ECE 421 Design Document

Kirby Banman

Ryan Thornhill

Feb 17, 2015

Answers To Questions

1) Shell REPL

1.) Is your problem a class or a module? What is the difference?

Class. Both contain state and methods using the state. A Class is dynamically stateful, a Module has only immutable state. Classes can be instantiated into objects, Modules cannot.

The shell should be a class because a shell needs to maintain state. For example, history, current background processes, etc.

2.) What shell(s) are you using to provide a specification? What features do they support?

The POSIX-specified `sh` will serve as our specification. It is simpler than the common BASH and has a well defined spec we can pull from. It supports many features, including:

- variables
- loops
- ranges
- Built-in commands - echo, CD, etc
- Pre-execution command expansions. (https://www.gnu.org/software/bash/manual/bashref.html#Shell-Expansions) with symbols, env variables,
- Subshell/subcommand command execution
- Pipes
- Output redirection
- Signal handling
- Error reporting
- User switching - su - user
- Root emulation - sudo

Our shell will support:

- cd
- history
- environment variables (static, not per execution)
- Command expansion (env $variables and directory shortcuts only)
- ctrl-c signal handling

3.) In your opinion, which features are essential (should include in your design) and which are “window dressing” (should not include in your design)?

Our shell will support:

- cd
- history
- environment variables (static, not per execution)
- Command expansion (env $variables and directory shortcuts only)
- ctrl-c signal handling

We don’t necessarily believe that other features are window dressing but they would take too much time to implement:

- Scripting (file input rather than REPL)
- Startup Scripts (EX: .bashrc)
- Keyboard shortcuts like moving forward one word (EX: ctrl + left)
- Looping
- Brace Expansion (EX: subshells)

4.) Economics: Which, if any, essential features will be omitted from your design due to unmanageable effort requirements?

The above list as well as:

- Tab completion
- Looping
- Conditionals
- I/O redirection
- Regular Expression

5.) Error handling? What percentage of code handles functional against potential pitfalls:

In the average commercial program? In your shell program?

If they are radically different, please provide a rationale.

In my school projects, I think the error handling code accounts for 20 - 40 % of the entire code base. In my professional experience I think the percentage is slightly higher, 30 - 50 %. In our shell program we expect to have about 50% of the code base dedicated to error handling.

Between contracts, sandboxing, signal handling (SIGINT, SIGCHLD), and exception handling there will be a significant amount of error handling code.

6.) Robustness? How do we make the system bullet-proof? Is Avoiding Core dumps of system shells important? Especially from a Security viewpoint, remember this dump will give access to underlying C system code and potentially Linux daemons?

We will be relying heavily on sandboxing for our shell program to avoid malicious input. Beyond that we will be doing some input sanitization and lots of exception handling to ensure that the program doesn't crash. Avoiding core dumps is essential because because the dump can contain sensitive information such as passwords. When the passwords are stored they are encrypted but can exist in plaintext in memory. Sometimes core dumps will follow symlinks and a user could exploit a bug that causes a core dump to overwrite system files.

7.) Describe the Ruby exception hierarchy, which classes of exceptions are applicable to this problem?

The ruby exception hierarchy looks like the following:

- Exception
    - fatal
    - NoMemoryError
    - ScriptError
    - SecurityError
    - SignalException
    - StandardError
    - SystemExit
    - SystemStackError

Each of the subheading have further subclasses of exceptions. Standard Error has the most subclasses including many important exceptions like RuntimeError or DivideByZeroError. We feel the following will be important to this problem:

SecurityError

SecurityError will be raised when the $SAFE level is above zero and a potentially unsafe action is taken. We will be using the $SAFE variable to implement sandboxing so we will need to be aware of this exception.

SignalException

SignalException will be raised when a signal is received. We will be implementing some ctrl^c functionality so it will be important to trap this exception.

IOError (Under Standard Error)

IOError occurs when an IO operation occurs. We will have to read and write to files to do sandboxing, so catching this error will be a part of good I/O practice.

RegexpError (Under Standard Error)

Raised when given an invalid regular expression, this will be important if we use regular expressions to sanitize input

ThreadError (Under Standard Error)

Raised when an invalid operation is attempted on a thread. We will use threads heavily in this assignment, so this will be important to be aware of.

