COMP 3659 – Operating Systems Programming Project 1: A Linux Shell

Overview

In groups of 2 or 3, students will develop their own shell program (`mysh`) that provides a command line interface for a Linux user. The `mysh` program is to provide a *subset* of the functionality provided by common Linux shells such as `bash`.[1]

Deadline to communicate group preferences:	Friday, September 13
Due date for submission of project deliverables:	Friday, October 11 (by 11:59 p.m.)[2]
Required group presentation to instructor:	October 28 – November 8

By the date indicated above, groups must submit the source code for a buildable, working shell, together with all supporting files.

Groups are also required to give a presentation to the instructor. The presentation is to include a student-led demonstration of shell functionality (including discussion of limitations and known bugs), discussion of design and implementation issues and decisions, exploration of course concepts that relate to the project, and explanation of how particular details of the `mysh` design, implementation, and functionality relate to these concepts. All participants must be prepared to answer questions from the instructor and to engage in discussion about the project. Successful presentations must demonstrate comprehensive understanding of related course concepts. Presentations require the participation in oral presentation by all group members and require supporting material (e.g., a small number of slides). More information about the presentation requirement will be posted to the course D2L site in a separate document, and groups will select their presentation time slot during the week of October 7-11. Presentations are expected to last approximately 20 (max. 30) minutes and will normally be conducted in person.

Solutions are to make direct use of the necessary Linux system calls, with minimal or no use of additional library functions (see below).

Groups

Students must inform the instructor via email of their preferred group membership by the deadline indicated above: one group spokesperson should email the instructor with a list of member names, and should copy those other students on the email. Following the deadline, the instructor will facilitate group formation for all remaining students (i.e., for students who do not express any preference or who need assistance).

[1] See the manual entry for `bash` (`man bash`).

[2] October 11 is the official due date, but submissions will be accepted penalty-free until October 18.

The instructor reserves the right to adjust group membership in order to form appropriately-sized groups. If your group has two members and is willing to accept a third, or if you are a student that needs assistance in finding a group to join, please inform your instructor by email.

The instructor also reserves the right to require groups to "divorce" at any point during the semester, i.e., if the group dynamic is dysfunctional. In this case, all group members will be required to complete the project individually.

Before beginning their development work, groups should discuss their mutual expectations and should decide on group work protocols to which all members agree. Groups should discuss any difficulties or perceived problems as soon as they arise. If concerns persist, groups or members should approach the instructor for assistance.

Normally, the groups formed for project 1 will also be the groups for project 2.

Functionality Requirements

Basic Functionality

The shell is to run in interactive mode: it must repeatedly prompt the user, read a command line from the standard input, and then cause the command(s) read to be executed (or the shell must respond with an appropriate user error message when execution is not possible). This process is to continue indefinitely, until the user exits the shell (at which point the shell process is to terminate). A sample interaction is shown below:

```
mysh$ /bin/ls
foo.c stuff sometextfile
mysh$ /bin/cat sometextfile
This is the content of sometextfile.
mysh$ exit
```

At each command line, it should be possible for the user to run valid external[3] commands (or limited sequences of such commands–see below), including commands that may accept zero or more command line arguments. It is also necessary for the shell to support at least one built-in command (`exit`).[4]

It must be possible for `mysh` to run as an interactive login shell.[5] In fact, even though `mysh` should be testable by invoking it directly from within a `bash` session, it should also be possible to set up a test user account that uses `mysh` as its default login shell.[6]

Command line format: a command line is to consist of a sequence of zero or more whitespace-delimited "words", all terminated by a newline. For simplicity, it *is* acceptable for `mysh` to place a reasonable upper bound on command line length (e.g., it may limit command line length to 256 characters).

---

[3] Here, an *external* command refers to a command that is not built into the shell, but rather is a separate executable file, such as an application or system program.

[4] Most commands that are runnable will correspond to commands external to the shell–for example programs like `/bin/ls` and `/bin/cat`.

[5] See the "invocation" section of the `bash` manual entry for a description.

[6] For more information, start by reading the manual entry for the `chsh` command.

