Device Drivers

- The interface between hardware and software in the operating system.
- Communicate with the rest of the operating system using various kernel APIs.
- Can easily cause the operating system to crash.

Diagram from [Witkowski, 2007]
Model Checking

- A method of formal verification that uses temporal logic specifications.
- Does not require a theorem prover (and does not construct a correctness proof).
- Produces a counterexample if the specification is violated.
- Used extensively on hardware, but little on software due to software’s large models.
Predicate Abstraction and Refinement

void main() {
    int a, b, c;

    if(a == b) {
        c = a;
    }

    if(a == b && c != b) {
        error();
    }
}

Original Program

void main() {
    bool b1, b2;
    b1 = random();
    b2 = random();

    if(b1) {
        b2 = false;
    }

    if(b1 && b2) {
        error();
    }
}

Boolean Program

void main() {
    bool b1, b2;
    b1 = random();
    b2 = random();

    if(b1) {
        b2 = false;
    }

    if(b1 && b2) {
        error();
    }
}

Refined Boolean Program

- A method of extracting the semantics of a program.
- Makes model checking of software more feasible.
- Involves producing a Boolean program from the original, checking it, then refining it as needed.
Model Checking Device Drivers

- Idea pioneered at Microsoft with the SLAM project.
- Stubs of driver API functions with assertions added.
- Driver methods called in arbitrary order with arbitrary data.
- Model checker checks whether assertions are violated.
- Used to find bugs in numerous Windows device drivers.
Tools for Model Checking

- SATABS: Converts C programs to Boolean programs and runs a model checker on them.
- Cadence SMV: A model checker that can be used with SATABS.
DDVerify: Model Checking Linux Device Drivers

- A program that allows Linux device drivers to be verified.
- Includes three models: one sequential and two concurrent.
- Checks properties related to locking, work queues, interrupt handling, I/O port usage, and memory usage.
- Generates a program that initializes the driver and then calls its methods in arbitrary order.
- Uses SATABS (and thus Cadence SMV) on this program to verify assertions in the model.
Updating DDVerify

- DDVerify was written in 2007, and couldn’t handle drivers from newer kernels.
- I updated it to handle current drivers.
- Mostly involved moving header files around, but also changing and adding some data structures.
- Verified that a bug still exists in the machzwd driver.
- Checked the iTCO driver, finding no bugs.
  - 67 claims related to I/O ports, array bounds, spinlocks, and context checks.
  - All but one verified within the default 20 CEGAR iterations. One took 37 iterations.
Adding Support for Network Drivers

- Three major classes of drivers in Linux: character, block, and network.
- DDVerify handled character and block.
- I added support for network drivers.
- Involved adding required header files, writing some stub methods, and writing the test harness code for calling network driver methods.
Issues with Network Drivers

- DDVerify will run on network drivers: it parses the code and generates claims ...
- ... but Cadence SMV crashes when trying to verify, likely because the model is too large for it to handle.
- Two likely causes of this:
  - The kernel’s network API is more complex than other APIs. More functions and more data structures result in a bigger model.
  - I probably left more code intact than absolutely necessary to capture the relevant semantics. More code means more to check.
Conclusions

- Model checking can be an effective method for finding bugs in device drivers.
- Creating the kernel models necessary for checking drivers is tedious work that must be done by hand.
- Keeping models up-to-date as the kernel changes may encourage Linux developers to use model checking more, but this would be best done by the kernel developers themselves.
References

Thorough static analysis of device drivers.

The slam project: debugging system software via static analysis.
In POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, pages 1–3, New York, NY, USA. ACM.

Predicate abstraction of ANSI–C programs using SAT.

SATABS: SAT-based predicate abstraction for ANSI-C.

Model checking.

Formal Verification of Linux Device Drivers.

Model checking concurrent linux device drivers.