Extend the shell program that you implemented in Assignment 2 so that it executes the built-in commands listed below. Your shell is a program that has a loop that will print a prompt (initially "[myshell]%"), read and parse the commands entered until the exit command is entered. In the loop you should check if the command is one of your built-in commands (described below). If the command entered is not one of your built-in commands, you should print "**command**: Command not found." where **command** is the command name that was entered. If the command entered matches one of your built-in commands, you should check that the number and type of argument is suitable for the command based on the description below. If the arguments are not valid, you should print "**command**: Arguments not valid." where **command** is the command name that was entered.

Note that your program will use the fork system call, so remember to use one of the machines in CL115 instead of hercules.

Built-in Commands to support:

- **exit** - takes no arguments; terminates the program
- **cd** - change directory to directory name given; with no arguments, change to the home directory.
- **pwd** - takes no arguments; prints the current working directory.
- **list** - with no arguments, lists the files in the current directory one per line. With arguments, lists the files in each directory given as an argument, with a blank line then the name of the directory followed by a ":" before the list of files in that directory.
- **pid** - takes no arguments; prints the pid of the shell
- **prompt** - When run with no arguments, prompts for a new prompt prefix string. When given one argument, makes that the new prompt prefix. For instance,

  ```
  [myshell]% prompt CS330
  [CS330]% _
  [CS330]% prompt
  input prompt prefix: hello
  [hello]% _
  ```

- **alias** - When ran with no arguments prints the aliases the shell knows about. When ran with arguments it should install an alias into the alias table the shell knows about. Additionally, the shell needs to be able to run your aliases.
- **history** - Should print the last n commands (by default n is 10) executed when ran with no options. When ran with a numerical argument changes the number of commands to list to that number.
- **printenv** - When ran with no arguments prints the whole environment. When ran with one argument, prints the value of that environment variable.

Your program should also be able to handle the execution of commands in background. A background command is specified by the presence of an ampersand (i.e., &) character at the end of a command. For example,

```
[myshell]% history 5 &
```

would result in the history command executing in background. When a command is executed in background, the program should not wait for the command to complete. Instead, it should immediately display the command prompt and wait for further input from the user. Note that any output generated by a background command will be directed to the terminal display and could be intermingled with any output generated from a foreground command.

Note that your shell program has to work completely on its own. You must not use any other available shells.

You might find the following system calls to be useful in implementing your shell:

a. fork: Will create a new process.
b. exec family of system calls: Will execute a file.
c. wait: Will wait for process to terminate.
d. waitpid: Will wait for a particular process to terminate.
e. chdir: Will change a process' working directory.

Extend the shell program that you implemented in Programming Assignment 3 so that it executes the additional built-in commands listed below. Your shell is a program that has a loop that will print a prompt (initially "[myshell]%"), read and parse the commands entered until the exit command is entered. In the loop you should check if the command is one of your built-in commands (described below). If the command entered is not one of your built-in commands, you should print "**command**: Command not found." where **command** is the command name that was entered. If the command entered matches one of your built-in commands, you should check that the number and type of argument is suitable for the command based on the description below. If the arguments are not valid, you should print "**command**: Arguments not valid." where **command** is the command name that was entered.

Note that your program will use the fork system call, so remember to use one of the machines in CL115 instead of hercules.

Built-in Command to support:

- **kill** - When given just a pid sends a SIGTERM to it with kill. When given a signal number (with a - in front of it) sends that signal to the pid. (e.g., kill 5678, kill -9 5678).

  Additionally:

  Ctrl-C (SIGINT) should be caught and ignored if the shell is prompting for a command, but sent to the running child process otherwise. Use signal and/or sigset for this.

  Ctrl-Z (SIGTSTP) should be ignored using sigignore or signal.

  Test your shell by running the following commands in it (in order):

```
list                                    ; test list
cd TestFiles                            ; cd to TestFiles directory
Ctrl-C
pwd
cd /blah                                ; non-existant
Ctrl-Z
Ctrl-C
cd TestFiles/Test JunkFiles/Junk        ; too many args
history                                 ; test history
Ctrl-C
history 15
pid                                     ; get pid for later use
kill
kill pid-of-shell                       ; test default
kill -1 pid-of-shell                    ; test sending a signal, should
                                        ; kill the shell, so restart a
                                        ; new one

prompt          (and enter some prompt prefix)
Ctrl-Z
prompt CS330Shell
printenv PATH
setenv TEST testing
Ctrl-Z
printenv TEST
exit
```