Automatic Verification of Data Race Freedom in Linux Device Drivers
Pantazis Deligiannis · Alastair Donaldson

**Problem**

Device drivers are notoriously hard to develop and even harder to debug. This has a negative impact in hardware product releases, as time-to-market is commonly dominated by driver development, verification and validation schedules. Even after a driver has shipped, it is frequently prone to many serious errors:

- The relative error-rate in driver code is up to 10 times higher than the rest of the Linux kernel
- 19% of the bugs in Linux drivers are due to concurrency issues such as data races and deadlocks
- 85% of the system crashes in Windows XP are caused by faulty device drivers

**Challenges**

- How to reduce the generated false positives?
- How to deal with the state-space explosion problem?
- How to model interrupts and other kernel / driver specific features?
- How to scale the technique to real life drivers?

**Research Question**

Can we automatically verify the absence of data races in Linux device drivers and at the same time achieve a high degree of precision, scalability and performance?

**Proposed Approach**

1. The driver’s LLVM bitcode is translated to the Boogie intermediate verification language using SMACK, an open source LLVM-IR translator that efficiently models heap manipulating programs.

2. The concurrent Boogie program is then transformed using pair-wise sequentialisation, and is instrumented with locksets and race checking. The result is a sequential program that over-approximates the behaviours of the original concurrent program, which can lead to false positives. Invariant generation can be applied to increase precision.

3. Perform sound static lockset analysis to find all unprotected accesses to shared resources.

4. Try to further limit the amount of reported false bugs. A generated counterexample can be used to guide a systematic concurrency testing tool to explore thread interleavings that might cause the potential bug to manifest. Only bugs that are found feasible are reported to the final user.