**ECE 421: Assignment 2 - Systems Programming in Ruby**

**By: Reem Maarouf, Calvin Ho, and Aaron Philips**

Part one pondering and answering questions

* Is your problem a class or a module? What is the difference?
  + The shell wrapper will be a class. It would need to keep track to the state its in which is why a class is chosen for this problem. The difference between a class and a module is that modules are about for providing methods for different classes. Classes in ruby are about creating an object not whilst modules are about creates functions that can used in those classes.
* What shell(s) are you using to provide a specification? What features do they support?
  + Bash will be used for specifications. It supports file navigation(ls, cd), file manipulation(mkdir, touch, rm), process commands(ps),self management (timing?, alias) Including but not limited to:  
    We still need a list of commands and what they mean
  + Alias   
    Ls  
    Cd  
    Rm  
    Mkdir
* In your opinion, which features are essential (should include in your design) and which are “window dressing” (should not include in your design)?
  + Essential features are the ones listed above, and “window dressing” features are autocomplete, time format.
* Economics: Which, if any, essential features will be omitted from your design due to unmanageable effort requirements?
  + Window dressing, and autocomplete
* 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 commercial programs, error handling can take up to 50% for safety critical systems. In our shell, it should take about 30% for checking valid inputs, buffer overflows, privilege checks, critical section checks, and security checks. Our shell should be robust and not crash often while the commercial program should notify if any errors occurred as soon as possible and try to fix them. This is because our shell does not try to execute any dangerous/ user code on its own. Rather, it relies on forking processes to handle this code, and does not attempt to recover errors that arise within them: only let the user know their state.
* 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?
  + As mentioned above, our system would be robust in it simplicity. As no faulty code would be directly run from the main shell process, it can never fault itself. Only its child processes can fail. Our shell would be secure and utilize information hiding such that dump cores’ information would be parsed so that the user would only see information pertinent to the user’s privilege level
* Describe the Ruby exception hierarchy, which classes of exceptions are applicable to this problem? - load error? syntaxError.   
  + Syntax error- an error which the input command is not recognized.
  + Load error- when a command does not succeed, for example trying to remove a file that does not exist
* <https://ruby-doc.org/core-2.1.1/Exception.html>
* What is Module Errno? Is it applicable to the problem? Explain your answer! Remember Ruby often wraps C code.
  + Yes its applicable. The Errno class maps the error codes returned by the system or c code to ruby error classes
* 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 cannot trust the user at all. We will implement sandboxing by adjusting Ruby’s $SAFE variable to a value of 4, which completely splits the system into 2 sections: a tainted and non-tainted portion
* Should we be using class GetoptLong? Or Regexp? Or shell? Or …
  + We are planning on using the OptionParser and Shell classes of Ruby. The OptionParser will get the inputted command while the Shell class will execute the command.
* What environment does a shell run within? Current Directory? Or ....
  + The default environment will be the current directory. While our shell (ruby script) will run out of the current directory that it’s in, the shell will be able have all the functionalities of other shell program: manipulating files in different paths, etc. The path of the directory can be specified as an input parameter.
* What features should be user controllable? Prompts? Input and Output channels? Or ....
  + Features such as menus of commands and help prompts that are mainly text input/output based should be user controllable. The user controllable features should not involve any critical section or memory manipulation to protect the system.

Part 2 questions

* A class or a module?
  + The control of time in the program will be implemented as a module. The functionalities of the module will be used within other classes when necessary.
* Error handling? Robustness? Security? Are any of these required?
  + Error handling, robustness, and security are all required for input checking. It must be ensured that the program is non-blocking and only allows for a few processes to be executed at a time.
* What components of the Ruby exception hierarchy are applicable to this problem? Illustrate your answer.
  + Syntax Error - error in which command line does not recognize
  + Interrupt Exception - Error occurs when control-c is pressed in the command line. It will safety exit the system
* Is Module Errno useful in this problem? Illustrate your answer.
  + Yes it is useful for this problem as it maps operating system exceptions and errors into ruby classes. An occurrence of this could be when the program interrupts the shell, raising a EINTR.
* Describe the article at: <http://today.java.net/pub/a/today/2006/04/06/exception-handling-Antipatterns.html>

This article describes some of the different patterns to avoid when throwing exceptions. Some of the anti-patterns are outlined below:

* + When handling errors, do not log the error AND throw it. Only do one as it’s easier for the support engineer to debug.
  + Ensure to always throw the specific exception instead of the general exception class because the general exception is not very descriptive and can make it difficult to debug.
  + In general, the article describes how to handle exceptions as it is key to building a robust, reliable system. By properly handling error exceptions, the system becomes maintainable, resilient to change, and works well with other systems.
* Convince the marker that these Anti-patterns don’t exist in your solution.
  + We will try to avoid using exception handling which is a hallmark of
* Do they exist in your Shell solution?
  + They do not exist in our shell solution.
* How can I make the timing accurate? What time resolution should I be looking at, remember real-time systems? Time formats?
  + Ensure the timing is to the nearest millisecond. The format of the timing will include the milliseconds.
* Does ‘C’ have better facilities for this problem than Ruby? (Big hint!)
  + Yes, it has a better time resolution that ruby and will be used in this part.
* What should be user controllable? Can we trust the user?
  + It is required for the user to be a superuser in order to alter or access any files in the system. There would still be a limit to what a superuser can modify. Kernel files will not be able to be accessed by the superuser.

