Chair of Decentralized Systems Engineering
Department of Informatics

Severity-analysis of data races in the Linux kernel

General information	**Advisor**	Dr. Marco Elver (Google) & Prof. Pramod Bhatotia (TUM)
	Email	elver@google.com, pramod.bhatotia@in.tum.de
	Date	06.12.2021
Type		Master / Bachelor / Guided Research
Description		Shared-memory multiprocessors are ubiquitous today. Yet, writing efficient parallel programs to exploit modern CPUs remains challenging. Over the past decades, a plethora of work in programming languages, compilers, and computer architecture aim to provide efficient abstractions to best utilize our parallel hardware. One of the most important abstractions is the memory consistency model, which formally specifies the behavior of the memory system, and is used by programmers to reason about concurrent programs. For instance, recent versions of C/C++ have received a formal memory model since C11/C++11. Crucially, such language-level memory models guarantee sequential consistency for programs without data races. The presence of data races, however, introduces undefined behaviour: data races occur when there are concurrent conflicting accesses from multiple threads, at least one of which is a plain (unmarked, non-atomic) access; accesses conflict if they all access the same memory location and at least one performs a write.

The main reason data races are undefined behaviour is so that compilers can still apply optimizations to code that would not change its behaviour in non-concurrent execution. Modern compilers, however, can apply a plethora of optimizations that would break concurrent code in the presence of data races.

For various reasons, certain programs contain deliberate data races, often called "benign data races". Programmers must make several assumptions in such cases, and reasoning properly about their code also becomes significantly harder. For one, rather than undefined behaviour, it must be assumed that the behaviour is implementation defined. Second, the code must be tolerant to all possible values that data-racy reads could return. In general, this approach should not be advocated, but may be required for several reasons: one of the most frequently stated reasons is performance, but usually due to a lack of suitable abstractions and no reasonable way to introduce them. The Linux kernel is one such project where this kind of reasoning is pervasive.

The Linux kernel also has its own memory model, the Linux-kernel Memory Consistency Model (LKMM) (also see this paper), which defines data races similarly to the C11/C++11 memory models. In the strictest sense, the LKMM also makes no prediction about data races, i.e. they are undefined.

Yet, data races are still pervasive throughout the Linux kernel. This was first demonstrated by the introduction of the Kernel Concurrency Sanitizer (KCSAN), a data race detector for the Linux kernel. Development of KCSAN also highlighted that data races in the Linux kernel are not going anywhere, by means of a new marking "data_race(<expr>)", to indicate the data race is intentional.

Marking of all racy accesses with appropriate primitives—such as READ_ONCE(), WRITE_ONCE(), or others, but also data_race()—is an ongoing process. For the time being, however, KCSAN still finds hundreds of data races in the Linux kernel.

To be more precise, often the existence of a data race is merely a symptom of a bigger issue. The existence of a data race may point out the following concurrency bugs:

A. Miscompilation may cause bugs, however, failure with current compilers is unlikely—these are usually called "benign". Merely marking the accesses appropriately is sufficient. Finding a failure requires a miscompilation. Requires no fundamental changes in program logic to fix.

B. Race-condition bugs where the bug manifests as a data race. Simply marking the accesses does not fix the problem. The fix requires more invasive changes to program logic (for example adding required locking).

The Linux kernel has too much of type (A), which means bugs of type (B) are drowned out by (A). By default, kernel developers consider bugs of type (A) very low priority, which means the status quo only changes very slowly.

Which data races should developers focus on? One outcome of this project is a tool to help developers prioritize data races as reported by KCSAN, or even those already marked data_race(), if the tool determines a data race can lead to a misbehaving kernel. One data race may be more severe than another, and appropriate severity signals need to be identified (for example a kernel crash).

One notable related work is SyzScope, which analyses severity of certain bug reports of Linux kernels fuzzed by Syzkaller. Unfortunately, they acknowledge that they "[...] exclude the ones detected by KCSAN [...] because they [...] do not have any valid reproducers".

As far as we are aware, no tools exist to analyze the severity of data races, which in part is due to most development communities acknowledging that the mere existence of a data race is a bug. Certain projects, however, have requirements that mean data races can be

tolerated. This philosophy is a double-edged sword, as developers are too easily blinded by how a simple data race points out a much bigger issue. Having a tool to demonstrate the severity of data races would help developers focus on high-impact issues, rather than ignoring all of them.

Possible approaches (from easiest to hardest):

1. Simulate miscompilations dynamically: inserting instrumentation that changes the values read or written on an observed data race (with help from KCSAN). Subsequently, being able to attribute kernel errors to that data race to determine severity.
2. Simulate miscompilations symbolically: given a concrete system state on a data race, turn a subset of that state most likely affected by the data race into symbolic values, and symbolically execute. Error conditions need to be identified, so that a solver can identify path-conditions leading to such errors.
3. Otherwise statically analyze the severity of a data race from existing KCSAN reports only.

Keywords

concurrency, memory consistency models, formal methods, compilers, operating systems, open source, Linux, software engineering

Goals

Concrete outcomes

1. Analysis of potential solutions, together with survey of relevant literature.
2. Definition of failure states following data races, used to classify data races "high-severity" (or additional levels of severity).
3. Design & implementation of a tool that can automatically classify data races.
4. Evaluation.

Bonus points

5. Reporting "enhanced" (vs. what KCSAN provides) data race reports to upstream developers.
6. Preparation (with intent to publish) of a paper resulting from this work.

Prerequisites

Compulsory

- Knowledge of C **or** C++.
- Familiar with basic concurrency concepts in C11 **or** C++11.
- Taken a course on formal methods or programming language semantics.

Preferred

- Knowledge of compiler-based dynamic analysis tools.
- Knowledge of static analysis techniques.
- Linux-kernel development.

References

1. https://lwn.net/Articles/816850/
2. https://www.kernel.org/doc/html/latest/dev-tools/kcsan.html
3. https://www.usenix.org/system/files/sec22summer_zou.pdf

Application process

Please send an email to the advisor including the following:

- Email subject: "Thesis application (DSE)"
- CV
- A copy of your transcript(s)
- A **motivation statement**, please include samples of your work that you are proud of (e.g., major projects, open-source contributions, Github page, etc.) and/or writing samples (e.g., your technical blog, project reports, etc.)