It is *not* necessary that the shell provide any sophisticated command-line editing functionality (e.g., `emacs`-like editing using GNU readline) or command history mechanism.

It is *not* necessary that the shell read any profile information on start-up.[7]

Background Jobs

By default, user commands are to be run in the foreground, and `mysh` is to suspend execution until command execution terminates. Users must also be able to run commands in the background (i.e., `mysh` will not wait for commands to terminate before proceeding) by ending the command line with an ampersand (`&`) symbol. The `mysh` functionality and behaviour must be similar to that of `bash` in this regard.

Note: it is relatively easy to implement a basic (and not entirely correct) version of this functionality. This allows the user to create pseudo-background processes, such that the shell process is not properly notified of background job status changes and the user is not able to monitor or control status. Partial functionality will be worth some credit; more complete functionality will be worth additional marks.

Basic I/O Redirection

By default, user commands are to read their input from the standard input and write their output to the standard output. Users must also be able to perform basic input and/or output redirection using less-than and greater-than symbols (<, >). It is not necessary for `mysh` to mimic the full range of `bash` I/O redirection functionality in order to receive an A-range grade, but a reasonable subset of redirection functionality must be supported.

Basic Command Pipelines

Users are to be able to connect the output of one command process to the input of the next by employing the pipe (`|`) symbol. At a minimum, `mysh` must support two-stage pipelines.

Other

Paths: Users should be able to specify any external command using its absolute pathname. Ideally, `mysh` will also support a default search path for commands.

Note carefully that a modern Linux shell is a very sophisticated program that provides a large, diverse set of powerful features. It is *not* necessary for `mysh` to support command completion, lists, compound commands, expressions, coprocesses, shell functions, comments, quoting, parameters (e.g., shell variables), expansion, aliases, evaluation (e.g., arithmetic or conditional), sophisticated job control, shell scripts, or other specialized features of a shell like `bash`.

However, this project is open-ended in that, once the core functionality has been implemented successfully, groups are welcome to continue refining and extending shell capabilities and features as time permits.

Groups that have implemented basic shell functionality as well as support for basic background execution, basic I/O redirection, command pipelines (including of length >2), and command search paths are encouraged to add support for “signals” and to attempt a more sophisticated version of background

[7] E.g., it is not necessary that it provides customization according to `.profile` or `.bashrc` -like files.

job handling. Such groups may also or alternatively wish to implement simplified versions of one or two other advanced features (e.g., simple forms of lists).

Implementation Requirements

General

The `mysh` program is to be implemented in C, must be buildable by running "`make`", and must be runnable on the Linux VM environment for this course.

Restrictions on the Use of Library Functions

The `mysh` program may not be linked to any libraries other than the C standard library. Further, the final version submitted must endeavour to make minimal or, ideally, *no* use of C standard library functions other than C wrapper functions to Linux system calls.[8] The intention is that the program relies only on system calls and not on any third-party library functions.

Code and Documentation Standards

Because this is a third year course for computer science majors, groups are expected to understand and follow good design, coding, documenting, and testing practices that yield clear, correct, and readable code that is easy to maintain, modify, extend, and reuse. A code and documentation standard will not be released for this course. Instead, each project group is expected to develop its own code/documentation standards policy document, to include it when submitting the solution, and to briefly describe it during the group presentation.

Note carefully that well-written solutions will be rewarded and poorly written solutions will be penalized.

Development and Assessment

Groups are encouraged to start early and to develop their shell incrementally according to the feature order described in the *Functionality Requirements* section. Various lab exercises are designed to provide just-in-time experience with key concepts and system features necessary to complete the project. The instructor may provide additional development guidance as appropriate, including during class time and via material posted to the D2L site.

An overall project mark (also covering assessment of the presentation portion) will be assigned after the group presentations have concluded. Work will be graded in accordance with the grading scheme described in the course outline.

