CS 3100 Lab 2: Nano Shell
Command Parsing and Process Creation

In Lab 2, you will create a simple command line interpreter. Command line interpreters are often called shells, particularly on systems that provide more than one and allow users to choose. We will create a shell program named **`nanosh`**, which will contain code (much of which you will write) to carry out a very, very small number of user commands but will also spawn processes to run external commands.

Learning Objectives

- Managing data in an array
- While loops
- If/then/else
- Error handling and reporting using errno and perror()
- Calling system functions to retrieve information from the environment and to set the current working directory
- Using fork(), exec() and wait() to run a second program and wait for it to complete

Assignment

A template .C file is provided for this assignment to suggest a logic flow for the program. Your task is to read input from the keyboard a line at a time, parse that line into commands and arguments, populate an array with each word type and executing commands based on what was typed.

`Makefile` and `nanosh.c` are found on icarus in `/var/classes/cs3100/lab2/lab2.tar` and should be copied to your `~/cs3100/lab2` folder:

```
mkdir ~/cs3100/lab2
scp USER@icarus.cs.weber.edu:/var/classes/cs3100/lab2/lab2.tar .
tar xvf lab2.tar
cp template/nanosh.c .
cp template/Makefile .
```

Edit `nanosh.c` and see what has already been provided. Do not alter `Makefile`. When you feel you are ready for grading, upload just `nanosh.c` to Canvas. Compile using `make,` which will call `gcc` with the correct parameters, rather than executing `gcc` directly. I strongly recommend testing each feature completely as you design and code it before moving on to the next feature. Include error handling and testing as you develop each feature.

Features

`nanosh` is a program that accepts commands and executes those commands, in a continuous command loop. In this loop, `nanosh` prints a prompt (`nanosh: `), allow the user to type a line of input and hit RETURN, then `nanosh` should recognize and execute the command entered. `nanosh` should recognize the following commands:

COMMAND	FUNCTION
`exit`	exits the program (don't laugh…it is important to have)
`pwd`	displays the current working directory on stdout
`cd`	changes the current working directory to the path found in environment variable $HOME
`cd NEWDIR`	changes the current working directory to NEWDIR
`ANYOTHERCOMMAND`	executes ANYOTHERCOMMAND and passes any commandline parameters. Waits until ANYOTHERCOMMAND completes.
`NULL`	If no command (or nothing but whitespace) is entered, ignore the command

The so-called "command loop" does the following (in a continuous loop) as a suggested logic flow:

- Prints the prompt
- Reads a line from the keyboard
- Populates an array of strings called `myArgv` with each element of the array pointing to a word on the commandline. All whitespace (tab, newline or space) is ignored. An integer called `myArgc` contains a count of the number of strings in `myArgv`.
- The first word on the line is compared to one of the known internal commands: `exit`, `cd`, `chdir`, `pwd`
- If the first word is one of the known commands, a function written for that command is called, passing it `myArgc` and `myArgv`. The called function checks for parameter errors and executes the requested function, then returns to the loop.
- If the first word is not a known command, call a function that will `fork()` and `exec()` a new process. The child process should `execvp()` the requested program and pass `myArgv`. The parent process should issue `waitpid()` and then return.
- returns to the top of the loop

Error handling:

- If `exit`, `pwd` or `cd` are entered with an invalid number of parameters, nanosh should set errno to `EINVAL`, call `perror()` with an appropriate message, ignore the command and restart the loop
- If one of the commands fail, call `perror()` with an appropriate message and restart the loop.
- If ANYOTHERCOMMAND is entered and that command cannot be found on the system (meaning `execvp()` fails, call `perror()` with an appropriate message.
- If a command is successful, do not issue any message whatsoever. Simply restart the command loop. Issue messages only when an error occurs.

