SE420, SQA: Exercise #4 – Module Integration and Test

GROUP SQA Assignment

DUE: As specified on Canvas

Please read SQA textbook Chapter 6 and 7 for more background as you wish (optional). Please download GitHub Desktop for Windows, Mac or use on PRClab and check-out Examples-RAID-Unit-Test, Examples-Imageproc-Unit-Test and Examples-Crypto-Unit-Test (checked in by siewertserau) to GitHub public Git repository for source code CMVC. If you need an overview on how to use CMVC and GitHub refer to this presentation.

You may work on this project in a group of 2 to 4 students on a common software module selected in Assignment #4. Please clearly identify all group members on the title page of your submission and indicate what sections of the overall effort each team member worked on [e.g. Analysis, Design, Prototype coding OR on the right-hand side of the V, test plans, design and testing of tests, execution of tests].

The goal of this lab is to gain bottom-up experience extending the functionality and features of a candidate subsystem based on starter example modules and modules you design and code to build a candidate sub-system for early integration and test with the ultimate goal to build a simple file-based RAID, file-based image processing, file-based encryption/decryption application or other open source application of your interest [by Assignment #6].

You should create a GitHub account for your own version of the starter code [for you long-term project to be completed as Assignment #6] and use the GitHub Desktop to manage import of your files and to collaborate if you are working with a partner. Track bugs that you find using GitHub Issues [or manually in Excel if you prefer], but formally and keep a disposition from now through Assignment #6. One of the major goals and outcomes for this assignment is to not only create integrated tests, but to learn how to use CMVC tools, to get your code into GitHub, and to start working with revision control.

Exercise #4 Requirements:

1) [25 points] Get a copy of the RAID, Imageproc, Crypto code **or other full source open source code that you plan to use as a starter unit for your remaining assignments and put it under source control** in your own GitHub repository using GitHub Desktop as described in Notes on Verification Tools Show a screenshot of your repository with imported code, publish it, commit [push] all files, then show a second screenshot with a cloned version of it. Provide your GitHub URL so I can take a look at your repository and code. When you check in your code and when you make revisions to it, please provide comments explaining what you checked in or changed, why, how it was tested and some information so a browser of your code can understand its current status.

2) [60 points] Choose the appropriate exercise for your chosen application:

   a) RAID – Create a new module that can:
      i) [20 points] Open 4 stripe files + 1 XOR parity file and open a 6th input file that you then read and stripe over the 4 stripe files 1 kilobyte at a time + 1 kilobyte of XOR parity until the input file is fully replicated in the 4+1 file set. You can use text files or binary as you see fit. Show in a test that your 5 RAID files are created as you would expect [e.g. each is 1/4th size of input file].
      ii) [20 points] Open the same 4 stripe files and read all data, reconstructing the stripes into a buffer and writing them back to a new 7th file. When done, you should be able to run “diff sixth_input_file.txt seventh_input_file.txt” or “diff sixth_input_file.bin seventh_input_file.bin” and you should get zero differences. Show the results of this test.
      iii) [20 points] Create a new feature so that if one of the 4 data files is missing (moved or deleted), then it can be re-built (regenerated from parity). Create test cases for this that delete any one of the 4 data files or 5th XOR parity and show that it can be re-built.

