Submitty: An Open Source, Highly-Configurable Platform for Grading of Programming Assignments

Matthew Peveler, Jeramey Tyler, Samuel Breese,
Barbara Cutler, and Ana Milanova

Rensselaer Polytechnic Institute

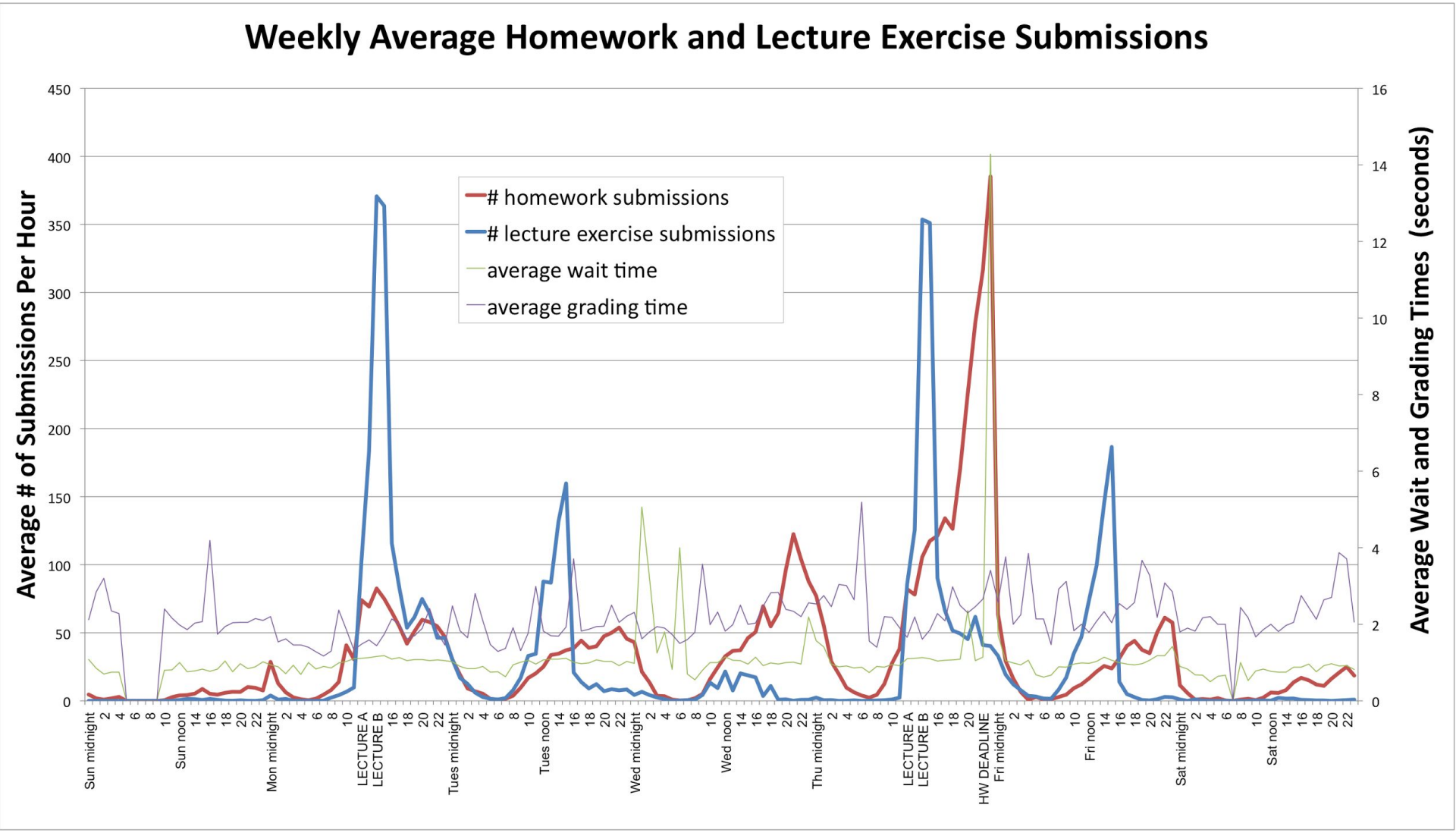
Submitty Demo

- Handout: username/password for demo installation
- Today's Demo:
  - What is Submitty?
  - Student, TA, and Instructor views of system
  - 12 example of autograding configs
  - Rainbow Grades
  - Current Work and Future Goals
- All material from this demo available at https://submitty.org/tutorial
- Feel free to ask questions throughout or email us! submitty@cs.rpi.edu

What is Submitty?

