ECE 421: Assignment 2 - Systems Programming in Ruby

Fortunately, systems programming in Ruby is similar to systems programming on any Linux box; hence this assignment we will rely on the knowledge that has been accrued from the operating systems class (CMPUT 379). Below are three short exercises to get everyone back up to speed while exercising some new concepts and ideas.

For example, systems programs are real security nightmares. Designing and implementing a truly secure program is actually a difficult task on Unix−like systems such as Linux and Unix. The difficulty is that a truly secure program must respond appropriately to all possible inputs and environments controlled by a potentially hostile user. Developers of secure programs must deeply understand their platform, seek and use guidelines (such as these), and then use assurance processes (such as inspections and other peer review techniques) to reduce their programs' vulnerabilities.

Following standard practice, here are some of the key guidelines that 'C' programmers follow:

- Validate all your inputs, including command line inputs, environment variables, CGI inputs, and so on. Don't just reject ``bad" input; define what is an ``acceptable" input and reject anything that doesn't match.
- Avoid buffer overflow. Make sure that long inputs (and long intermediate data values) can't be used to take over your program. This is the primary programmatic error at this time.
- Structure program internals. Secure the interface, minimize privileges, make the initial configuration and defaults safe, and fail safe. Avoid race conditions (e.g., by safely opening any files in a shared directory like /tmp). Trust only trustworthy channels (e.g., most servers must not trust their clients for security checks or other sensitive data such as an item's price in a purchase).
- Carefully call out to other resources. Limit their values to valid values (in particular be concerned about meta-characters), and check all system call return values.
- Reply information judiciously. In particular, minimize feedback, and handle full or unresponsive output to an un-trusted user.
- Avoid Core dumps, especially in situations where the core dump may have the SETUID bit set!

Um – now which of these are likely to apply to Ruby programs? Remember Ruby programs are often only Ruby wrappers around components written in other languages, especially C.

These exercises are worth 20% of the final mark for this class; as with all of our assignments it has three components:

(i) You must provide complete design rationale and system research – i.e. the written answers to the design questions; this must be handed in by **Tuesday February 23rd @ midnight**;

(ii) You must provide complete design by contract test cases for the three problems; this must be handed in by **Tuesday February 23rd @ midnight**; please note: since these problems are systems programming problems you can expect your contracts to be more abstract than in the previous assignment; and

(iii) Your system must be completed by: **Tuesday February 23rd @ midnight** and must be ready for demonstration in the following practical session.

In addition to the practical demonstration, you are required to hand-in:

1) A detailed rational for ***any augmented decisions*** (from Part 1) with regard to the below design questions.
2) A list of deviations in the contracts implemented form the contracts specified in Part 2.
3) A copy of the code
4) A description of any additional testing beyond that described by your contracts.
5) A list of known errors, faults, defects, missing functionality, etc ….telling us about your system's limitations will score better than letting us find them!

Please hand in all components by emailing them to {jimm, zuhori}@ualberta.ca and hence all sub-components by definition must be machine readable. In addition,

- The Subject Line should be in a specific format - Assignment 2: Group 1 [, Part 1...]
- File types should be only of .doc or .pdf
- File names for code should be in ruby format (all lower cases, with words concatenated with a dash)
- For each assignment with a code section, there must be a primary executable file that is the main entry for other codes. This has to have the format: assignment2_main.rb (where 2 corresponds to the appropriate number in the assignment sequence). This main file should also contain a comment section with a list of the group members if necessary and some description of the runtime requirement of the code (e.g. commandline activation format)

1. A shell program can be regarded as the normal interface between the application programmer and the operating system or real-time kernel. The shell's task is to accept command lines and execute the appropriate command returning error messages as required. This task is repeated indefinitely.

Write your own basic shell utility **– as a reusable component or components.**
Hint: a 'rough' algorithm:

```
loop
        get_command
        create child/worker process
        Parent : wait for worker (child) to finish
        Child: change job
        Parent: report results
end_loop
```

Please note: In our current set-up, your shell program will sit above the current Linux shell. However, you are to assume, okay pretend, that Linux shell does not exist! That is make decisions, and deploy coding strategies appropriate for your shell being the only shell program!

Questions (for pondering and answering)

- Is your problem a class or a module? What is the difference?
- What shell(s) are you using to provide a specification? What features do they support?
- In your opinion, which features are essential (should include in your design) and which are "window dressing" (should not include in your design)?
- Economics: Which, if any, essential features will be omitted from your design due to unmanageable effort requirements?
- 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.
- 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?
- Describe the Ruby exception hierarchy, which classes of exceptions are applicable to this problem?
- What is Module Errno? Is it applicable to the problem? Explain your answer! Remember Ruby often wraps C code.
- 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?
- Should we be using class GetoptLong? Or Regexp? Or shell*?* Or ….
- What environment does a shell run within? Current Directory? Or ….
- What features should be user controllable? Prompts? Input and Output channels? Or ….

2. The control of time in many real-time systems is very important. Further, many real-time systems run on processors with limited capabilities and capacity.

Processes must know how to wait, not only for an event, but also for a certain time period.

Write a reusable component(s) plus a "driver" program, which has two parameters accepted from the command line at the start of execution:

a) An amount of time.
b) A message

Your program should wait for the specified amount of time and then print out the specified message. Meanwhile control of the shell should be returned to the user, i.e. the program must be non-blocking. Finally, your program should minimize the number of processes it creates, if any, as the memory capacity on the target system is expected to be very low.

Questions (for pondering and answering)

- A class or a module?
- Error handling? Robustness? Security? Are any of these required?
- What components of the Ruby exception hierarchy are applicable to this problem? Illustrate your answer.
- Is Module Errno useful in this problem? Illustrate your answer.
- 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.
Do they exist in your Shell solution?

- 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? (**Big hint!**)
- What should be user controllable? Can we trust the user?

3. An important task in any system is to keep track of all files in existence at any point in time. In additional many security systems monitor change information in the file system to help detect erroneous alteration.

To help with this and other related tasks; write a reusable component(s) plus a "driver" program, which monitors files or potential files. The list of all names to be monitored is fully specified at execution time. If the file is altered, created or destroyed, the component should undertake a user-specified action after an optional, but highly accurate, time delay. Note that while these components are 'watching' the user must still be able to enter commands normally to the shell; that is, the system must be non-blocking. Think of this system as a typical batch-processing job that runs indefinitely.

The user-defined actions, and any time delay, must be specific to the type of alteration. That is the user must be able to specify different behaviour for each type of change (altered, created or destroyed). The user must also by able to define which type of alternation to which (potential) file(name) they are interested in watching. Specifically, this component must provide a facility, which supports one of the following *interface protocols* – you are required to decide which one:

FileWatch(type of alteration, duration, list of filenames) {action}

Or

FileWatchCreation(duration, list of filenames) { action}
FileWatchAlter(duration, list of filenames) { action}
FileWatchDestroy(duration, list of filenames) { action}

Or

Give us a better idea – but you will need to be convincing.

Questions (for pondering and answering)

- A class or a module?
- Error handling? Robustness? Security? Are any of these required?
- What components of the Ruby exception hierarchy are applicable to this problem? Illustrate your answer.
- Does this problem require an iterator?
- Describe Java's anonymous inner classes.

- Compare and Contrast Java's anonymous inner classes and Ruby Proc objects; which do you think is better?
- From a cohesion viewpoint, which *interface protocol* is superior? Explain your decision!
- Is Module Errno useful in this problem? Illustrate your answer.
- 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.
- 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.
- 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.
- 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?