   b) Imageproc – Create a module that can:
      i) [20 points] Open an input PPM file, run it through a basic transform that enhances the image [e.g. you can use the previously provided brightness and contrast adjustment, but I encourage to consider something new and interesting like a simple negative, gamma correction, or threshold to max/min saturation] and then writes it out to a second PPM file. Demonstrate this code on examples from http://mercury.pr.erau.edu/~siewerts/extra/images/ for at least 3 different images of PPM type [different sizes and resolution]. Show that this works with test cases.
      ii) [20 points] From an input PPM file, run it through a PSF [Point Spread Function] to sharpen the image (see example code for this on our class resource page) and then write out the result to a new PPM file. Demonstrate this code on examples from http://mercury.pr.erau.edu/~siewerts/extra/images/ for at least 3 different images of PPM type [different sizes and resolution]. Show that this works with test cases.
      iii) [20 points] Create a new feature so that you can sequence either transform before the other and show that you can set up a pipeline of these transforms [implemented as a single program or multiple] so the file_1 -> transform_1 -> file_2 -> transform_2 -> file_3. Demonstrate this code on examples from http://mercury.pr.erau.edu/~siewerts/extra/images/ for at least 3 different images of PPM type [different sizes and resolution]. Show that this works with test cases. Comment on and think about whether once could run this pipeline in reverse so that file_3 can be converted back to file_1 as follows: file_3 -> inverse_transform_2 ->

file_2 -> inverse_transfrom_1 -> file_1. Don't implement the inverse pipeline, just think about requirements for this.

c) Crypto – Create a new module that can:
   i) [20 points] Open an input plain text file, run it through at least 3 methods of encryption [e.g. substitution table, transposition table, and substitution hash or transposition hash] to create a new cypher text file and show that it works in reverse as well to create a third file with zero differences from the original input file.
   ii) [20 points] Create a simple pass phrase file [also encrypted] that your application can open, update, create that initially asks a user to create authentication for a file they want to encrypt in the same directory using the method above. If the same user later wants to decrypt this file, challenge them for their pass phrase, authenticate them, and then decrypt the file of their choice. Show that for at least 3 different users, you can encrypt, authenticate, decrypt and show zero difference with the original input. Provide negative tests to show that wrong pass phrase do not work. Show that the original file can be deleted and recovered in plain text with proper authentication.
   iii) [20 points] Create a new feature so that any user can encrypt all files in a directory, delete all originals and replace with encrypted files of the same name, but with ".crpt" appended. If a user can authenticate, decrypt all files and restore original names.

d) **<u>Other open source application</u> – Create a new module that can**:
   i) [20 points] Interface to a GUI, file system, and/or Database.
   ii) [20 points] Provides interactive processing based on user input and/or configuration files.
   iii) [20 points] Create new features to extend the open source to have at least 3 new capabilities and be sure to provide test cases for the new features as well as existing in a new main program that acts as a test driver for the code that will become the product.

3) [15 points] Once your code has been developed, integrated, and tested to your satisfaction, check all updates back into your GitHub project and show me a screen dump with your updates. Also create a list of bugs using GitHub Issues or a simple Excel spreadsheet.

Overall, provide a well-documented professional report of your findings, output, and tests so that it is easy for a colleague (or instructor) to understand what you've done, what worked, what did not and why (even if you can't complete to your satisfaction). Include any C/C++ source code you write (or modify) and Makefiles needed to build your code. I will look at your report first, so it must be well written and clearly address each problem providing clear and concise responses to receive credit, but I will look at your code and test results as well if I have questions.

In this class, you’ll be expected to consult the Linux manual pages and to do some reading and research on your own, so practice this in this first lab and try to answer as many of your own questions as possible, but do come to office hours and ask for help if you get stuck.

Upload all code and your report completed using MS Word or as a PDF to Blackboard and include all source code (ideally example output should be integrated into the report directly, but if not, clearly label in the report and by filename if test and example output is not pasted directly into the report). ***Your code must include a Makefile so I can build your solution on PRClab. Please zip your solution with your last name embedded in the file name.***

Grading Rubric

[25 points] Requirements specification:

[5 pts] download and use of GitHub Desktop____________________________________

[10 pts] establishment of project with starter code______________ __________________

[5 pts] initial check-in______________________________ _________________________

[60 points] Acceptance test specification and prototypes:

[20 pts] basic file transform and tests____

[20 pts] multi-file transform and tests________ ___________________________________

[20 pts] multi-file, multi-feature transform and tests__________________ ____________

[15 points] Code walk-through:

[5 pts] GitHub Push Update ____________________________________

[10 pts] List of known bugs from testing _________________________