- Students upload code (and resubmit) for auto-grading
- TAs review and add additional grading/feedback
- Configurable number of late days per gradeable
- Open-source, free to use
- Installed on your own hardware or VPS
  - Instructors have ssh access to files & logs for debugging
- Support for any language / tool installed on your server
  - We use Python, C, C++, Java, Scheme, Prolog, and SPIM
  - JUnit, Emma code coverage, Dr Memory, static analysis
- Supports dozen of courses with thousands of users at RPI
  - 650+ students in Computer Science I
  - 450+ students in Data Structures
  - > 100 students in many of our junior/senior courses

Server Performance



RENSSELAER POLYTECHNIC INSTITUTE
CS
COMPUTER SCIENCE
Submitty
Rensselaer Center for Open Source

Demo!

2 / 2 **Test 6 Double Root** `1 6 9` Details

Student STDOUT.txt

```
1  Enter 3 integer coefficients to a quadratic function: a*x*x + b*x + c = 0
2  The roots are: -3 and -3
3
```

Expected STDOUT.txt

```
1  Enter 3 integer coefficients to a quadratic function: a*x*x + b*x + c = 0
2  The roots are: -3 and -3
3
```

2 / 2 **Test 7 Zero Root** `1 4 0` Details

1 / 3 **Test 8 a != 1** `2 7 3` Details

Student STDOUT.txt

```
1  Enter 3 integer coefficients to a quadratic function: a*x*x + b*x + c = 0
2  The roots are: -2 and -12
3
```

Expected STDOUT.txt

```
1  Enter 3 integer coefficients to a quadratic function: a*x*x + b*x + c = 0
2  The roots are: -0.5 and -3
3
```

Standard Error (STDERR)

WARNING: This file should be empty

```
1  ERROR:  -2 is not a root of this formula.
2  ERROR:  Unable to verify one or both roots.
3
```

Execution Logfile

```
1  Child exited with status = 1
2
```

examples/01_simple_python


```
{
  "testcases" : [
    {
      // Student-visible testcase name.
      "title" : "Python - Simple Grading",

      // Commands to run (in order). These are not shell commands, although
      // they support some common shell wildcards. This can either be a
      // list or a single string.
      "command" : [ "python *.py" ],

      // Point value of this testcase.
      "points" : 10,

      "validation" : [
          {
              // Grade by "diffing" the student output with an
              // instructor-provided file.
              "method" : "myersDiffbyLinebyChar",
              // The student's output. Corresponds to the position
              // of a command in the "command" list above: since
              // "python *.py" is the first command, it's output
              // will be written to "STDOUT_0.txt".
              "actual_file" : "STDOUT.txt",
              // The title seen by students.
              "description" : "Program Output",
              // The instructor-provided file (the correct answer).
              "expected_file" : "output.txt"
          }
      ]
    }
  ]
}
```

examples/02_simple_cpp


```
{
  // For compiled languages, typically two testcases are used to allow points
  // to be asssigned independently for compilation and execution.
  "testcases" : [
    {
      // Indicate that this is a compilation step.
      "type" : "Compilation",
      "title" : "C++ - Compilation",
      "command" : "clang++ -Wall -o a.out -- *.cpp",
      // Name of the result of compilation.
      "executable_name" : "a.out",
      // Point value of compilation.
      "points" : 5
    },
    {
      "title" : "C++ - Execution",
      "command" : "./a.out",
      // Point value of correct output.
      "points" : 15,
      "validation" : [
        {
          "method" : "myersDiffbyLinebyChar",
          "actual_file" : "STDOUT.txt",
          "description" : "Program Output",
          "expected_file" : "test1_output.txt"
        }
      ]
    }
  ]
}
```

New submission for: Python Simple Homework Multipart

Submit each part of your homework to the right bucket or you will not receive credit.

