Automatic Verification of Data Race Freedom in Device Drivers* (Extended Abstract)

Pantazis Deligiannis and Alastair F. Donaldson

Imperial College London {p.deligiannis, alastair.donaldson}@imperial.ac.uk

Abstract

Device drivers are notoriously hard to develop and even harder to debug. They are typically prone to many serious issues such as data races. In this paper, we present *static pair-wise lock set analysis*, a novel sound verification technique for proving data race freedom in device drivers. Our approach not only avoids reasoning about thread interleavings, but also allows the reuse of existing successful sequential verification techniques.

1998 ACM Subject Classification D.2.4 Software/Program Verification

Keywords and phrases Device Drivers, Verification, Concurrency, Data Races

Digital Object Identifier 10.4230/OASIcs.ICCSW.2014.36

1 Introduction

Device drivers are complex pieces of system-level software responsible for the interaction between an operating system and any hardware devices that are attached to a computer [10]. Drivers are notoriously hard to develop and even harder to debug. Even after a device driver has shipped, it is typically prone to many serious errors [6, 31]. Regarding *concurrency bugs*, a recent study [28] found that they account for 19% of the total bugs in Linux drivers, showcasing their significance. The majority of these concurrency bugs were found to be *data races* or *deadlocks* in various configuration functions and hot-plugging handlers.

The main focus of this paper is on data races, which can lead to nondeterministically occurring bugs that can be challenging to reproduce, isolate and debug. Well-known Linux kernel analysers, such as sparse [8], coccinelle [22] and lockdep [9], have successfully found deadlocks in kernel code, but are typically unable to detect data races. Techniques such as [1, 7, 25, 12, 14, 16, 27, 17] have been used to analyse Linux and Windows device drivers, but primarily focus on sequential program properties. Furthermore, most previous techniques that attempt to reason about thread interleavings face significant scalability issues, because of the exponentially large state-space of realistic concurrent programs [19].

This work-in-progress paper presents *static pair-wise lock set analysis*, a novel technique for automatically verifying data race freedom in device drivers. The key idea behind our approach is that a driver can be proven free from data races by (i) deriving a sound *sequential* model that *over-approximates* the originally concurrent driver, (ii) instrumenting it for lock set analysis and race checking, and (iii) asserting that all accesses to the same shared resource are protected by at least one common lock. The immediate benefit is that our approach not only avoids reasoning about thread interleavings, and thus has the potential to scale well, but also allows the reuse of existing successful sequential verification techniques.

* This work is part of the research project "Automatic Synthesis of High-Assurance Device Drivers" and is generously funded by a gift from Intel Corporation.

Imperial College Computing Student Workshop (ICCSW'14).
Editors: Rumyana Neykova and Nicholas Ng; pp. 36–39
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Static Pair-Wise Lock Set Analysis

Our technique involves reasoning about the *lock sets* of a driver. Lock set analysis has its roots in Eraser [29], a dynamic data race detector that tracks the set of locks that consistently protect a memory location during program execution. If that set ever becomes empty, the tool reports a potential data race. This is because an inconsistent lock set suggests that a memory location can be accessed simultaneously by two or more threads.

Eraser avoids reasoning about arbitrary thread interleavings, and thus can scale well in realistic concurrent programs, but suffers from imprecision (i.e. can report false bugs), because a violation of the locking discipline does not always correspond to a real data race [29, 23, 20, 11, 13]. Furthermore, as a dynamic analyser, Eraser's bug finding ability is limited to the execution paths that the tool explores. To counter the second limitation, we apply the idea of Eraser's lock set analysis in a static verification context.

Static pair-wise lock set analysis begins by performing *two-thread reduction*, an abstraction that removes all but two arbitrary threads, each running an entry point of the originally concurrent driver. This reduction was inspired by a reduction to two threads employed in the GPUVerify tool for verification of GPU kernels [5, 2]. The technique then proceeds with *pair-wise sequentialisation*, which combines the two arbitrary threads in a single sequential pair. The pair is finally instrumented for lock set analysis with assertions that check if each memory location is consistently protected by at least one common between the two threads lock. This process repeats until all pairs of entry points have been sequentialised. To achieve soundness, each time an entry point performs a read access to a shared resource, we return a nondeterministic value. This over-approximates any effects from all the unmodeled threads on the driver shared state.

