AutoCheck - Simple automated marking

June 16, 2016

1 Aims and Objectives

1. Create a system to enable automated checking of student's practicals in various courses.
2. Focus initially on CS1001, 1002, 2001 and 2002.
3. Students will be able to run a subset of the tests.
4. A report will be generated for every student automatically, from a student submission.

Not aims and objectives:

1. Not automating everything (such as GUIs).
2. Will not 100% protect against malicious students (will be much better than current situation, where many staff run student's codes on their own machines, as themselves).
3. Will not provide a server-based architecture which provides students with continuous feedback.
   We might do this in a later, but the server capacity/maintenance required is excessive. If such a server only ran daily / twice daily it would not give students feedback fast enough, and if it ran continuously students may overload it / complain if it breaks before deadlines.

2 Design Principles

1. Transparency
   It should be as clear how the tester is running a particular test.

2. Extensibility
   The tester should be easy to extend with new types of tests.

3. Simplicity
   Writing tests (particularly of common existing types) should be as simple as possible.

4. Fallback
   If the tester / a server explodes, we should be in no worse position than this year.

3 System Design

The system can be thought of as three major components:

tests: The actual tests which are to be run for each practical.

Student runner: A method for students to quickly run the tests on their practical, and get feedback.

MMS runner: A method of running tests on a large number of students.

3.1 Tests

All tests will fundamentally be a **shell script**. This is (I believe) the only option which can meet the requirements for transparency, extensibility and simplicity.

Each test will be by default be a single shell script which returns a value based on if the test passes or fails. The tester will run each of these tests in turn, and return their result, and the script's output (if any) for inspection.

A mercurial repository will be set up to share "best practices" in writing tests, in particular including some common kinds of tests (such as running a program with a given stdin / arguments, and checking the output).

3.2 Student Runner

The student runner will run a set of tests on a student's current submissions. These tests will accept a zip file, as could be submitted to MMS, and run the tests.

These tests will be run by default on a host server, under the student's own account. Students who choose to copy the scripts to their own machines do so at their own risk.

This introduces some slight dependencies (student's code may work in their account, but not when submitted), but removes the need to provide students access to virtual machines (which is, I am told, prohibitively difficult).

3.3 MMS Runner

Automatically run tests for all students, based on a download from MMS.

This tester will take a full set of files downloaded from MMS, and run the tester on all of them. This will be done within virtual machines, using vagrant.

4 Requirements

4.1 Physical Requirements

1. A virtual machine image which closely matches the lab/host servers. This will be used for running student's code in an environment which closely matches where the will be testing.

2. Servers for running the MMS Runner. Assume 3GB ram per student. It would be nice to be able to cycle the whole set in 1 hour. Assuming 5 minutes per student, that is 10 cores. These numbers are initial figures.

4.2 Software Requirements

1. Maintain Student runners and MMS runners.

2. Write practical testers.
   Fundamentally, writing practical testers is the **lecturer's responsibility**. In particular, there are some practicals which will be almost impossible to test.

5 Low Level Issues

This is a collection of low level issues I have observed during initial design.

1. intelliJ is very hard to control from the command line, so accepting intelliJ projects for automatic checking may be difficult.

2. Students who submit binaries in CS2002 practicals with Makefiles cause problems, as Make is too stupid to realise it needs to delete the mac binary and build a linux one. We could try to auto-detect this mistake and clean it up.

3. Should the tester for students work directly in their source directory, or require a zip?