EECS 678 - Quash Shell

Introduction

In this project, you will complete the Quite a Shell (quash) program from scratch using techniques you have learned so far. You may work in groups of 2 or individually. The learning goals of this project are as follows:

- Getting familiar with the Operating System (UNIX) interface.
- Exercising UNIX system calls.
- Understanding the concept of a process from the user point of view.

In essence, Quash should behave similar to csh, bash or other popular shell programs. **You must use C, C++, Rust, or Go to implement this project.**

Features

The following features should be implemented in Quash:

- Quash should be able to run executables (the basic function of a shell) with command line parameters
- If the executable is not specified in the absolute or relative path format (starting with sequences of '/', './', or '../'), quash should search the directories in the environment variable PATH (see below). If no executable file is found, quash should print an error message to standard error. Quash should allow both foreground and background executions. Character '&' is used to indicate background execution. Commands without '&' are assumed to run in foreground.
    - When a command is run in the background, quash should print: "Background job started: [JOBID] PID COMMAND"
    - When a background command finishes, quash should print: "Completed: [JOBID] PID COMMAND"

```
 [QUASH]$ program1 &
Background job started: [1]    2342    program1 &
[QUASH]$ ls
Documents Downloads
Completed: [1]    2342    program1 &
```

- Quash should implement I/O redirection. The `<` character is used to redirect the standard input from a file. The `>` character is used to redirect the standard output to a file while truncating the file. The `>>` string is used to redirect the standard output to a file while appending the output to the end of the file.

```
[QUASH]$ echo Hello Quash! > a.txt  # Write "Hello Quash!\n" to a file
[QUASH]$ cat a.txt
Hello Quash!
[QUASH]$ echo Hey Quash! > a.txt  # Trucates/overwrites a.txt contents
[QUASH]$ cat a.txt          # Print file contents
Hey Quash!
[QUASH]$ cat < a.txt        # Make cat read from a.txt via standard in
Hey Quash!
[QUASH]$ cat < a.txt > b.txt  # Multiple redirect. Read from a.txt and write to b.txt.
[QUASH]$ cat b.txt
Hey Quash!
[QUASH]$ cat a.txt >> b.txt  # Append output of a.txt to b.txt
[QUASH]$ cat b.txt
Hey Quash!
Hey Quash!
[QUASH]$
```

- Quash should support pipes `|`.

```
[QUASH]$ cat src/quash.c | grep running
// Check if loop is running
bool is_running() {
  return state.running;
  state.running = false;
  while (is_running()) {
[QUASH]$ cat src/quash.c | grep running | grep return
  return state.running;
```

- Quash should support comments `# this is a comment`. Quash should parse and interpret anything before the '#' character if applicable, and discard anything after it.

```
[QUASH]$ echo "hello world" # this is a comment
hello world
[QUASH]$ # this is another comment -> Quash does nothing
[QUASH]$
```

Built-in Functions

All built-in commands should be implemented in quash itself. They cannot be external programs of any kind. Quash should support the following built-in functions:

- `echo` - Print a list of strings given as arguments. The string arguments may or may not be surrounded by single or double quotes. The output format should be the same as bash (a string followed by new line '\n'). Variable expansion should be performed for expressions of the form `$VAR`.

```
[QUASH]$ echo Hello world! 'How are you today?'
Hello world! How are you today?
[QUASH]$ echo $HOME/Development
/home/jrobinson/Development
[QUASH]$ echo "Double quoted string" 12345
Double quoted string 12345
[QUASH]$
```

- `export` - Sets the value of an environment variable. Quash should support reading from and writing to environment variables.

```
[QUASH]$ export PATH=/usr/bin:/bin  # Set the PATH environment variable
[QUASH]$ echo $PATH                 # Print the current value of PATH
/usr/bin:/bin
[QUASH]$ echo $HOME
/home/jrobinson
[QUASH]$ export PATH=$HOME  # Set the PATH environment variable to the value of HOME
[QUASH]$ echo $PATH         # Print the current value of PATH
/home/jrobinson
[QUASH]$
```

- `cd` - Change current working directory. This updates both the actual working directory and the PWD environment variable.

```
[QUASH]$ echo $PWD
/home/jrobinson
[QUASH]$ cd ..              # Go up one directory
[QUASH]$ echo $PWD
/home
[QUASH]$ cd $HOME           # Go to path in the HOME environment variable
/home/jrobinson
[QUASH]$
```

- `pwd` - Print the absolute path of the current working directory. Make sure you are printing out the actual working directory and not just the PWD environment variable.

```
[QUASH]$ pwd                # Print the working directory
/home/jrobinson
[QUASH]$ echo $PWD          # Print the PWD environment variable
/home/jrobinson
[QUASH]$ export PWD=/usr    # Change the PWD environment variable
[QUASH]$ pwd
/home/jrobinson
[QUASH]$ echo $PWD
/usr
[QUASH]$
```

- `quit` & `exit` - Use these to terminate quash.

```
[BASH]$ ./quash
Welcome...
[QUASH]$ exit
[BASH]$ ./quash
Welcome...
[QUASH]$ quit
[BASH]$
```

