Vlad Predovic

CS 362

7/16/2015

Midterm

1. Strategy for finding and correcting bugs:

The first step is to analyze the scope and nature of the beast. We know there are four primary functions and behind these exist interlocked behaviors of the directory tree. It is essential to have a strong understanding of the code to test for with complete code coverage. I would begin by writing robust manual unit tests under the assumption of ‘black box testing’ for these four functions, primarily focusing on testing edge conditions. I would then expand the scope, by using random testing in order to test the system from many different angles at once. The focus of this is integration testing. Any errors found up to this point will be assessed by altering the code and noticing changes in how it runs (regression testing). For any bugs that are difficult to identify slicing methods will be used to locate them. My next steps are to test load and performance of the system. Specifically, how the file system handles a large amount of users and information and whether or not race conditions can be identified. Finally, to finish it is critical to let the customer test the program and gauge the acceptance and ease of use of the program.

2. Fundamental approach and test cases:

My approach consists of starting small and then broadening my scope trying to find more bugs. Then these bugs are removed through the use of isolating and comparing techniques. Finally, the limits of the system are tested by overloading the program.

Unit test pseudocode:	Example:
• Chosen input declared	char[] Blah223$
• Call file system function giving this input	create(Blah223$)
• Assert that correct reaction occurred	Does return == -1?(Can’t have symbols in name)

Random test pseudocode:

- Generate random input
- Generate number
- If number x, run function y with random input….
- Repeat these 3 steps 1000000 times.
- Come back a later and check for bugs.

Slicing pseudocode:

- Locate domain of bug in the code.
- Extract and make sure it still runs correctly.
- Remove everything not involved with the error.
- Run again and analyze for issues.

Regression testing pseudocode:

- Change code in filing system
- Run previous program and current new version.
- If different output, analyze changes and respond accordingly.

To do my performance and load testing I will use a program called Valgrind I would run the following pseudocode off multiple users into the same server for each function and different functions at the same time:

- For n times
- “Do this task(could be int/create/write/delete)
- If successful repeat with n+1(order of magnitude) times.

Valgrind will be used to analyze the performance of your program and it’s response times.

3. Debugging 265 crashing cases that are 500+ operations in length:

Even one case at over 500 operations would be extremely time-consuming to pinpoint and solve the error. I would begin by grouping the test cases where possible. Many might fall under similar areas. To do this I will use two methods. The primary method is based off of delta debugging or using the binary search algorithm on long operations to isolate where the errors exist and then manually attack these points. This will save a lot of time but it is also an automated debugging method and could fail at finding the cause for certain bugs. At this point, the use of static program slicing comes into play. A manual analysis of isolated or ‘sliced’ parts of code that affect the variable or point where we have narrowed down the known location of the bug will further reduce the time required to locate and fix errors.

4. Different approach if we did not have the source code:

Our specific case only supports a small set of operations narrowing the scope of design. Given no source code, differential testing would be effective primarily because of the popularity of file systems. Every company uses them, and generic file systems exist practically anywhere computer systems are involved. Due to this, it would not be difficult to create or find source code for what we would expect. This method would make it easy to notice differences which might otherwise go by unnoticed. Finally, load testing would be used on the set of operations provided because a file system could be accessed and manipulated constantly by many users.