Drag your Part 1 here or click to open file browser	Drag your Part 2 here or click to open file browser	Drag your Part 3 here or click to open file browser
part1.py 0.10kb	part2_wrong_output.py 0.20kb	

By clicking "Submit" you are confirming that you have read, understand, and agree to follow the Academic Integrity Policy.

Submit | Clear | Get Most Recent Files

Select Submission Version: Version #1 Score: 4 / 10 GRADE THIS VERSION

Do Not Grade This Assignment

Note: This version of your assignment will be graded by the instructor/TAs and the score recorded in the gradebook.

Submitted Files

part1/part1.py (0.10kb)
part2/part2_wrong_output.py (0.20kb)

submission timestamp: 03/09/2017 08:50:51 PM
days late: 0 (before extensions)
grading time: 1 seconds
queue wait time: 1 seconds

Results

4 / 10	**Total**	
3 / 3	**Test 1 Part 1 Compute square root**	Details
1 / 4	**Test 2 Part 2 Solve for x^2 + 5x + 6 = 0**	Details
0 / 3	**Test 3 Part 3 Count from 1 to 10**	Details

examples/03_multipart


```
{
  // Instructors can create multi-part assignments.
  // Each part will appear as a different submission box.
  "part_names" : [ "Part 1", "Part 2", "Part 3" ],

  // Submissions for each part are just placed in the part1, part2, etc.
  // directories. From there, they can be graded in the same manner as any
  // other submission.
  "testcases" : [
    {
      "title" : "Python - Part 1",
      "command" : "python part1/*.py",
      "points" : 3,
      "validation" : [
        {
          "method" : "myersDiffbyLinebyChar",
          "actual_file" : "STDOUT.txt",
          "description" : "Program Output",
          "expected_file" : "part1_sol.txt"
        }
      ]
    },
    {
      "title" : "Python - Part 2",
      "command" : "python part2/*.py",
      "points" : 4,
      "validation" : [
        {
          "method" : "myersDiffbyLinebyChar",
          "actual_file" : "STDOUT.txt",
          "description" : "Program Output",
          "expected_file" : "part2_sol.txt"
        }
      ]
```

Incorporating Static Analysis

- Very large enrollments in our CS1
- Impractical to have TAs manually verify that students use specific new language syntax

```
length = 5
width = 16.5
height = 12.5
volume = length * width * height
area = 2*length*width + 2*length*height + 2*width*height
print 'volume =', volume
print 'area =', area
```

Correct!

```
volume = 5 * 16.5 * 12.5
area =  722.5
print 'volume =', volume
print 'area =', area
```

Wrong!

examples/04_python_static_analysis


```
"testcases" : [
  {
    "title" : "Python - Static Analysis",

    // Here, multiple commands are provided (to be executed in sequence).
    // The first is to execute the student code, the second is to count the number of calls to th
    // "print" function.
    "command" : [ "python *.py",
                  "submitty_count_node -l python2 assign *.py",
                  "submitty_count_node -l python2 mul *.py",
                  "submitty_count_node -l python2 add *.py" ],
    "points" : 4,
    "validation" : [
      // First, ensure that the student received the correct answer.
      {
        "method" : "myersDiffbyLinebyChar",
        "actual_file" : "STDOUT_0.txt",
        "description" : "Program Output",
        "expected_file" : "output.txt"
      },
      {
        // Grade by comparing the contents of a file to a given integer.
        "method" : "intComparison",
        // Use the output of the second command.
        "actual_file" : "STDOUT_1.txt",
        "description" : "Number of assignments",

        // The method by which to compare the output of "submitty_count_node"
        // against the provided term (here, "greater-than-or-equal" is used).
        "comparison" : "ge",
        // The integer against which to compare.
        "term" : 5,

        // Message to the student.
        "failure_message" : "Re-read the problem instructions.",
        // Only display a message if the test failed.
        "show_message" : "on_failure",
        // Hide the student/instructor comparison.
        "show_actual" : "never"
      },
      {
        "method" : "intComparison",
```

examples/05_cpp_static_analysis


```
"testcases" : [
  {
    // Static analysis can also be performed upon C++ code.
    // Here, no compilation is performed at all; instead, student code is examined to
    // ensure that it does not use the "goto" keyword.
    "title" : "C++ - Check for goto",
    "type" : "Compilation",
    "command" : [ "submitty_count_token -l c goto *.cpp" ],
    "points" : 10,
    "validation" : [
      {
        "method" : "intComparison",
        "actual_file" : "STDOUT.txt",
        "description" : "Number of `goto`",
        "comparison" : "eq",
        "term" : 0,
        "failure_message" : "You must not use the `goto` keyword.",
        "show_message" : "on_failure",
        "show_actual" : "never"
      }
    ]
  }
]
```

