LINGI2251
Software Quality Assurance
Assignment 3
Spring 2021

Charles Pecheur

Assignment due Monday May 17, 2021

This assignment can be performed in **groups of two students**. Individual submissions are also accepted. Interaction between students and groups is allowed but plagiarism will not be tolerated. Results will be submitted in the *Assignments* section of the Moodle website, before the deadline stated above. Late submissions will not be accepted.

Please post any question regarding this assignment on the corresponding forum on the course website. For personal issues you may contact *Charles Pecheur*.

Subject

Daisy: a Simple File System

Daisy is a prototype implementation in Java of a NFS UNIX-like filesystem. The class `daisy.Inode` represents an internal file node; the class `daisy.Petal` represents the disk. The class `daisy.Daisy` and `daisy.DaisyDir` provide functions to access files and directories, respectively. Note that those classes are static function containers, not instantiable object classes.

A test program is provided by `DaisyTest.main()`. This method spawns several concurrent user threads, as instances of `DaisyUserThread`, that perform file operations. Note that `DaisyUserThread.run()` contain assertions that check the consistency between the result of file lookup and file operations.

The provided code has concurrency bugs: the assertions may be violated. The objective of the mission is to reveal those errors and verify the program

using Java PathFinder (JPF). Of course, the goal is not merely to identify and report the errors (which you might possibly find by careful scrutiny), but to demonstrate how JPF was used to unveil and diagnose these errors.

Instructions

The Java source code is provided on the course website. The test code has parameters for the number of files (`FILECOUNT`) and the number of operations performed by each user (`ITERATIONS`); both are set to 1 as provided. There are two user threads, performing a read and a delete operation, respectively.

Compile the source code and execute JPF on the source code using the provided `daisy.jpf` configuration file.

1. By analyzing the provided test code, express analytically the number of different *concurrent test cases* that the `main()` method will produce, as a function of `FILECOUNT`, `ITERATIONS`, the number of user threads and the operations performed (either fixed or random). A concurrent test case is defined as a combination of single sequential executions of each thread, considering explicit choices but ignoring variants due to scheduling. For example, for three threads with two binary choices each, the number would be $(2 \times 2)^3 = 64$. (The number of paths and states explored will of course be much larger, due to all the possible interleavings for each test case.)

2. Execute JPF with the parameters as provided. You should find an assertion violation. Measure and report statistics (time, memory, numbers of states).

3. Execute JPF again with a depth limit (`search.depth_limit=xxx`) to obtain a shorter error trace. Experiment with different depths to get the shortest possible trace. Report statistics on your attempts (depth limit, time, number of states, trace length).

4. Analyze the trace, identify the source of the problem, and propose a fix in the program (this could be on class `FileHandle`).

5. Execute JPF on your corrected program (without depth limit). You should find a second assertion violation. Analyze the trace to identify the source of the problem. In this case, the fix should consist in removing or disabling some assertions in `DaisyUserThread` that are actually not relevent. Measure and report statistics (time, memory, numbers of states).

6. Execute JPF on your newly corrected program. You should find no more errors. Measure and report statistics (time, memory, numbers of states) for the full state-space exploration. The time should be a couple of minutes (on INGI lab machines).

7. Execute JPF again with a depth limit of 100. Modify the test code to each of the following conditions:
   (a) thread 2 performs a write operation,
   (b) thread 2 performs a random operation,
   (c) thread 2 performs two random operations,
   (d) there are two files,
   (e) there are three files,
   (f) an additional thread 3 performs a write operation.
   For each case, measure and report statistics (time, memory, numbers of states). You can abort a verification that takes longer than 10 minutes. Discuss whether your measurements are consistent with the analytic prediction established in the first point.

Your report should be no longer than six pages and be self-contained; this assignment will be evaluated based on the report alone. In particular, include any relevant fragment of the code where necessary to illustrate your work. You can assume that the initially provided program is known.

Tips

- The test harness uses methods from the `Random` class; do not forget to activate random enumeration in JPF (option `-random` or config `cg.enumerate_random=true`).

- To allow JPF to include source code in error traces, set the `sourcepath` config parameter to the path to your source, and `classpath` to your compiled classes.

Deliverables

Assignment deliverables will be submitted as an archive file (`.zip` or `.gz`) whose base name is the last names of the students (e.g. `Pecheur_Martin.zip`, no accents please). Please submit your deliverables on the Moodle course website. Late submissions will not be accepted. The archive should at least contain:

- the report in PDF format,
- all modified Java source files needed to reproduce your results,
- a shell script that reproduces the reported results.

Make sure to **include proper identification** (course, year, assignment number, student names) at the beginning of all documents.