We have prototyped this technique in Whoop, a practical tool for automatic concurrency verification of Linux drivers written in C [15]. Whoop initially compiles the driver source code, together with an environmental model, to LLVM-IR using Clang/LLVM [18]. The program is then compiled to the Boogie [3] verification language using SMACK [26], an LLMV to Boogie translator which can efficiently model heap manipulating programs. Next, the Boogie program is sequentialised and instrumented, using static pair-wise lock set analysis. The abstract program is finally send to the Boogie verifier, which generates verification conditions [4] and discharges them to a theorem prover. Successful verification implies that the original driver is free of data races, while an error denotes a *potential* data race.

The main limitation of our approach is that it can potentially report many false positives, as we over-approximate the shared state. To tackle this problem, we plan to investigate (i) invariant generation for taming our coarse abstraction and (ii) counterexample feasibility checking to evaluate if a reported bug is real or spurious.

3 Related Work

Notable previous works on static analysis for race detection include the static analysers Warlock [30] and LockLint [21], which, however, heavily rely on user annotations. Whoop does not require any source code modifications, and thus can be applied with zero effort.

Most related to our work are the static lock set analysers RELAY [32] and Locksmith [24]. Both tools, though, have significant limitations. RELAY uses unsound post-analysis filters to limit the false positives, but these can filter out true races. Although Locksmith successfully detected data races in 7 medium-sized Linux device drivers, it reported a significant number of false positives. The authors also reported that Locksmith was unable to run on several large programs, showcasing its limited scalability.

References

1 Thomas Ball, Ella Bounimova, Byron Cook, Vladimir Levin, Jakob Lichtenberg, Con McGarvey, Bohus Ondrusek, Sriram K Rajamani, and Abdullah Ustuner. Thorough static analysis of device drivers. *ACM SIGOPS Operating Systems Review*, 40(4):73–85, 2006.

2 Ethel Bardsley, Adam Betts, Nathan Chong, Peter Collingbourne, Pantazis Deligiannis, Alastair F Donaldson, Jeroen Ketema, Daniel Liew, and Shaz Qadeer. Engineering a static verification tool for GPU kernels. In *Proceedings of the 26th International Conference on Computer Aided Verification*, 2014.

3 Mike Barnett, Bor-Yuh Evan Chang, Robert DeLine, Bart Jacobs, and K Rustan M Leino. Boogie: A modular reusable verifier for object-oriented programs. In *Formal methods for Components and Objects*, pages 364–387. Springer, 2006.

4 Mike Barnett and K Rustan M Leino. Weakest-precondition of unstructured programs. *ACM SIGSOFT Software Engineering Notes*, 31(1):82–87, 2005.

5 Adam Betts, Nathan Chong, Alastair Donaldson, Shaz Qadeer, and Paul Thomson. GPUVerify: a verifier for GPU kernels. In *Proceedings of the ACM International Conference on Object Oriented Programming Systems Languages and Applications*, pages 113–132. ACM, 2012.

6 Andy Chou, Junfeng Yang, Benjamin Chelf, Seth Hallem, and Dawson Engler. An empirical study of operating systems errors. In *Proceedings of the eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 73–88. ACM, 2001.

7 Edmund Clarke, Daniel Kroening, Natasha Sharygina, and Karen Yorav. Predicate abstraction of ANSI-C programs using SAT. *Formal Methods in System Design*, 25(2-3):105–127, 2004.

8 Jonathan Corbet. Finding kernel problems automatically. `https://lwn.net/Articles/87538/`, 2004.

9 Jonathan Corbet. The kernel lock validator. `https://lwn.net/Articles/185666/`, 2006.

10 Jonathan Corbet, Alessandro Rubini, and Greg Kroah-Hartman. *Linux device drivers (Third Edition)*. O'Reilly, 2005.

11 Tayfun Elmas, Shaz Qadeer, and Serdar Tasiran. Goldilocks: A race and transaction-aware Java runtime. In *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '07, pages 245–255. ACM, 2007.

12 Dawson Engler, Benjamin Chelf, Andy Chou, and Seth Hallem. Checking system rules using system-specific, programmer-written compiler extensions. In *Proceedings of the 4th USENIX Symposium on Operating System Design and Implementation*. USENIX, 2000.

