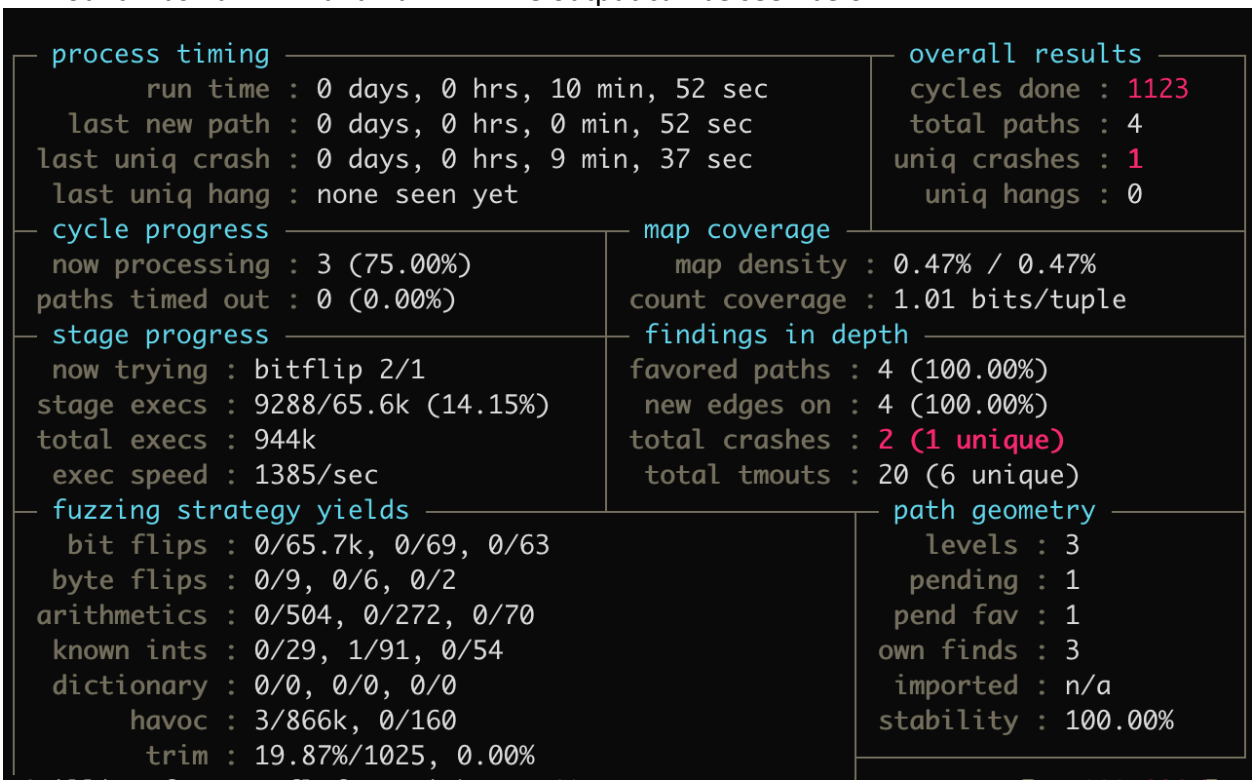
For this assignment, I started off with writing a program where symbolic execution wins. For symexwins.cpp, I used two integers generated by Deepstate and checked if 5 * num1 + 3 == num2. I chose this equality because it will take longer for fuzzers to generate input that causes this to be true. As for symbolic execution, it should be very quick since it is a very small math problem to solve and the symbolic execution would solve the same way as humans do. First, I ran deepstate-angr to see how symbolic execution would do, and my intuition was correct. Deepstate-angr ran for about 6 milliseconds, as seen by the output below:

```
user@d378fd5af966:~/kdl222_cs499_sp20/assignment3$ deepstate-angr ./symexwins
INFO     | 2020-04-13 20:07:31,559 | deepstate.angr | Running 1 tests across 1 workers
CRITICAL | 2020-04-13 20:07:37,820 | deepstate | symexwins.cpp(20): Checked condition
ERROR    | 2020-04-13 20:07:37,820 | deepstate | Failed: Symexwins_StrTest
INFO     | 2020-04-13 20:07:37,821 | deepstate | Saved test case in file out/symexwins.cpp/Symexwins_StrTest/2db38db099c1a031d20ced3a0505e3e6.fail
```

The input that angr found was num1=1 and num2=8.

Next, I tried to run AFL. Again, my intuition was correct. It took AFL about 4 minutes to find two numbers that caused the assert to fail, even though I let it run for 10 minutes. The input that AFL found was num1=-1 and num=-2. The output can be seen below:



Next, I let LibFuzzer run for 18 minutes and gave it a corpus directory to save the failed input. In that time, it found 2 failing inputs:

- Num1=721420287, num2=-16777216
- Num1=-6710887, num2=-33554432

I am not 100% sure when the first failure was found, but it definitely took longer than angr to find.