- `jobs` - Should print all of the currently running background processes in the format: "[JOBID] PID COMMAND" where JOBID is a unique positive integer quash assigns to the job to identify it, PID is the PID of the child process used for the job, and COMMAND is the command used to invoke the job.

```
[QUASH]$ find -type f | grep '*.c' > out.txt &
Background job started: [1]    2342    find / -type f | grep '*.c' > out.txt &
[QUASH]$ sleep 15 &
Background job started: [2]    2343    sleep 15 &
[QUASH]$ jobs               # List currently running background jobs
[1]    2342    find / -type f | grep '*.c' > out.txt &
[2]    2343    sleep 15 &
[QUASH]$
```

- `kill` - Given a POSIX signal number (int) and a PID (int), Quash should send the signal to the given process. The format shall be `kill SIGNUM PID`.

```
[QUASH]$ sleep 100 &
Background job started: [1]    4071    sleep 100 &
[QUASH]$ kill 2 4071         # send SIGINT signal to PID 4071
[QUASH]$ jobs                # the process was terminated => no output
[QUASH]$
```

Useful System Calls and Library Functions

The following is a list and brief description of some system calls and library functions you may want to use and their respective man page entries. Note that this list may not be exhaustive, but be sure what ever library functions you use will run on the lab machines:

- atexit(3) - Enroll functions that should be called when exit(3) is called
- chdir(2) - Changes the current working directory
- close(2) - Closes a file descriptor
- dup2(2) - Copies a file descriptor into a specified entry in the file descriptor table
- execvp(3) - Replaces the current process with a new process
- exit(3) - Imediately terminate the current process with an exit status
- fork(2) - Creates a new process by duplicating the calling process
- getenv(3) - Reads an environment variable from the current process environment
- getwd(3) - Gets the current working directory as a string (get_current_dir_name(3) may be easier to use)
- get_current_dir_name(3) - Gets the current working directory and stores it in a newly allocated string
- kill(2) - Sends a signal to a process with a given pid

- open(2) - Opens a file descriptor with an entry at the specified path in the file system
- pipe(2) - Creates a unidirectional communication pathway between two processes
- printf(3) - Prints to the standard out (see also fprintf)
- setenv(3) - Sets an environment variable in the current process environment
- waitpid(2) - Waits or polls for a process with a given pid to finish

You may NOT use the system(3) function anywhere in your project.

Project Hints and Comments

In Quash, a job is defined as a single command or a list of commands separated by pipes. For example the following are each one job:

cat file.txt # A job with a single process running under it

find | grep *.qsh # A job with two processes running under it

A job may contain more than one process and should have a unique id for the current list of jobs in Quash, a knowledge of all of the pids for processes that run under it, and an expanded string depicting what was typed in on the command line to create that job. When passing the pid to the various print job functions you just need to give one pid associated with the job. The job id should also be assigned in a similar manner as bash assigns them. That is the job id of a new background job is one greater than the maximum job id in the background job list. Experiment with background jobs in Bash for more details on the id assignment.

Grading Policy

Partial credit will be given for incomplete programs. However, a program that cannot compile will get 0 points. The feature tests are placed into multiple tiers of completeness. The output to standard out from your code must match our output exactly, except for whitespace, for the next tier of grading to be accessible. This is due to reliance of previous tiers in subsequent tier tests. If we cannot run your code in one tier then it becomes far more difficult test later tiers. The point breakdown for features is below:

Description	Score
• Tier 0 ◦ Quash compiles	10%

• Tier 1
  ◦ Single commands without arguments (ls)
  ◦ Simple built-in commands
  ▪ pwd
  ▪ echo with a single argument
	30%

• Tier 2
  ◦ Single commands with arguments (ls -a /)
  ◦ Built-in commands
  ▪ echo with multiple arguments
  ▪ cd
  ▪ export
  ◦ Environment Variables
  ▪ echo with environment variables (echo $HOME)
  ▪ Execution with environment variables (du -H $PWD/..)
	30%

• Tier 3
  ◦ Built-in commands
  ▪ jobs	

▪ kill ◦ Piping output between one command and another (find -type f | grep '*.c') ◦ Redirect standard input to any command from file (cat < a.txt) ◦ Redirect standard output from a command to a file (cat b.txt > a.txt) ◦ Background processes ▪ Job completion notification	30%
• Tier 4 (extra credit) ◦ Pipes and redirects can be mixed (cat < a.txt | grep -o World | cat > b.txt) ◦ Pipes and redirects work with built-in commands ◦ Append redirection (cat a.txt | grep -o World >> b.txt)	10%
• Valgrind Memory Errors ◦ While not ideal, you will not lose any points for "still reachable" blocks ◦ Unacceptable Memory Leaks ▪ Definately lost ▪ Indirectly lost ▪ Possibly lost ◦ Unacceptable Access Violations ▪ Invalid Read ▪ Invalid Write ▪ Invalid free ▪ Use of uninitialized values	-5% from tier grade down to 0% for each tier with violations

Submission

Each group (or individual) should submit the project via Canvas. The following will be expected in your deliverables:

- Source code files (.c, .cpp, .h, .rs, .go)
- Makefile with the `quash` target defined
- If using Rust or Go, include an 'instructions.txt' file for building and running your program. Include the language version and other potentially helpful information.

Create a tar.gz archive of your deliverables. The TA should be able to run `make quash` to build your `quash` executable and then run `./quash` to execute your program. Ensure your code compiles and executes on the EECS cycle servers (if C/C++).