8.) What is Module Errno? Is it applicable to the problem? Explain your answer! Remember Ruby often wraps C code.

Module Errno is created dynamically to map operating system error numbers to ruby classes. Each error number is mapped to a subclass of SystemCallError. The Errno module is applicable to this problem because we will be using system calls within our c code that can return OS error codes. We'd like to be able to handle these responses directly and cleanly from our ruby code. Since the module is created dynamically for each platform, it makes our code much more portable.

9.) Security? How will we protect the system from tainted objects? Can we trust the user?

Is sand boxing applicable to this problem? Is it feasible to write security contracts?

We will be protecting the system from tainted objects using sandboxing. Specifically in ruby we are given access to the &SAFE global variable that allows us to partition the running program into two parts, system code and unsafe user input. Sandboxing is enabled in ruby by setting the

$SAFE variable equal to 4. Since the main purpose of our shell program is to take user input we want to ensure that any malicious input has very restricted permissions this makes sandboxing very relevant to this problem. The effort required to write in depth security contracts would simply be too great for our short timeline. Developing highly secure software is a topic of continued research and unfeasible in this case. We will adhere to standard security protocols and use the as many of the out of the box security features that we can leverage.

10.) Should we be using class GetoptLong? Or Regexp? Or shell? Or

GetoptLong is more appropriate for the 2nd or 3rd part of this assignment. I don't think starting the shell will require arguments.

Regexp will definitely be applicable and useful in sanitizing user input. To what extent we utilize regular expressions for input sanitization is unknown right now, but we will use them in some capacity.

The Shell class will be a utility that we will utilize to delegate common shell activities and will provide some very useful functionality.

11.) What environment does a shell run within? Current Directory?

A Shell typically runs with the home directory environment, and settings defined in a startup script such as bashrc. We are planning on having our own shell environment variables, so a startup script might be something we investigate doing as well. Our shell will run with the home directory as the starting environment like bash and sh does.

12.) What features should be user controllable? Prompts? Input and Output channels?

Input and output channels are an important feature in shell programs, we will be implementing a version of stdin and stdout that can be set by the user for input and output redirection. Customizable prompts are a feature that we think aren't vital to the operation of a shell, we will forgo this in our design.

2) Timer

1.) Is your problem a class or a module?

The timer itself fits a module well, there is no need for it to be instantiated or keep track of state. However, we have used the decorator design pattern to implement our contracting system and this requires us to use a class for any functionality that will implement contracts, so it will be a class. The driver program will have a class to organize the code.

2.) Error handling? Robustness? Security? Are any of these required?

Since we are taking command line inputs there will be a need for heightened security. This component will be written in C and take advantage of multithreading so robustness and error handling will be a central part of the design.

3.) What components of the Ruby exception hierarchy are applicable to this problem? Illustrate your answer.

Errno Module

The errno module will be of use for this task as the C code will return error codes from the OS. Handling them portably in ruby will make use of this module.

ThreadError

Since the timer is to be non-blocking we will make use of threading. We will have to be ready to catch threading errors.

RegexpError

We may use regular expressions for the parsing of the command line arguments, we will look for this exception.

4.) Is Module Errno useful in this problem? Illustrate your answer.

As I detailed above, the C code will interact with system calls that will return OS error codes. To make our module as reusable and portable as possible we will make use of the Errno module. This way the errors can be handled in the Ruby system layer rather than in the C code.

5.) Describe the article at: http://today.java.net/pub/a/today/2006/04/06/exception-handling-antipatterns.html Convince the marker that these Anti-patterns don't exist in your solution.

The article today.java.net describes some poor practices when writing error/exception catching code and why they are incorrect. In all of the examples the article gives the best practice alternative. The article stresses the importance of reliably handling errors and making the causes easy to find. In our solution we will keep these antipatterns in mind during all stages of design and development.

6.) How can I make the timing accurate? What time resolution should I be looking at, remember real-time systems? Time formats? Does ‘C’ have better facilities for this problem than Ruby?

To make timing accurate we must be able to poll the time very often. This requires that the code run very quickly. For this problem C has far better facilities because we can not only write a tight loop for accurate timing but also make use of the time.h library to keep track of time at a resolution of nanoseconds. Our solution will offer functions to keep track of different resolutions of time (milliseconds, seconds, minutes…) at an accuracy of .nanoseconds.