Original code:

```
# submission for assignment 1
for x in range(0,3):
    print(x)
```

Correct student rewrite:

```
# submission for assignment 1
x = 0
while x < 3:
    print(x)
    x += 1
```

examples/06_loop_types


```
"testcases" : [
  {
    "title" : "Python - Distinguish for and while Loops",
    "command" : [ "submitty_count_node for *.py",
                  "submitty_count_token while *.py" ],
    "points" : 2,
    "validation" : [
      {
        "method" : "intComparison",
        "actual_file" : "STDOUT_0.txt",
        "description" : "Number of for loops",
        "comparison" : "eq",
        "term" : 0,
        "failure_message" : "Must not use for loops",
        "show_message" : "on_failure",
        "show_actual" : "never"
      },
      {
        "method" : "intComparison",
        "actual_file" : "STDOUT_1.txt",
        "description" : "Number of while loops",
        "comparison" : "ge",
        "term" : 1,
        "failure_message" : "Must use a while loop",
        "show_message" : "on_failure",
        "show_actual" : "never"
      }
    ]
  }
```

Custom Static Analysis


```
for x in [1, 2, 3]:
    print(x)
for x in [1, 2, 3]:
    print(x)
```

Loop depth=1

```
for x in [1, 2, 3]:
    for y in range(0, x):
        print(y)
    for z in [1, 2, 3]:
        print(z)
```

Loop depth=2

examples/07_loop_depth


```
import lang.parser
paths = lang.parser.leaf_paths(
    lang.parser.python(open("student.py").read()))
print(max([len([x for x in path if x.name in ["for", "while"]])
           for path in paths]))
```

```
"testcases" : [
  {
    "title" : "Python - Determine Loop Depth",

    // Here, an instructor-provided static analysis script is used, rather
    // than one of the provided scripts like count_token and count_function.
    // This works in much the same way as those scripts.
    "command" : [ "mv loopdepth.py loopdepth",
                  "python3 loopdepth *.py" ],
    "points" : 10,
    "validation" : [
      {
        "method" : "intComparison",
        "actual_file" : "STDOUT_1.txt",
        "description" : "Loop Depth",

        "comparison" : "le",
        "term" : 3,

        "failure_message" : "Must have less than four nested loops",
        "show_message" : "always",
        "show_actual" : "always"
      }
    ]
  }
]
```

C/C++ Memory Debugging

- Common mistakes made by both new and experienced programmers:
  - Reading uninitialized memory
  - Reading/writing beyond the bounds of an array
  - Memory leaks
  - Segmentation faults
- Causing mysterious crashes or output that are hard to find with print statements or a traditional debugger

Use an External Tool like Dr Memory or Valgrind!

examples/08_memory_debugging


```
{
  "testcases" : [
    // Grading of C++ code can also be supplemented with the use of a memory
    // debugger. Here, the tool Dr. Memory is used to penalize student code
    // containing memory errors.
    {
      "type" : "Compilation",
      "title" : "C++ - Compilation",
      "command" : "clang++ -m32 -g -Wall -o a.out *.cpp",
      "executable_name" : "a.out",
      "points" : 5
    },
    {
      "title" : "C++ - Memory Debugger",
      "command" : "drmemory -brief -- ./a.out",
      "points" : 5,
      "validation" : [
        {
          "method" : "warnIfEmpty",
          "actual_file" : "STDOUT.txt",
          "description" : "Standard Output (STDOUT)"
        },
        {
          "method" : "DrMemoryGrader",
          "actual_file" : "STDERR.txt",
          "description" : "Standard Error (STDERR)",
          "deduction" : 1.0
        }
      ]
    }
  ]
}
```

