

Operating Systems Lab (CS 470):

Lab 1: Create a mini shell using C/C++ programming language.

Overview

The OS command interpreter is the program used by the user to interact with the computer in order to launch and control different programs. In UNIX/Linux like systems the command interpreter is usually called *shell*. It is a user-level program that gives people a command-line interface to launch, suspend, and kill (terminate) other programs. `sh`, `ksh`, `csh`, `tcsh`, and `bash` are all examples of such UNIX/Linux shells used by current operating systems.

The example below illustrates the use of the `cwushell`, the shell that you will create. The example shows a prompt `cwushell>` and the user's next command: `cat Prog.c`. This command displays content of the file `Prog.c` on the terminal using the UNIX/Linux `cat` command.

```
cwushell> cat Prog.c
```

Every shell is structured as a loop that includes the following:

1. print a prompt
2. read a line of input from the user
3. parse the line into the program name, and an array of parameters
4. use the `fork()` system call to spawn a new child process
 - o the child process then uses the `exec()` or some other system call to launch the specified program
 - o the parent process (the shell) uses the `wait()` system call to wait for the child to terminate
5. once the child (i.e. the launched program) finishes, the shell repeats the loop by jumping to 1.

Although most of the commands people type on the prompt are the name of other UNIX/Linux programs (such as `ls` or `cat`), shells recognize some special commands (called internal commands) which are not program names. For example, the `exit` command terminates the shell, and the `cd` command changes the current working directory. Shells directly make system calls to execute these commands, instead of forking child processes to handle them.

Instructions

Write a mini shell program (in C/C++) called `cwushell`. The shell has the following features:

- It recognizes the following internal commands:
 1. `quit [n]` -- terminates the shell, either by calling the `exit()` standard library routine or causing a return from the shell's `main()`. If an argument (`n`) is given, it should be the exit value of the shell's execution. Otherwise, the exit value should be the value returned by the last executed command (or 0 if no commands were executed.)

2. `change_prompt [new_prompt_name] --` will change the current shell prompt to the `new_prompt_name`. The default prompt should be `cwushell>`. Typing `change_prompt` in your shell should restore the default shell prompt.
3. `distro -switch` will print on the screen different cpu related information based on the switch. The different switches to be implemented are:
 - 1) `-v` – will print the distro’s version (see 18.04) , 2) `-n` –will print the distro’s name (see Ubuntu), 3) `-c` – will print the distro’s codename (see bionic)
4. `info -switch --` will print on the screen different memory related information based on the switch. The different switches to be implemented are:
 - 1) `-s` – will print the current memory page size in KB, 2) `-n` –will the currently available page numbers 3) `-p` – will print the number of available processors
5. **all other existing shell commands** (internal and external commands e.g. `ls`, `cat`, `pwd`, etc.) should also be executed by the `cwushell`.

Notes

- Error handling should be considered (see erroneous commands, non-existing switches, etc.).
- For the `cwushell` a help file should be provided, which should be invoked from the shell using the `manual` command.
- For each command a specific help should be invoked if called without a parameter or using `-help` or `-h` switches. This will help the user to see how to use the different commands in the `cwushell`. See for details the Unix/Linux manual pages. The help should be structured/formatted in the same way as used by the man pages.
- The shell has to receive commands until explicitly the exit is invoked. Please study the behavior or current shells such as `bash`/`tcsh`/`ksh`.
- For the implementation requested in 1) -4) **NO system()** function can be considered!
- The software should work in the container defined in the class (see `ubuntu_cwu_2020` image).
- **It is not required (but advised!) to implement the commands using `fork()`!**

Rubric

Task	Points
error handling, help mechanisms, other commands	1 + 1 + 1
distro	3
info	3
quit, change_prompt	0.5 + 0.5