Part 3 questions

* A class or a module?
  + Module, for file mangement
* Error handling? Robustness? Security? Are any of these required?
  + All three of error handling, robustness, security are required. For example if you try to destroy a list of files that do not exist, you need to be able to handle that error, and calling program to continue to function. Similarily if any one of the FileWatcher actions fails or runs into any error( ie modified during action) the program should be able to continue to run, so robustness is required.
* What components of the Ruby exception hierarchy are applicable to this problem? Illustrate your answer.
  + Interrupt exceptions
* Does this problem require an iterator?
  + Yes this problem requires an iterator to cycle through the filenames and execute the alteration and action
* Describe Java’s anonymous inner classes.
  + Anonymous inner classes are in line unnamed classes, that either extend existing classes, or are entirely new. They are useful when you need to create a new instance of a new object, but an object of this kind is never used elsewhere. Its a single use object that's created and instantiated all in line. A lambda object if you will
* Compare and Contrast Java’s anonymous inner classes and Ruby Proc objects; which do you think is better?
  + Ruby proc objects are better in a way as they are just blocks of code. These blocks of code are more versatile as they are just bound to some local variables that are accessed whenenever the proc is called. Also you don’t need to speficify an entire class just for a few lines of code
* From a cohesion viewpoint, which interface protocol is superior? Explain your decision!
  + Since cohesion is about how well a module sticks to doing one thing and one thing only, the second interface protocol would be preferable. The motivation behind the first protocol is probably logical cohesion as all three alteration types may do similar things, and it would follow DRY principles to keep it in one interface. We, however, are going to get around that by having all the common elements of all alterations in a private parent module, so as to hold DRY principles.
* Is Module Errno useful in this problem? Illustrate your answer.
  + Module Errno would be useful to handle all the system error codes thrown and be able to convert them into their ruby counterparts. For example trying to delete a file that no longer exists should read the error code properly
* 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.
  + We will try to avoid these
* Describe the content of the library at: <http://c2.com/cgi/wiki?ExceptionPatterns> Which are applicable to this problem? Illustrate your answer.
  + The security door pattern could be useful to make sure only users with sufficient privilege are altering important files. Tidying up before throwing could be useful when we are altering file sin the system
* Which are applicable to the previous two problems? Illustrate your answer.
  + Let exceptions propagate is a good idea for the first problem, as we want to keep the code that handles failures to a minimum; it is already secure due to sandboxing
* Is a directory, a file? Is a pipe, a file? Is a ....., a file? Tell us your thoughts on the definition of a file in a LINUX context.
  + Everything is file according to linux. Even ports and pipes. They are just files with consequences as to what happens to the current system, or others, as well as the programs within them interact with these “files”. This can include sockets and hardware devices. A directory may not seem like a file, and is just another index in a file system. But in unix systems even these indices can be treated as file descriptors. This file descriptor form of labeling extends to pipes sockets etc.
* 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?
  + A file change is anything that modifies a file’s contents or its metadata. For example. The ‘touch’ command adjusts a file or potential file if it doesn’t exist other than just accessing. It either creates it, or modifies its modified last time. There is also the modify and change timestamp that are affected. The contents however are not changed

def shell\_invariant()

assert(self.respond\_to?(:get\_errno), 'not a valid shell\_wrapper')

end

def post\_invariant()

assert\_equal self.get\_errno,:NOERROR, 'failed shell\_intialization'

shell\_invariant

end

def pre\_create\_manage\_child(commands)

assert(commands.respond\_to?(:chomp), 'command was not a string')

shell\_invariant

end

def post\_create\_manage\_child()

assert(self.get\_errno.respond\_to?(:to\_s), 'failed to get errno' )

shell\_invariant

end

def pre\_execute(commands)

assert(commands.respond\_to?(:chomp), 'command was not a string')

shell\_invariant

end

def post\_execute(myval)

assert(myval.respond\_to?(:round), 'recieved an invalid errno')

shell\_invariant

end

def pre\_convert\_errno(myval)

assert(myval.respond\_to?(:round), 'recieved an invalid errno')

shell\_invariant

end

def post\_convert\_errno(new\_errno)

assert(new\_errno.respond\_to?(:to\_s), 'failed to get errno' )

shell\_invariant

end

def pre\_parse\_command(commands)

assert(commands.respond\_to?(:chomp), 'command was not a string')

shell\_invariant

end

def post\_parse\_command()

shell\_invariant

end

def pre\_timed\_delay\_message(milliseconds)

assert(milliseconds =~ /^\d+$/ ? true:false, 'invalid delay time')

end

def post\_timed\_delay\_message(parent\_pid)

assert(Process.pid() != parent\_pid, 'parent process still alive' )

end

def pre\_creation(duration, list\_filenames)

assert(duration.respond\_to?(:round), 'invalid delay time')

assert((list\_filenames.respond\_to?(:to\_a) or list\_filenames.respond\_to?(:chomp)), 'invalid list of filenames')

end

def post\_creation(file\_name\_hash)

assert(file\_name\_hash.empty?, 'list of file names not empty')

end

def pre\_alter(duration, list\_filenames)

assert(duration.respond\_to?(:round), 'invalid delay time')

assert((list\_filenames.respond\_to?(:to\_a) or list\_filenames.respond\_to?(:chomp)), 'invalid list of filenames')

end

def post\_alter(file\_name\_hash)

assert(file\_name\_hash.empty?, 'list of file names not empty')

end

def pre\_destroy(duration, list\_filenames)

assert(duration.respond\_to?(:round), 'invalid delay time')

assert((list\_filenames.respond\_to?(:to\_a) or list\_filenames.respond\_to?(:chomp)), 'invalid list of filenames')

end

def post\_destroy(file\_name\_hash)

assert(file\_name\_hash.empty?, 'list of file names not empty')

end