examples/09_java_testing


```
"testcases" : [
  // The Submitty system supports Java, and Java code can be graded by
  // running JUnit tests.
  {
    "type" : "Compilation",
    "title" : "Java - Compilation",
    "command" : "javac -cp submitty_junit.jar Factorial.java",
    "executable_name" : "Factorial.class",
    "points" : 2
  },
  {
    "type" : "Compilation",
    "title" : "Java - Instructor JUnit Test Compilation",
    "command" : "javac -cp submitty_junit.jar:. FactorialTest.java",
    "executable_name" : "FactorialTest.class",
    "points" : 2
  },
  {
    "title" : "Java - JUnit Test",
    "command" : "java -cp submitty_junit.jar:submitty_hamcrest.jar:submitty_emma.jar:. org.junit.
    "points" : 16,
    "validation" : [
      {
        "method" : "JUnitTestGrader",
        "actual_file" : "STDOUT.txt",
        "num_tests" : 4
      }
    ]
  }
]
```

examples/10_java_coverage


```
    {
      "title" : "Java - Running Student JUnit Tests in hw0/tests/",
      "command" : "java -noverify -cp submitty_junit.jar:submitty_hamcrest.jar:submitty_emma.jar:submitty_junit/:. TestRunner hw0",
      "points" : 4,
      "validation" : [
          {
              "method" : "MultipleJUnitTestGrader",
              "actual_file" : "STDOUT.txt"
          }
      ]
    },

    {
      "title" : "Java - Generating Coverage Report for Student Tests",
      "command" : "java -cp submitty_emma.jar emma report -r txt -in coverage.em,coverage.ec -Dreport.txt.out.file=emma_report.txt",
      "points" : 6,
      "validation" : [
        {
          "method" : "errorIfEmpty",
          "actual_file" : "STDOUT.txt",
          "description" : "EclEmma report generation output",
          "deduction" : 0.0
        },
        {
          "method" : "EmmaCoverageReportGrader",
          "actual_file" : "emma_report.txt",
          "coverage_threshold" : 90,
          "deduction" : 1.0
        }
      ]
    },

    {
      "title" : "Java - Instructor JUnit Tests",
      "command" : "java -noverify -cp submitty_junit.jar:submitty_hamcrest.jar:submitty_emma.jar:. org.junit.runner.JUnitCore hw0.test.FactorialTest",
      "points" : 6,
      "validation" : [
        {
          "method" : "JUnitTestGrader",
          "actual_file" : "STDOUT.txt",
          "num_tests" : 4
        }
      ]
    }
  ]
}
```

examples/11_resources


```
{
  "resource_limits" : {
      // Allow the submission to run for 100 seconds.
      "RLIMIT_CPU" : 100
  },

  "testcases" : [
    {
      "type" : "Compilation",
      "title" : "C++ - Compilation",
      "command" : "clang++ -Wall -o a.out -- *.cpp",
      "executable_name" : "a.out",
      "points" : 5
    },
    {
      "title" : "C++ - Execution",
      "command" : "./a.out",
      "points" : 4,
      "validation" : [
        {
            "method" : "myersDiffbyLinebyChar",
            "actual_file" : "STDOUT.txt",
            "description" : "Program Output",
            "expected_file" : "test1_output.txt"
        }
      ]
    }
  ]
}
```

