TAS – Article Presentation
Verification of Device Drivers and Intelligent Controlles: a Case Study
David Monniaux

Thibault Krebs Jordi Bertran de Balanda

Février 2017

Overview

Device drivers and critical systems

Device driver

- Extremely hard to verify in isolation - take the **hardware** reaction into account.
- Hardware and software interface with memory mapped registers, within a simple specification.

Industry shift

- Critical system devices were *extremely* simple
- Highest security rating devices ship with limited I/O subsystem in place of an OS
- Progressive shift towards off-the-shelf interfaces like USB or Ethernet

Device driver certification is **very** desirable. To achieve it, we need to be able to prove a device driver's code is *sound*.

Challenges

Two main challenges to using consumer interfaces in critical systems:

- Many unnecessary features: hotswapping, plug-and-play, dynamic configuration, etc... Unneeded in most **static** critical applications
- Higher bandwidth, which means the use of DMA (Direct Memory Access), making the device driver behaviour essentially **asynchronous**, therefore harder to analyze

Asynchronous behaviour is BAD

Asynchronous behaviour is **especially** important in device drivers, where software/hardware interfaces *and* driver state are all, in most drivers, in *shared* memory space.

Drivers chosen

The author chose the OHCI specification for USB 1.1 devices as his testbed for device driver analysis.

Goals

1. Develop a formal definition of the informal English specification of OHCI
2. Extend the Astrée abstract interpreter for use with the specifics of
3. Model the device driver's asynchronous behaviour into something that can be analyzed by Astrée

We will see later that some concessions had to be made in order to have a working end result.

Extensions made to Astrée

Main potentially problematic issues with device driver code:

- Pointer arithmetic of variable step: modelized with $\mathit{ptr} = \mathit{base} + \mathit{offset}$ using non-relational abstract domains. Since most of the structures used will be static within the "critical system" application, this is especially important.
  - interval domain $[m, M]$ for the bounds of the pointer
  - congruential domain $a + b.\mathbb{Z}$ for pointer stride
- Extensive use of bitmaps coming from the concern of optimizing memory usage - typically in option vectors or state variables. This is done with a specific domain, with no further description from the author.

Dynamic State Partitioning

Instead of analyzing a code fragment for all possible input states, split the input states according to a predicate.
For example:

```
if (condition) {
  a;
}
else {
  b;
}
c;
```

```
if (condition) {
  a;
  c;
}
else {
  b;
  c;
}
```

This is especially useful when an array index variable potentially points to many different locations verifying local invariants.

Driver/Program Asynchronous Modeling

We consider the interactions of the controller and driver program a sequence $p_1 a^* p_2 a^* p_3 a^* \dots$, where p_i are the atomic steps of execution of the main program, and a steps are the non-deterministic actions of the controller.
We **ignore** the sequences that do not interfere: all we are interested are the $p_1 a^* p_2 a^* p_3 a^*$ where a^* precedes a p_i that **reads or writes** into shared memory.

Abstract Interpretation Results

With the extensions, the analyzer is able to detect:

- Out of bounds memory accesses
- Null pointer issues and invalid pointer dereferencing
- Invalid pointer arithmetic
- Arithmetic issues - overflow, 0-division, etc..

End result

The extended analyzer is able to prove correct (free of runtime errors) the USB 1.1 device driver used by the author, even in the presence of the “done queue”.

Limitations and Caveats

- Code alterations:
    - `goto` instructions were removed to achieve better precision.
    - Padding in some data structures caused Astrée to allocate sufficient unneeded variables as to impact performance.
- Loss of precision in data bit length because of the "done queue" confusing 16 bit FD for 32 bit FD
- Bitmap troubles: OHCI specification sometimes orders information storage in the 2 lowest-order bits of pointers that are supposed to be aligned in 4-byte stripes.
- The deeply nested *Endpoint Descriptor* data model, when active in the driver, caused the analyzer to run over 100 times slower (6 minutes with the "done queue" disabled against 10 hours with it enabled)

Related Work

- “Bug-finding” techniques (, ad-hoc techniques for Linux/BSD kernels)
    - Microsoft SLAM group - model checking of boolean abstractions: begin with a boolean control-flow abstraction of the program, then refine it.
    - Ad-hoc techniques with Linux and OpenBSD kernels - statistical analysis, attempting detection of incorrect or inconsistent usage, etc..
- “Doomed” approaches for device drivers advocating a language change:
    - “Safe” subset of C (MISRA-C)
    - Type-safe languages (CCured)

Conclusion

End result

A verification tool for proving the absence of runtime errors in a **critical** device driver.

- Some compromises are acceptable, like ignoring unneeded features (hotswapping, plug-and-play, etc..)
- Some compromises are natural, like using mostly static allocation in a system where the components (hardware and software) are known.
- Some future work on the efficiency of this approach is desirable
  - through limiting the use of Dynamic State Partitioning
  - through a better pointer abstract domain, enabling better handling of complex dynamic structures

No generic solution

No one-size-fits-all solution! This specific driver's proof required iterative changes to existing abstract interpreters before being possible in a single pass.