Operating Systems Lab (CS 470):

Lab 1: Create a mini shell under Linux/MacOS 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. On 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 the file *Prog.c* on the terminal using the UNIX `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
    - the child process then uses the `exec`() or some other system call to launch the specified program
    - 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` under Linux/MacOS. The shell reconnizes the following internal commands:

1. `exit  [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. `prompt [new_prompt]` -- will change the current shell prompt to the new_prompt. The default prompt should be cwushell>. Typing `prompt` should restore the default shell prompt.
3. `cpuinfo -switch` will print on the screen different cpu related information based on the switch. The different switches to be implemented are:

   1) -c – will print the cpu clock (e.g. 3.2 GHz), 2) -t –will print the cpu type (e.g. Intel) and

   3) -n – will print the number of cores (e.g. 8).

4. `meminfo -switch --` will print on the screen different memory related information based on the switch. The different switches to be implemented are:

   1) -t – will print the total RAM memory available in the system in bytes, 2) -u –will print the used RAM memory and 3) -c – will print the size of the L2 cache/core in bytes.

5. all other existing shell commands (internal and external Linux 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 (containing all the commands, their syntax and their outcome) 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.
- It is not required to implement the commands using fork()! For 1-4 the system() function can not be used!

Rubric

Task	**Points**
error handling, help mechanisms	1 + 1
cpuinfo	1 + 1 +1
meminfo	1 + 1 +1
exit, prompt	1 + 1