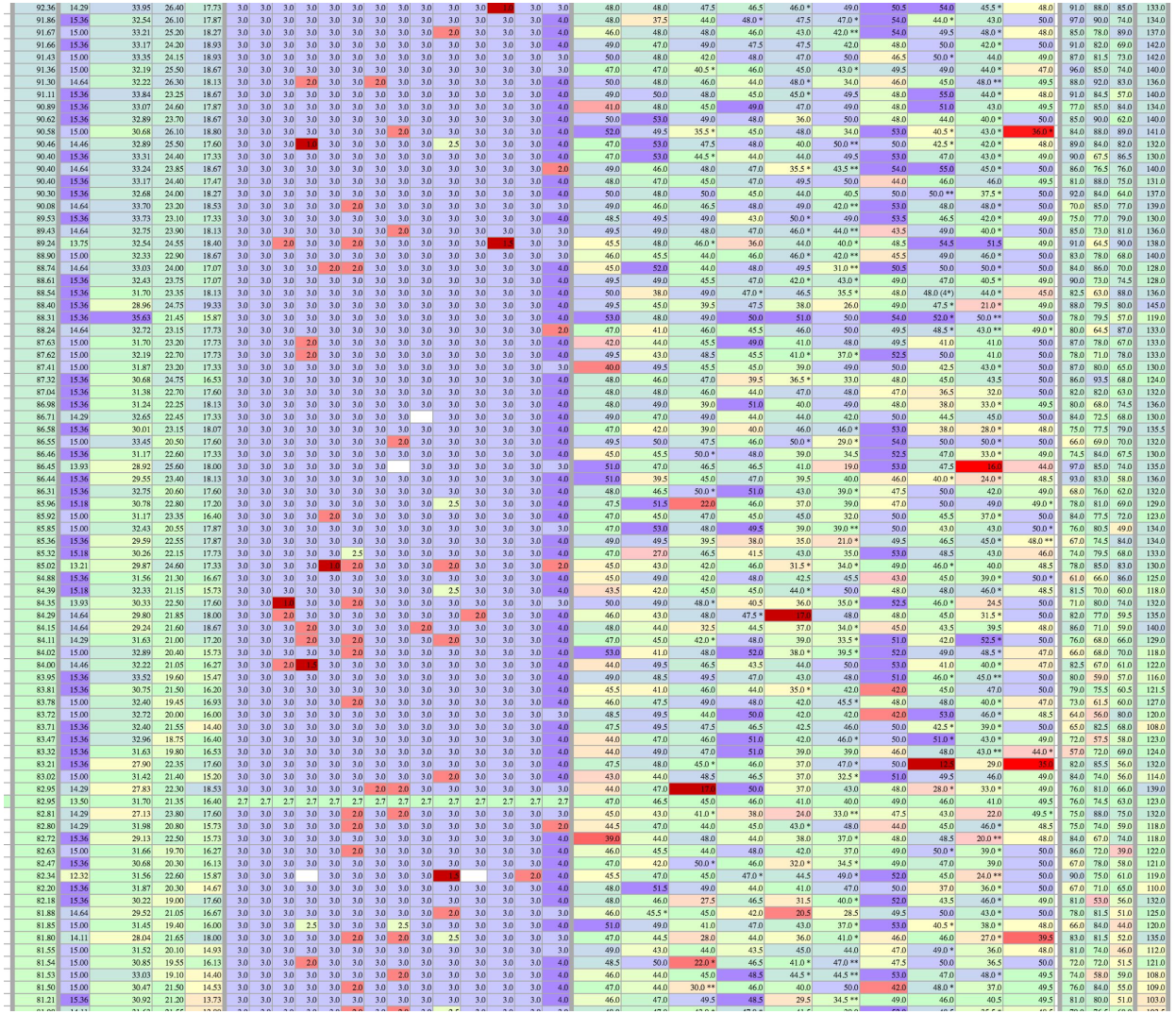
examples/12_system_calls

```
{
  "resource_limits" : {
    // Allow the submission to run for 10 seconds.
    "RLIMIT_CPU" : 10,
    // Allow up to 20 additional processes launched by the student code.
    "RLIMIT_NPROC" : 20
  },

  // Allow the student code to use IPC and multiprocessing system calls.
  "allow_system_calls" : [
    // This allows us to use ipc, pipe, semop, shmat, shmtcl, ...
    "ALLOW_SYSTEM_CALL_CATEGORY_COMMUNICATIONS_AND_NETWORKING_INTERPROCESS_COMMUNICATION",
    // This allows us to use clone, execve, fork, set_tid_address, vfork
    "ALLOW_SYSTEM_CALL_CATEGORY_PROCESS_CONTROL_NEW_PROCESS_THREAD"
  ],

  "testcases" : [
    {
      "type" : "Compilation",
      "title" : "C - Compilation",
      "command" : "/usr/bin/gcc -Wall -o a.out *.c",
      "executable_name" : "a.out",
      "points" : 2
    },
    {
      "title" : "C - Execution",
      "command" : "./a.out 10",
      "points" : 4,
      "validation" : [
        {
          "method" : "searchToken",
          "data" : [ "ALL DONE! 10 successful forks" ],
          "actual_file" : "STDOUT.txt",
          "description" : "Standard Output (STDOUT)",
          "deduction" : 1.0
        }
      ]
    }
  ]
}
```