7.) What should be user controllable? Can we trust the user?

The amount of time and the message to be displayed should be user controllable. Since we aren’t “executing” user input we don’t have to be overly protective. We should however, be on the lookout for bad input, eg. a negative time input.

3) File Watcher

1.) Is your problem a class or a module?

This problem lends itself to the class model a little better than a module. We will probably need to keep track of state and initialization makes sense for a file watcher.

2.) Error handling? Robustness? Security? Are any of these required?

Error handling and robustness are always important for user experience, we don’t want to be throwing exceptions and printing stack traces all the time. A trivial error mode that we should be on the lookout for is the file being renamed. Security for this system isn’t quite as important as for part 1. This system is being written as a usable component so we trust the user of the component to create his own security precautions.

3.) What components of the Ruby exception hierarchy are applicable to this problem? Illustrate your answer.

IOError

IOError will be important since we will be dealing with the file system. We will have to be ready for all kinds of I/O Exceptions

ThreadError

Since the system is to be non-blocking we will be using a multithreaded approach. Thus we have to be ready for threading errors.

4.) Does this problem require an iterator?

An iterator will be required when the system is watching a directory. We will need to iterate over every file (or subdirectory) in the directory and look for changes.

5.) Describe Java's anonymous inner classes

Java's anonymous inner classes are classes defined within the code of another class method and are never saved to a class variable. They are useful for instantiating an object that implements an interface for a specific purpose.

6.) Compare and Contrast Java's anonymous inner classes and Ruby Proc objects; which do you think is better?

Both the java anonymous inner class and Ruby Proc are ways of implementing closures. Closures have become an important construct in modern OO programming as a way to pass functionality around a program. Obviously the anonymous class being part of Java is type safe, but the Ruby Proc gives a little more versatility since any Proc can be passed to any proc-accepting method. I believe the java version is the lesser of the two, there is a significant amount of boilerplate code required to use an anonymous class which can be cumbersome. The Ruby version is extremely flexible and doesn't require unnecessary lines of code. However, I think this comparison is more a comparison of using a construct in a strongly typed language vs a duck typed language.

7.) From a cohesion viewpoint, which *interface protocol* is superior? Explain your decision!

If we ignore the code that would actually implement those protocols the protocol with only one method is superior from a cohesion standpoint. However, if we consider that the backing of those two interfaces is probably identical, we would say that the cohesion of the *module* is probably the same.

8.) Is Module Errno useful in this problem? Illustrate your answer.

The only module that will be interacting with OS error codes is the timer. All of the error handling for that module should be self contained so I do not think there will be a need for Errno in this part of the task.

9.) Do any of the Anti-patterns described at:

http://today.java.net/pub/a/today/2006/04/06/exception-handling-antipatterns.html

Exist in your solution.

As in the above answer to this question, we will be conscious of these anti-patterns in all levels of our design. We will be sure to avoid any of the mentioned bad practices.

10.) Describe the content of the library at:

http://c2.com/cgi/wiki?ExceptionPatterns

Which are applicable to this problem? Illustrate your answer. Which are applicable to the previous two problems? Illustrate your answer.

The library describes a number of design patterns to be used when throwing and catching exceptions. While the focus is on the Java language the methodology is applicable to Ruby and many other languages that support exceptions.

There are some very relevant patterns to the problem we are solving in this assignment:

Design by contract

The focus of the first part of this task, the documentation is important motivation as to why and how we implement design by contract.

UseExceptionsInsteadOfErrorValues

Although the Errno module is not applicable to this part, this is essentially what the Errno Module is doing so it is applicable to part 2 of this assignment.

Bouncer Pattern

The bouncer pattern is a good approach to input checking, it will be used throughout all parts of the assignment.

Let Exceptions Propagate

Let Exceptions Propagate is important in cases where we can gracefully handle a thrown exception. This might be an exception that means more to the user than to us.

11.) The UNIX file system treats 4 entities as files:

1. Ordinary Files:

   Ordinary files are what a typical user thinks of as a file, images, documents, etc.

2. Directories:

   Directories are files that hold other files

3. Special Files:

   These are files that are meant to represent physical devices, like usb drives, or printers

4. Pipes:

   Pipes act as a temporary Ordinary File that are used to transfer input and output between commands.

