execute the shell's read/eval loop */
    while (1) {

        /* Emit prompt */
        printf("%s", prompt);
        fflush(stdout);

        /* Read command line */
        if ((fgets(cmdline, MAXLINE, stdin) == NULL) && ferror(stdin))
        {
            printf("fgets error");
            exit(1);
        }

        /* Evaluate the command line */
        eval(cmdline); /* This function you'll need to write */
        fflush(stdout);
    }
    exit(0); /* control never reaches here */
}
```

In a separate file called `eval.c`, implement two new functions:

1. `builtin_cmd`: Recognizes and interprets two built-in commands: `quit` and `jobs`.
2. `eval`: Main routine that interprets and executes the command line; uses `builtin_cmd`.

Specifications for `builtin_cmd`:

- (a) If the user has typed in `quit`, then exit the program immediately (use `exit(0)`);

- (b) Otherwise, if the user has typed it `jobs`, then print the list of jobs and return 1.
- (c) Otherwise, return 0 (not a built-in command).

Specifications for `eval`:

- (a) If the user has requested a built-in command (`quit` or `jobs`) then execute it immediately.
- (b) Otherwise, fork a child process and run the job in the context of the child. If the job is running in the foreground, wait for it to terminate and then return.

Use the following declarations and function definitions:

```
#define MAXARGS      128    /* max args on a command line */

/*
 * builtin_cmd - If the user has typed a built-in command
 *               then execute it immediately.
 */
int builtin_cmd(char **argv)
{
    /* Fill in the code here */

    return 0;    /* not a builtin command */
}

/*
 * eval - Evaluate the command line that the user has just typed in
 */
void eval(char *cmdline)
{
    char *argv[MAXARGS]; /* argv for execv */
    int bg;               /* should the job run in bg or fg? */

    /* Parse command line */
    bg = parseline(cmdline, argv);

    if (argv[0] == NULL)
        return; /* ignore empty lines */

    /* Fill in the code here
     * - if a built-in command, execute immediately and return
     * - otherwise, fork a child
     * - the child executes the command (use execvp)
     * - the parent adds the job to the list jobs
     * - if a FG job, - parent waits for child to terminate
     *                 - parent deletes child from jobs
     */

    return;
}
```

We have provided for you the cumbersome function of parsing the command line. This function is available on tanner in the file `/tmp/parseLine` (no file extension). Please add this function to the `eval.c` module. You will need to update your `makefile` to incorporate the new module(s) (`eval.c`).

Important Notes

1. Programs such as `more`, `pico`, and `vi` do strange things with the terminal settings. Do not run these programs from your shell. Stick with simple text-based programs such as `ls`, `ps`, `echo`, and any text-based user programs.
2. Once you have completed this assignment, you will notice that your shell `tsh` does not correctly execute the `jobs` command. In particular, *background* jobs that have completed their execution will still appear in the list of jobs your shell prints out. That's OK. We will learn how to fix this problem in a future assignment.

Evaluation

Your score will be computed out of a maximum of 50 points based on the following distribution:

40 Correctness.

10 Style points. We expect you to have good comments and to check for errors.

Hand In Instructions

Hand in a printed copy of the following:

1. Source code of each of your files (`tsh.c`, `eval.c`, `eval.h` - if you have one, and `Makefile`). Include `joblist.c` and `joblist.h` only if you have changed it from the previous assignment, and *highlight* the changes. Normally, these files should not need to be modified (unless the project specifications for manipulating jobs change, which is not the case).
2. A sample output of your code.
3. A short README file explaining any bugs and deviations from the assignment specifications.

Make sure you have included your name and your Unix login name in the README file. Leave your source code in your Unix account. Good luck!