Iris - Rainbow Grades

Submitty
Rensselaer Center for Open Source

Instructor view



Student view

USERNAME		PERFECT	LOWEST A-	LOWEST B-	LOWEST C-
FIRST			approximate	approximate	approximate
LAST					
OVERALL	75.52	100.00	83.90	73.33	58.93
FINAL GRADE	B				
LAB %	14.11	15.00	13.50	12.00	10.50
HOMEWORK %	31.31	35.00	31.70	27.59	20.59
TEST %	16.10	30.00	22.30	19.21	15.57
EXAM %	14.00	20.00	16.40	14.53	12.27
Lab 1	3.0	3.0	2.7	2.4	2.1
Lab 2	3.0	3.0	2.7	2.4	2.1
Lab 3	3.0	3.0	2.7	2.4	2.1
Lab 4	2.0	3.0	2.7	2.4	2.1
Lab 5	3.0	3.0	2.7	2.4	2.1
Lab 6	2.0	3.0	2.7	2.4	2.1
Lab 7	3.0	3.0	2.7	2.4	2.1
Lab 8	3.0	3.0	2.7	2.4	2.1
Lab 9	3.0	3.0	2.7	2.4	2.1
Lab 10	2.5	3.0	2.7	2.4	2.1
Lab 11	3.0	3.0	2.7	2.4	2.1
Lab 12	3.0	3.0	2.7	2.4	2.1
Lab 13	3.0	3.0	2.7	2.4	2.1
Lab 14	3.0	3.0	2.7	2.4	2.1
Homework 1	47.0	50.0	47.0	45.5	42.0
Homework 2	42.0	50.0	46.5	42.0	27.0
Homework 3	47.0 *	50.0	45.0	34.0	24.0
Homework 4	44.5	48.0	46.0	42.5	33.0
Homework 5	39.0	50.0	41.0	35.5	22.0
Homework 6	27.0 *	50.0	40.0	27.0	15.0
Homework 7	53.0	50.0	49.0	47.0	43.0
Homework 8	48.5	50.0	46.0	40.5	24.0
Homework 9	30.5 *	50.0	41.0	30.5	19.5
Homework 10	48.0	50.0	49.5	48.0	43.5
Homework 11 Extra Credit	19.0				
Test 1	57.0 *M1*	100.0	76.0	66.0	53.0
Test 2	52.0 *M13*	100.0	74.5	64.0	53.0
Test 3	34.0 *B1*	100.0	63.0	51.5	38.0
Exam 1	105.0 *N6*	150.0	123.0	109.0	92.0

Current / Future Projects

- Expand usage of Submitty beyond RPI
- Open source plagiarism detection system
- Support SQL autograding
- Peer/Group grading
- Integration with Github repos
- PDF/Code annotation for TA grading

Thanks!

- Submitty Website
  http://submitty.org

- Demo Site
  http://submitty.org/tutorial

- GitHub organization
  https://github.com/Submitty

- Email us!
  submitty@cs.rpi.edu

Open Source Software

- Advantages over proprietary software:
  - Free to use
  - No subscription required
  - No third-party data collection
- Open source can be more reliable than closed source software
- User interface can be improved and customized
- Community helps maintain codebase
- Improved security: More people can study and test the software to find bugs and security vulnerabilities

Security

- Database access done through the PHP Data Objects (PDO) library which protects against malicious and malformed inputs
- Instructor configures appropriate resource limits (GNU Linux `rlimit`) to sandbox testing of electronically-submitted student code and prevent issues like infinite loops, runaway output, and excessive use of other system resources
- Before running the student code, we switch from a privileged system user to an untrusted user using GNU Linux `setresuid`
- Careful design of file and directory permissions and database access maintains confidentiality of student work and grades
- Uses secure computing mode (GNU Linux `seccomp`) to prevent use of sockets, fork, and other unnecessary system calls by student code

Thanks to RPISEC (our undergraduate Computer Security Club) for helping find & patch vulnerabilities