A Crash Course on Testing

(pun intended)

Robbert van Renesse

Why?

- Reduces embarrassments
- It is part of the fun of developing software
  - It’s detective work, research, etc.
    - finding a bug is a new discovery
  - It’s creative
    - every software artifact needs a different set of tests
    - you get to make up those tests
  - It’s appreciated
    - people are super-happy when you find a bug
      - particularly if you can be specific about it
  - It’s a skill
    - you get better at it the more you do it

Step 1: Does your software compile *without warnings*?

- Run cc with the –Wall flag
- Any warning is something that should be investigated
  - often sign of a bug
- Don’t remove warnings simply by “casting your way out of it”
  - Unless that is exactly what is needed…
    - mostly casting void* to specific type*
    - e.g., printf(“%s\n”, (char *) arg);

Be *systematic*!!

- Start small
  - Does my sort program work for an empty array?
  - Does my sort program work for an array with just one element?
  - Does my sort program work for two elements? Check all three cases:
    - sort([a, b]), sort([b, a]), sort([a, a])
    - probably also works then for [b, b], [a, c], and any other two elements

Be *systematic*!!

- Start small
  - Does my sort program work for an empty array?
  - Does my sort program work for an array with just one element?
  - Does my sort program work for two elements? Check all three cases:
    - sort([a, b]), sort([b, a]), sort([a, a])
    - probably also works then for [b, b], [a, c], and any other two elements
- Stay small
  - Does my sort program work for three elements? Check all cases!
  - If my sort program works for 3 elements, it probably works for 3000000000
  - But it's a lot easier to consider all 12 cases when you have only 3 elements:
    - aaa, aab, ~~aac~~, aba, abb, abc, ~~aca~~, acb, ~~acc~~, baa, bab, bac, bba, ~~bbb~~, ~~bbc~~, bca, ~~bcb~~, ~~bcc~~, ~~caa~~, cab, ~~cac~~, cba, ~~cbb~~, ~~cbc~~, ~~cca~~, ~~ccb~~, ~~ccc~~

Be *systematic*!!

- Test 1

  int main(){ thread_init(); thread_exit(); printf("Test1: should not get here\n"); }
- Test 2

  void test(void *arg){}

  int main(){ thread_init(); thread_create(test, NULL); thread_exit(); }
- Test 3

  void test(void *arg){ thread_exit(); printf("Test2: should not get here\n"); }

  int main(){ thread_init(); thread_create(test, NULL); thread_exit(); }
- Test 4

  void test(void *arg){ thread_yield(); }

  int main(){ thread_init(); thread_create(test, NULL); thread_exit(); }
- …

If your code runs with 2 or 3 threads, it'll probably work with 200000 threads, but it's much hard to hit all the corner cases with 200000 threads

Be *systematic*!!

- Dining philosophers with 1, then 2, then 3 philosophers
- Diminishing returns for more philosophers
  - May not even be able to generate deadlock situations

Also systematically check *Constants*

- Consider the constants in your code, e.g.,
  - #define NLEVELS 3
  - #define BUFFER_SIZE 10
  - …
- Does your code still work if you set these to 1??
  - If not, you most likely have a bug

Stress testing

- Great for stuff that deteriorates with use
  - e.g., bridges, brakes, appliances, etc.
  - Does your code deteriorate with use?
    - maybe: memory leaks, O(n) data structures, performance bugs, …
- May find bugs in corner cases, but better to systematically search
  - Highly likely not to find corner cases

Run one test at a time

```
void test0(){
        if (something's wrong) exit(1);
}
int main(int argc, char *argv[]){
        switch (atoi(argv[1])) {
        case 0: test0(); break;
        case 1: test1(); break;
        …
        }
        return 0;
}
```

```
void (*tests[])(void) =
  { test0, test1, …};

int main(int argc, char *argv[]){
        tests[atoi(argv[1])]();
        return 0;
}
```

Check *Code Coverage*

- Is every line in my application code executed at least once?
  - For every if statement, are both cases tested?
  - For every case statement, are all cases tested (including default)?
  - For every while loop, are both cases (none, some) tested?
- Code coverage tools exist
  - E.g., gcov
  - Or use your debugger with breakpoints in code you want to test
- Or simply add some code that checks the above
  - Use #define / #ifdef if you like to not muddy up your code

Black box vs *Gray box* testing

- Do as much as possible with black box tests
  - so your test program is useful with other implementations or if you change your implementation
- But do put assert statements in the code you're testing
  - so you can check the application implementation's invariants

Memory Leaks

- What is a memory leak?
  - Memory lost in repeated operations
  - Like forgetting to release the memory used for a dequeued queue entry
  - Like forgetting to release the stack or TCB of a thread that has run
  - Program will eventually run out of memory if there is a leak
- But not:
  - Memory that is allocated just once during the execution of the program
  - Like the run queue

Finding *memory leaks*

- Have functions for allocating and releasing each type of object
  - alloc_tcb()
  - free_tcb()
  - alloc_queue()
  - free_queue()
  - …
- Keep counters for each type of object
- At end of test program, check that the counters are small
  - do not necessarily need to be 0
  - should not grow as a function of complexity of test
- Use tools if possible
  - valgrind (but has trouble with homebrew threading systems…)

Test your *test programs*!!

- Intentionally place bugs in your application
  - Does your test program find them?
- Many bugs are in test programs
  1. Can’t find a bug that is there
  2. Finds a bug that isn’t there
- In my experience, test programs are usually buggier than the code they are supposed to test

Working in a *team*

Who is responsible for what?

- Coding the main program
- Coding the test programs
- Testing the main program
- Testing the test programs

Working in a *team*

Who is responsible for what?

- Coding the main program
- Coding the test programs
- Testing the main program
- Testing the test programs

Team members are responsible together for all these

- Also do code reviews for one another

Take a *software engineering* course

CS 5154: Software Testing

- taught by Owolabi Legunsen
- may be the best investment in your coding skills