J Autom Reasoning (2009) 42:123–124
DOI 10.1007/s10817-009-9126-9

Operating System Verification

Gerwin Klein · Ralf Huuck · Bastian Schlich

Received: 1 March 2009 / Accepted: 3 March 2009 / Published online: 17 March 2009

When one is interested in assuring the safety, security, or functional correctness of a computing system, the formal verification of its operating system (OS) is one of the obvious places to start. The operating system has privileged access to hardware and is therefore able to undermine any assurance that might have been derived independently for other parts of the system.

This was recognised early, and a number of projects set out to formally verify the functional correctness of operating systems in the late 1970s and early 1980s. These pioneering efforts included UCLA Secure Unix, the PSOS project, and later Bevier's small KIT. It turned out that operating system verification is a hard nut to crack and none of the initial efforts ended up with a formally verified, realistic operating system or operating system kernel. OS verification is hard because the flaws one is interested in uncovering often occur in the implementation layer. This is because operating systems are commonly implemented in low-level languages like C that are hard to reason about, and because convenient abstractions such as virtual memory, message passing, and memory allocation are services that are implemented by the OS and cannot be assumed.

In recent years, there is a renewed interest in the formal analysis and verification of operating systems, both in the OS research community and in the formal verification area. Formal verification techniques and proof assistants have advanced dramatically in the past 30 years, as has our understanding of language semantics.

G. Klein (✉) · R. Huuck
NICTA and University of New South Wales,
Locked Bag 6016, Sydney, NSW 1466, Australia
e-mail: gerwin.klein@nicta.com.au

R. Huuck
e-mail: ralf.huuck@nicta.com.au

B. Schlich
Embedded Software Laboratory, RWTH Aachen University,
Ahornstr. 55, 52074 Aachen, Germany
e-mail: schlich@embedded.rwth-aachen.de

This special issue collects current advances in theorem proving, model checking, and static analysis from the leading research groups working on OS verification.

The first four papers are dedicated to programming language and semantics issues with a special focus on low-level systems. Tuch presents a foundational formal treatment of the C programming language. His work is one of the corner stones in the verification of the seL4 microkernel. Tews et al concentrate on the problem of virtual memory and device access and how they interact with the semantics of C/C++. Gallardo et al work one layer above and apply model checking to reason about memory allocation in operating systems using abstraction and common code patterns. Tlili and Debbabi propose an interprocedural static analysis to assure memory and type safety in low-level systems code.

The next two papers are concerned with specific aspects of OS verification. Feng et al present a Hoare-logic framework for hardware interrupts and preemption, an essential and specific component of the correctness of operating systems and device drivers. Daum et al investigate the issue of the concurrency abstraction their OS provides to the user level. They prove fairness and correctness of a microkernel scheduler.

The final paper by Alkassar et al gives an in-depth overview of the different, interleaved layers of semantics of the Verisoft project. They are used to formally verify a complete stack of systems software covering basic versions of verified hardware, a verified compiler and a verified operating system.