Homework 11
Debugging

Solutions

2 Finding primes

2.1 Debugging with `gdb`

You may find it helpful to consult the gdb lecture notes or this quick command reference.

First, make sure you are on the `gdb-debug-1` branch:

```
> git branch
  master
* gdb-debug-1
```

Next, build and run the supplied program:

```
> make
> ./prime
Find all prime numbers between 3 and ?
10
Segmentation fault (core dumped)
> gdb -q ./prime
Reading symbols from ./prime...done.
(gdb) run
Starting program: /media/sf_prime/prime
Find all prime numbers between 3 and ?
10

Program received signal SIGSEGV, Segmentation fault.
0x000000000040068f in check_prime (k=3) at check.c:15
15      if (is_prime[j] == 1)
```

Explain in what case(s) executing this line of code could cause a segmentation fault?

This question is asking only about this specific line of code, not all of the things that could possibly cause segmentation faults.

This is an array access, which is a fancy way of dereferencing a pointer. Since we are in C, `is_prime[j]` is the exact same as `*(is_prime + j)`.

This means that this line of code could cause a segmentation fault if the value `is_prime + j` does not point to a valid memory address. This is most likely to happen if `is_prime` is not a valid pointer or `abs(j)` is an enormous value.

Worth Noting: While a value of `j=-1` is very likely incorrect, it is unlikely to cause a segmentation fault. It will simply point to a different, nearby variable in the program's memory space. The fact that bugs like this do not cause crashes (at least not immediately) can make them hard to track down.

What gdb commands could you run next to prove your hypothesis right or wrong?

Either `print is_prime` or `ptype is_prime` to learn the size of the array, then `print j` to learn whether `j` is valid.

Asking gdb to `print is_prime[j]` will directly show that the memory pointed to is inaccessible.

Now checkout gdb-debug-2:

```
> git checkout gdb-debug-2
> make
> ./prime
Find all prime numbers between 3 and ?
10
3 is a prime
5 is a prime
7 is a prime
9 is a prime
```

Copy your debugging session and add notes explaining your thought process as you track down why this program thinks 9 is a prime.

Hint: It looks like things go well up until 9. A good place to start then may be to break in and observe how the code determines whether 9 is a prime number.

```
$ gdb -q ./prime
Reading symbols from ./prime...Reading symbols from [...] ...done.
done.

Things go well up until checking if 9 is prime, so jump there
(gdb) break check_prime if k==9
Breakpoint 1 at 0x100000e87: file check.c, line 13.
(gdb) run
Starting program: /private/tmp/debugging-basics/prime
Find all prime numbers between 3 and ?
10
3 is a prime
5 is a prime
7 is a prime

Breakpoint 1, check_prime (k=9) at check.c:13
13    for (j=2; j*j <= k; j++) { expect to run this loop for j=2 and j=3

There's not a whole lot going on in this loop. As we move through it, the
only variable that changes is j. (We can infer the values of things like
is_prime[j] based on the path the code takes, though you could certainly
print those values as well if you like)
(gdb) display j
1: j = 0
(gdb) step
14      if (is_prime[j] == 1) expect this to be true, as 2 is prime
1: j = 2
(gdb) <enter> repeats last command, in this case step
15        if (k % j == 0)  { 9 % 2 isn't 0
1: j = 2
(gdb) <enter>
19      j++; okay, onto the next iteration of the loop
1: j = 2
(gdb) <enter>
13    for (j=2; j*j <= k; j++) { starting off the loop for j=3
1: j = 3
(gdb) <enter>
23    is_prime[k] = 1; wait, we just dropped out of the loop, and j is 4
1: j = 4
(gdb)
Key thing to notice: The loop body never executed for j=3. Why? j was
incremented twice, once at the end of the body of the for loop, and once
by the declaration of the for loop. This double-increment is the bug.
```

This homework is based off of a simplified and updated version of this gdb tutorial: http://heather.cs.ucdavis.edu/~matloff/UnixAndC/CLanguage/Debug.html If you would like some more practice, or simply a slightly different explanation of the material, this may be a good source.

3 Valgrind

Notice that "3 is a prime" printed before this warning. Why was this warning not emitted when running `check_prime(3, ...)`?

Valgrind detects errors at run time. For `check_prime(3, ...)`, the for loop check `j*j <= k` (`2*2 <= 3`) fails and the body of the loop is never entered. This means `is_prime[j]` (`is_prime[2]`) never runs, which means valgrind does not detect the error.

The key takeaway here is that because of how valgrind works, it can only catch an error when it actually happens.

```
> valgrind --vgdb=yes --vgdb-error=0 ./prime
```

In another terminal, follow the directions that valgrind prints to connect gdb. In this case, valgrind has already 'run' the program for you and inserted a breakpoint at the very beginning of the program, we just need to 'continue' it. The program will run until valgrind encounters an issue, at which point valgrind will automatically break for you.

```
(gdb) continue

valgrind terminal:
==23589== Conditional jump or move depends on uninitialised value(s)
==23589==    at 0x400623: check_prime (prime.c:32)
==23589==    by 0x4006C4: main (prime.c:52)
==23589==
==23589== (action on error) vgdb me ...  # this is where valgrind waits for gdb

gdb terminal:
Program received signal SIGTRAP, Trace/breakpoint trap.
0x0000000000400623 in check_prime (k=5, is_prime=0xffefff910) at prime.c:32
32      if (is_prime[j] == 1)   # this is the problematic line of code
```

What is the value of `j` at this point?

```
(gdb) print j
j = 2
```

What is the value of `is_prime[j]` at the point?

```
(gdb) print is_prime[j]
= -16777744 ← This will be a garbage value and vary person to person
```

Use git to look at the most recent commit. What line of code was deleted that should not have been?

```
$ git log -p

[... trim some output ...]

  int main() {
+       int is_prime[MAX_PRIMES];
+       int upper_bound;
        int N;

        printf("Find all prime numbers between 3 and ?\n");
        scanf("%d", &upper_bound);

-       is_prime[2] = 1;
-
        for (N = 3; N <= upper_bound; N += 2) {
-               check_prime(N);
+               check_prime(N, is_prime);
                if (is_prime[N])
                        printf("%d is a prime\n",N);
  }
```