Finally, I ran the executable with –fuzz. I expect this fuzzer to take longer to find an input that causes the test to fail because of the fuzzer being very “basic” and not as “smart” when it generates random input. Again, my inuition was correct. 820913 tests were ran but all of them passed. I am not surprised that no failures were found since it would be difficult to make the test fail accidentally. The output can be seen below:

```
user@d378fd5af966:~/kdl222_cs499_sp20/assignment3$ ./symexwins --fuzz -o fuzz_out/ --timeout 600
INFO: Starting fuzzing
WARNING: No seed provided; using 1586814517
WARNING: No test specified, defaulting to first test defined (Symexwins_StrTest)
INFO: 2020-04-13 21:49:07: 1568 tests/second: 0 failed/47044 passed/0 abandoned
INFO: 2020-04-13 21:49:37: 1597 tests/second: 0 failed/95866 passed/0 abandoned
INFO: 2020-04-13 21:50:07: 1596 tests/second: 0 failed/143683 passed/0 abandoned
INFO: 2020-04-13 21:50:37: 1577 tests/second: 0 failed/189277 passed/0 abandoned
INFO: 2020-04-13 21:51:07: 1581 tests/second: 0 failed/237170 passed/0 abandoned
INFO: 2020-04-13 21:51:37: 1562 tests/second: 0 failed/281180 passed/0 abandoned
INFO: 2020-04-13 21:52:07: 1553 tests/second: 0 failed/326261 passed/0 abandoned
INFO: 2020-04-13 21:52:37: 1513 tests/second: 0 failed/363196 passed/0 abandoned
INFO: 2020-04-13 21:53:07: 1479 tests/second: 0 failed/399417 passed/0 abandoned
INFO: 2020-04-13 21:53:37: 1452 tests/second: 0 failed/435885 passed/0 abandoned
INFO: 2020-04-13 21:54:07: 1406 tests/second: 0 failed/464081 passed/0 abandoned
INFO: 2020-04-13 21:54:37: 1407 tests/second: 0 failed/506736 passed/0 abandoned
INFO: 2020-04-13 21:55:07: 1391 tests/second: 0 failed/542501 passed/0 abandoned
INFO: 2020-04-13 21:55:37: 1367 tests/second: 0 failed/574144 passed/0 abandoned
INFO: 2020-04-13 21:56:07: 1354 tests/second: 0 failed/609726 passed/0 abandoned
INFO: 2020-04-13 21:56:37: 1354 tests/second: 0 failed/650124 passed/0 abandoned
INFO: 2020-04-13 21:57:07: 1358 tests/second: 0 failed/692799 passed/0 abandoned
INFO: 2020-04-13 21:57:37: 1366 tests/second: 0 failed/737734 passed/0 abandoned
INFO: 2020-04-13 21:58:07: 1368 tests/second: 0 failed/780042 passed/0 abandoned
INFO: Done fuzzing! Ran 820913 tests (1368 tests/second) with 0 failed/820913 passed/0 abandoned tests
```