13 Cormac Flanagan and Stephen N Freund. FastTrack: Efficient and precise dynamic race detection. In *Proceedings of the 2009 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '09, pages 121–133. ACM, 2009.

14 Thomas A Henzinger, George C Necula, Ranjit Jhala, Gregoire Sutre, Rupak Majumdar, and Westley Weimer. Temporal-safety proofs for systems code. In *Computer Aided Verification*, pages 526–538. Springer, 2002.

15 Brian W Kernighan, Dennis M Ritchie, and Per Ejeklint. *The C programming language*, volume 2. prentice-Hall Englewood Cliffs, 1988.

16 Volodymyr Kuznetsov, Vitaly Chipounov, and George Candea. Testing closed-source binary device drivers with DDT. In *Proceedings of the 2010 USENIX Annual Technical Conference*. USENIX, 2010.

17 Akash Lal, Shaz Qadeer, and Shuvendu K. Lahiri. A solver for reachability modulo theories. In *Proceedings of the 24th International Conference on Computer Aided Verification*, CAV'12, pages 427–443. Springer, 2012.

18 Chris Lattner and Vikram Adve. LLVM: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization*, pages 75–86. IEEE, 2004.

19 Madanlal Musuvathi, Shaz Qadeer, Thomas Ball, Gerard Basler, Piramanayagam Arumuga Nainar, and Iulian Neamtiu. Finding and reproducing heisenbugs in concurrent programs. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation*, pages 267–280. USENIX, 2008.

20 Robert O'Callahan and Jong-Deok Choi. Hybrid dynamic data race detection. In *Proceedings of the 9th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '03, pages 167–178. ACM, 2003.

21 Oracle Corporation. Analyzing program performance with Sun WorkShop, Chapter 5: Lock analysis tool. `http://docs.oracle.com/cd/E19059-01/wrkshp50/805-4947/6j4m8jrnd/index.html`, 2010.

22 Yoann Padioleau, Julia Lawall, René Rydhof Hansen, and Gilles Muller. Documenting and automating collateral evolutions in Linux device drivers. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems*, Eurosys '08, pages 247–260. ACM, 2008.

23 Eli Pozniansky and Assaf Schuster. Efficient on-the-fly data race detection in multithreaded C++ programs. In *Proceedings of the 9th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '03, pages 179–190. ACM, 2003.

24 Polyvios Pratikakis, Jeffrey S. Foster, and Michael Hicks. LOCKSMITH: Context-sensitive correlation analysis for race detection. In *Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '06, pages 320–331. ACM, 2006.

25 Shaz Qadeer and Dinghao Wu. KISS: Keep it simple and sequential. In *Proceedings of the ACM SIGPLAN 2004 Conference on Programming Language Design and Implementation*, PLDI '04, pages 14–24. ACM, 2004.

26 Zvonimir Rakamaric and Alan J Hu. Automatic inference of frame axioms using static analysis. In *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*, pages 89–98. IEEE, 2008.

27 Matthew J Renzelmann, Asim Kadav, and Michael M Swift. SymDrive: Testing drivers without devices. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation*. USENIX, 2012.

28 Leonid Ryzhyk, Peter Chubb, Ihor Kuz, and Gernot Heiser. Dingo: Taming device drivers. In *Proceedings of the 4th ACM European conference on Computer systems*, pages 275–288. ACM, 2009.

29 Stefan Savage, Michael Burrows, Greg Nelson, Patrick Sobalvarro, and Thomas Anderson. Eraser: A dynamic data race detector for multithreaded programs. *ACM Transactions on Computer Systems*, 15(4):391–411, 1997.

30 Nicholas Sterling. WARLOCK - A static data race analysis tool. In *Proceedings of the 1993 Winter USENIX Conference*, pages 97–106. USENIX, 1993.

31 Michael M. Swift, Brian N. Bershad, and Henry M. Levy. Improving the reliability of commodity operating systems. In *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, pages 207–222. ACM, 2003.

32 Jan Wen Voung, Ranjit Jhala, and Sorin Lerner. RELAY: Static race detection on millions of lines of code. In *Proceedings of the the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, pages 205–214. ACM, 2007.