Example output

Here is an example of how your `nanosh` should behave:

```
tcowan@tcowan-Ubuntu:~/cs3100/lab2$ ./nanosh
nanosh: pwd
/home/tcowan/cs3100/lab2
nanosh: cd ..
nanosh: pwd
/home/tcowan/cs3100
nanosh: cd
nanosh: pwd
/home/tcowan
nanosh: cd cs3100/lab2
nanosh: pwd
/home/tcowan/cs3100/lab2
nanosh:
nanosh:
nanosh:
nanosh: cd too many parameters
Invalid number of parameters to cd: Invalid argument
nanosh: nosuchprogram
execvp() failed: No such file or directory
nanosh: ls -al /etc/vim
total 24
drwxr-xr-x   2 root root  4096 Aug  4 23:12 .
drwxr-xr-x 130 root root 12288 Jan 16 11:28 ..
-rw-r--r--   1 root root  2149 Oct 18  2013 vimrc
-rw-r--r--   1 root root   662 Jan  2  2014 vimrc.tiny
nanosh: cd /etc/vim
nanosh: ls
vimrc  vimrc.tiny
nanosh: exit please!
exit command failed: Invalid argument
nanosh: exit
tcowan@tcowan-Ubuntu:~/cs3100/lab2$
```

Special form of the “cd” command that chdir() to $HOME

Here I hit RETURN 3 times

I called the external “ls” command

Invalid number of parameters

Hints and Suggestions

- When issuing an error message, make sure `errno` is set properly before calling `perror()` or `perror()` will issue the message "Success". Always call `perror()` to issue error messages and `printf()` to issue informational messages.
- Use the `man` pages for syntax for system calls (we will discuss this in class)
- `exit`: call `exit(0)` if no parameters are specified; otherwise, issue an error message, ignore the command and return to loop
- getting parameters: use `strtok()` to read words from the line typed by the user, add to `myArgv` and count in `myArgc`.
- `pwd`: call `getcwd()`
- `cd`: call `chdir()` and `getenv("HOME")`
- executing external programs: call `fork()`, `execvp()` and `waitpid()` and always check the return code from each to report any errors. If an error occurs in `execvp()`, exit the process with `exit(0)` after issuing an error message. If this is mysterious, ask why in class.
- If an error occurs and you need to exit the program, use `exit(1)`. On normal termination, use `exit(0)`.

Grading

I often use scripts to help grade programs. Please note the following:

- My scripts use the `gcc` compiler to compile your programs. If you use a different system to create your programs, please test them on Ubuntu Desktop 14 before submitting them for grading.
- The automated test bed will fail if you do not follow the naming instructions, so please be sure that the programs are named correctly.
- Upload the files you modified for this assignment to Canvas when you are ready for me to grade your work. You should upload only the following, using the Firefox web browser in ubuntu:
  `nanosh.c`
- It is not necessary to upload your files to icarus for this assignment.

Here is how you earn points for this assignment:

FEATURES	POINTS
Must-Have Features	
Files are named correctly and uploaded to Canvas	20
Compiles without errors or warnings	20
Correctly displays the appropriate prompt followed by a single space: `nanosh:`	10
`exit` causes nanosh to exit without any errors, crashes or messages	15
`exit` with any parameters issues perror() with errno=EINVAL and is ignored	10
Required Features	
`pwd` prints the correct current working directory on a single line followed by a newline	10
`pwd` with any parameters issues perror() with errno=EINVAL and is ignored	10
`cd` NEWDIR changes the current working directory to NEWDIR	10
`cd` without any parameters changes the working directory to $HOME	10
`cd` with more than two parameters issues perror() with errno=EINVAL and is ignored	10
`cd` to a nonexistent NEWDIR issues perror() and is ignored	10
Any unknown command is correctly executed with `fork()` and `execvp()`	20
External programs are serialized with `waitpid()`	10
Errors from `execvp()` are reported with a call to `perror()`	10
No messages appear on STDOUT except prompts or output from external commands and all error messages appear on STDERR	10
Chid processes terminate and do not cause multiple instances of the command loop	15
Grand Total	200