   We will support all of the above files in our implementation of the file watcher

12.) Define what is meant (in a LINUX environment) by file change? Does it mean only contents? Or does it include meta-information? What is meta-information for a file?

File changes in the linux environment means not only data within the file but also meta-data. Meta-data includes:

- Filename
- Owner Group ID
- Owner User ID
- Time of last access
- Time of last modified
- Total Size
- Device

- Inode
- protection
- etc

For our implementation we will probably focus on only a few of these attributes because of the short time frame.

Augmented Design Decisions

Shell

4.) Economics:
We were able to add some functionality that we originally believed was going to require too much time. We have added:

- tab completion for file paths
- IO redirection with pipes (|) and file output redirection (both append >> and overwrite >)
- keyboard shortcuts for cursor movement (EX: ctrl + left)
- background task execution (trailing & character)

6.) Robustness and Security
Ruby exceptions, system errors, and system signals are handled for all paths of execution using user input. This mitigates the risk of exploitation by targeted failures, as well as the risk of exposing dangerous information to the user (like a core dump).

9.) Security
We have found that we did not have to use the $SAFE variable provided by ruby to do adequate sandboxing. We have 2 types of commands, builtins and exec commands. Our builtins do not need to be sandboxed, because only valid, safe inputs are accepted and all others are rejected. So even though the builtins have access to shell state, they cannot damage it. And for the exec commands (commands passed to the underlying shell), all execution occurs in a separate forked process. Hence, shell state is safe from modification regardless of malicious intent and malformed input.

10.) In addition to Regexp and Shell features of ruby, the following shell-related features were used:

- Readline
- Shellwords

Timer

2.) Robustness

We ended up implementing the error handling primarily through the contracts. Using them to both sanitize input and also handle error conditions. Robustness proved to be unimportant, since the timer runs as a user program and could not produce a core dump that would produce sensitive information

3.) Exception Hierarchy
ThreadError was not necessary since we decided to use fork to create a separate process instead. Similarly, RegexpError was not necessary as we did not need to use regular expressions to sanitize input. We relied on a duck typing structure, and a let it fail (gracefully) methodology.

FileWatcher

3.) Exceptions
We ended up delegating much of the file system interfacing to a 3rd party gem so we did not need to explicitly interact with IOError.

4.) We delegated this task to a 3rd party gem, as such it did not require us to do any iteration besides iterating over multiple paths passed via the terminal.

Contract Deviations

Shell

In order to make our own contracts library work within Command initialization, Command parsing was broken out into a separate class - CommandParser. This should have been a module, but the contracts library does not support modules. (The contracts library will be replaced by an alternative in future for these reasons.) So, parsing method preconditions and postconditions were moved to the CommandParser class.

To simplify the Command.execute method, input/output redirection was rewritten as a method that reassigned $stdin and $stdout, yielded to a block, then restored $stdin and $stdout. This new Command.redirect_io method was given a precondition to ensure the desired IO targets were actually streams.

Finally, as reasoned above in Augmented Design Decisions -> Shell -> 9.), the precondition on Command.execute for $SAFE==4 was removed.

File Watcher

We ran into a few problems while trying to utilize our home grown contracting solution. The contracts sent in the initial submission utilized this framework heavily. After I had completed the

implementation of FileWatcher, I extracted these contracts into module methods and simply called them at the beginning and end of methods that they belonged to.

Additional Testing

Time Delay

This implementation lent itself well to automated testing, as such I created a test suite for the Mini Test library to test it's features. Unfortunately there wasn't enough time to implement an automated suite to test the driver program. As such, the testing for the driver was done manually, by entering a series of valid and invalid inputs at the command line in order to elicit error modes.

FileWatcher

This implementation proved difficult to test as I made use of exit calls in order to refuse invalid command line options. When the exit call was executed the Minitest framework would also be stopped. Furthermore, this solution relied upon long running background processes, doing cleanup after each test would be complicated and there simply wasn't time to implement such a solution.

With the above complexities in mind the decision was made to complete the entirety of the testing for this module and it's driver manually. This was done primarily by entering combinations of valid and invalid arguments at the command line in order to elicit error modes in the module.