The functionality portion of the instructor's detailed marking scheme will conform to the general guidelines below:

- Project that correctly implements basic functionality requirements: high F (~45%) to D range
- Project that also correctly implements *basic* background jobs: D+ to C- range
- Project that then also correctly implements basic I/O redirection: C range
- Project that then also correctly implements basic (2-stage) command pipelines: C+ to B- range

[8] This means that the shell is not to rely on dynamic memory allocation implemented via library functions like `malloc` and `free`.

- Project that then also correctly implements *n*-stage command pipelines: B range
- Project that then also correctly implements command search paths: B+ range
- Project that then also implements some additional enhancements, up to and including some basic signal handling: A- range
- Project that then also implements more complete and correct signal, terminal, and/or background job handling: A to A+ range
- Project that then implements further shell features: high A+ range (~100%)

However, there will also be major portions of the marking scheme that cover the following:

- Quality of design and implementation
- Evidence of systematic and robust quality assurance practices
- Quality of supporting material (to be submitted as part of the project deliverables), including:
    - Documents related to quality assurance, including a comprehensive test plan
    - Documents related to project governance and management (e.g., policies developed by the group, such as code/documentation standards; development timeline; plan for division of labour; evidence of regular group meetings and development activity, e.g., via minutes or project logs; high-level design documents; etc.)
- Quality of accompanying presentation

Each group may be required to perform self-assessments at one or more points during the project and to share these with the instructor, and groups may be asked to participate in informal self-assessment activities of each other's work.

Miscellaneous

Groups are encouraged to employ software development practices and systems (of their choice) for efficiently and effectively managing their source code and related data. Groups should discuss and make these decisions, and they should document them as part of the project's supporting material. This may include the use of version control software that facilitates collaboration,[9] as well as processes for safeguarding work through robust backup and recovery protocols.

This document does not provide a full account of all background information needed to complete the project. Groups are expected to start early, manage their time wisely over the time allotted, research relevant Linux and system call details, and seek guidance from the instructor as appropriate.

Submission Instructions

Each group must submit their project source code tree to the instructor by the due date indicated above. Groups seeking an extension must contact the instructor prior to the reading break.

Remember: The `mysh` program must be implemented in C, must be buildable by running "`make`", and must be runnable on the Linux VM environment for this course.

Once ready to submit, *one* member of the group must do the following:

[9] E.g., Git. A guide to using Git in COMP 3659 has been posted to the course D2L site.

1. From your personal Linux VM, ensure that your copy of the source code tree is up to date[10] and ready for submission;
    a. Perform a final, full rebuild of the project;
    b. Perform a final test of shell functionality;
2. Delete all auto-generated files, such as ".o" files, the final executable, and auto-backup files generated by text editors;[11]
3. Change to a directory one level higher than your project working directory, and then create a compressed archive of that directory using the following commands:

    ```
    cd ..
    tar cf mysh.tar projdir      ← where projdir is your directory
    gzip mysh.tar
    ```

    You should now have a file named `mysh.tar.gz` in the current working directory.
4. **Using D2L**, upload this file as part of your group's submission for the project.[12]
    a. To submit this this file, it is easiest to first copy it to your local personal computer (e.g., using a file copy utility like WinSCP) and then to upload it to D2L from there.
5. Also using D2L, upload any additional documents that your group created and maintained during the development process, such as:
    a. Documents related to quality assurance, including a comprehensive test plan;
    b. Documents related to project governance and management (e.g., policies developed by the group, such as code/documentation standards; development timeline; plan for division of labour; evidence of regular group meetings and development activity, e.g., via minutes or project logs; high-level design documents; etc.)
6. Note carefully: it is *not* required to submit supporting material for your presentation (e.g., slides) at this time. Your group will finalize and share these later, in the lead-up to your presentation time slot.

If you have any questions about the submission process, please consult with your instructor. Please also familiarize yourself with the group presentation requirements, which will be posted separately on the course D2L site.

[10] E.g., if using a tool like Git, all contributors should have pushed their changes to the shared repository and then the group member who is submitting should have pulled all such changes.
[11] Ideally, your Makefile supports a "`make clean`" option.
[12] Please do not use email to submit your work unless D2L submission is not working.