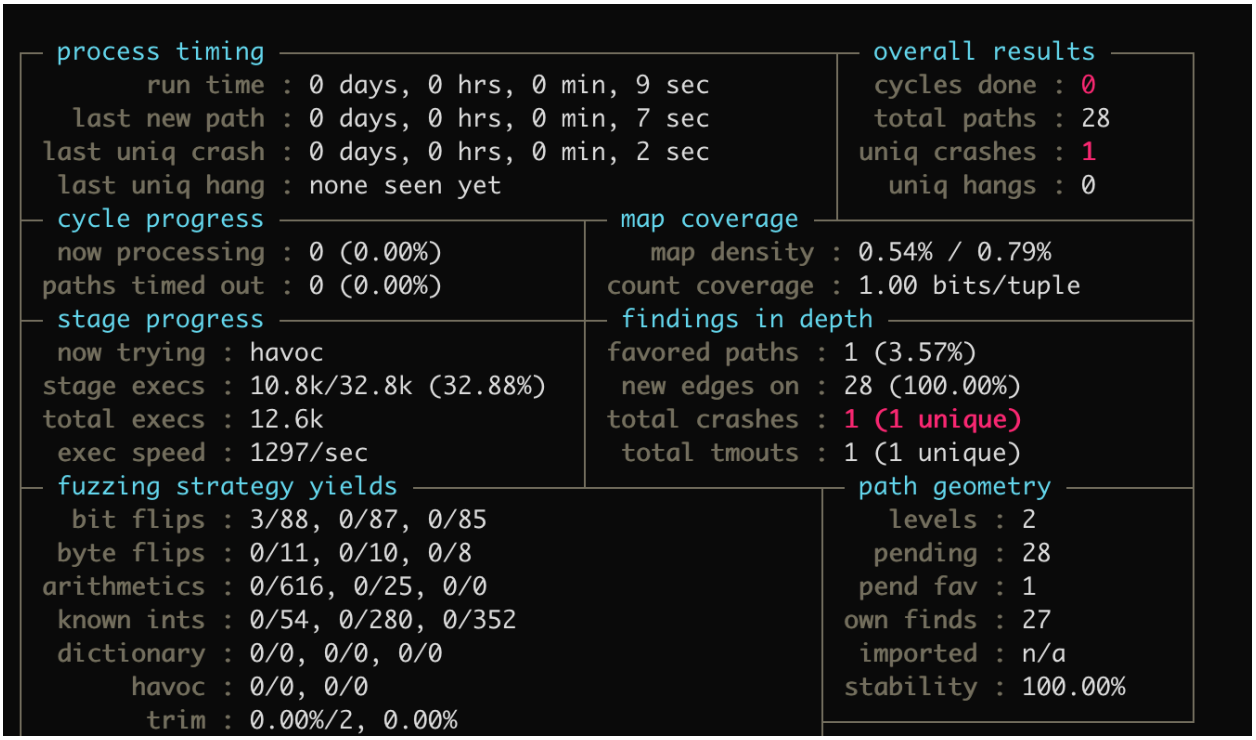
For the fuzzingwins.cpp, I chose to run bubblesort on an array of 10 unsigned integers and assert that the third element is not 2. I chose this because running bubblesort on 10 items does not take very long and is easy to understand. Symbolic execution does not do as well because there is a nested loop in the sort. Additionally, there are many different possibilities for the unsorted array that could be passed into the function. This makes it difficult to try to solve for inputs where the sorted list has a value of 2 as the third element. The symbolic executution has to try to reason backwards from <assert array[2] == 2>, causing many paths in the code and constraints to be met.

First, I tried running deepstate-angr to see how symbolic execution would handle the program. I let angr run for 40 minutes with no failures being generated. Again, my intuition was correct.

Next, I ran AFL for the first fuzzer test. For the input, I passed in 10 0’s. AFL immediately found a crash. It took 9 seconds to find input to make the test fail. The input that caused the failure was:

```
TRACE: fuzzingwins.cpp(28): Value [0]: 1684275714
TRACE: fuzzingwins.cpp(28): Value [1]: 33686018
TRACE: fuzzingwins.cpp(28): Value [2]: 33686018
TRACE: fuzzingwins.cpp(28): Value [3]: 33686304
TRACE: fuzzingwins.cpp(28): Value [4]: 2
TRACE: fuzzingwins.cpp(28): Value [5]: 1979646080
TRACE: fuzzingwins.cpp(28): Value [6]: 268435472
TRACE: fuzzingwins.cpp(28): Value [7]: 2135702
TRACE: fuzzingwins.cpp(28): Value [8]: -1768515946
TRACE: fuzzingwins.cpp(28): Value [9]: -1768515946
```

And the output was:



The second fuzzer I ran was LibFuzzer. I let it run for 4 minutes and passed in a corpus directory to save the output so that the tests could be reran. In that 4 minutes, LF found 11 failing inputs. The first input was:

```
TRACE: fuzzingwins.cpp(28): Value [0]: 1633789242
TRACE: fuzzingwins.cpp(28): Value [1]: 681468513
TRACE: fuzzingwins.cpp(28): Value [2]: 1633771777
TRACE: fuzzingwins.cpp(28): Value [3]: 31
TRACE: fuzzingwins.cpp(28): Value [4]: 1632698368
TRACE: fuzzingwins.cpp(28): Value [5]: 2
TRACE: fuzzingwins.cpp(28): Value [6]: 1364262912
TRACE: fuzzingwins.cpp(28): Value [7]: 1358954496
TRACE: fuzzingwins.cpp(28): Value [8]: 0
TRACE: fuzzingwins.cpp(28): Value [9]: 0
```

Finally, I tried running the executable with –fuzz. Before running the fuzzer, I thought that it would not be able to find an input that fails since the requirements are super specific, and I was correct. As with symexwins, the fuzzer could not find an input because it generates inputs randomly without being “smart” and there is a very small chance that it would generate such input.

The symexwins function seems more realistic because it is checking a simple math equation and the test makes sense. A bigger program could easily use this type of function (changing the numbers around if needed). As for fuzzingwins, it is not probable that anyone would be checking if just the third element of an array is 2. If they were actually testing the bubble sort, they would compare the whole output with some truth data.