Reprinted from *Operating Systems Review* 45(3):1–4, December 2011. http://dx.doi.org/10.1145/2094091.2094093

Summary of PLOS 2011: The Sixth Workshop on Programming Languages and Operating Systems *

Eric Eide
University of Utah
eeide@cs.utah.edu

Gilles Muller
INRIA/LIP6-Regal
Gilles.Muller@lip6.fr

Wolfgang Schröder-Preikschat
University of Erlangen
wosch@informatik.uni-erlangen.de

Olaf Spinczyk
TU Dortmund
olaf.spinczyk@tu-dortmund.de

Abstract

This report summarizes the Sixth Workshop on Programming Languages and Operating Systems (PLOS 2011), which was held in conjunction with the SOSP 2011 conference. It presents the motivation for the PLOS workshop series and describes the contributions of the PLOS 2011 event.

1. Introduction

PLOS 2011, the Sixth Workshop on Programming Languages and Operating Systems, was held on October 23, 2011 in Cascais, Portugal, immediately prior to the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP). This was the second time that PLOS was part of the official workshop program for the biennial SOSP conference, and it was the *third* time that PLOS was colocated with SOSP. PLOS has been colocated with SOSP since 2007—even before SOSP had an official workshop program!—and it continues to be an exciting workshop because of the people who participate.

Approximately thirty-nine people participated in PLOS 2011. They heard a keynote address by Gernot Heiser, participated in eight paper presentations and discussions, saw four live system demonstrations, and at the end of the day, collaborated in three focused working groups.

2. Motivation

The goal of the PLOS workshop series is to bring together researchers and developers from the programming languages (PL) and the operating systems (OS) domains to discuss recent work at the intersection of these fields. PLOS is a platform for discussing visions, challenges, experiences, problems, and solutions arising from the application of advanced programming and software engineering concepts to operating systems construction, and vice versa. The long-standing connection between OS development and programming languages is what makes this workshop relevant to SOSP.

Historically, OS development and programming language development went hand-in-hand. Cross-fertilization was the norm. Challenges in one area were often approached using ideas or techniques developed in the other, and advances in one enabled new capabilities in both. Today, although C is the workhorse language of the systems community, novel approaches to OS construction based on new programming language ideas continue to be an active and important area of research. The systems field continues to provide a wealth of challenge problems and new results that should spark advances in programming languages, software designs, and idioms.

The connection between OS development and programming languages is both significant and current. This is demonstrated by operating systems such as seL4 [9], Singularity [5], Verve [17], and CiAO [11]; embedded OS frameworks such as TinyOS [10] and Neutron [3]; OS extension frameworks such as SafeDrive [18]; and static analyses that find hundreds of operating-system defects [14]. The PLOS workshop series is intended to be a venue for new and emerging research that follows in the footsteps of these examples—new ideas that explore the synthesis of PL and OS concepts.

3. Preparation

Eric Eide, Gilles Muller, Wolfgang Schröder-Preikschat, and Olaf Spinczyk proposed to hold the sixth PLOS workshop in conjunction with SOSP 2011. The proposal was accepted, and the organizers published a call for workshop papers in January 2011. The suggested topics for submissions to PLOS 2011 included:

- critical evaluations of new programming language ideas in support of OS construction;
- domain-specific languages for operating systems;
- type-safe languages for operating systems;
- object-oriented and component-based operating systems;
- language-based approaches to crosscutting system concerns, such as security and run-time performance;
- language support for system verification;
- language support for OS testing and debugging;
- static/dynamic configuration of operating systems;
- static/dynamic specialization within operating systems; and
- the use of OS abstractions and techniques in language runtimes.

These suggested topics were unchanged from the PLOS 2009 and 2007 calls for papers.

Wolfgang Schröder-Preikschat was the PLOS 2011 program committee chair. The other members of the committee were:

- Yolande Berbers, KU Leuven
- Eric Eide, University of Utah
- Michael Franz, UC Irvine
- Robert Grimm, New York University
- Thomas Gross, ETH Zürich
- Tim Harris, Microsoft Research Cambridge
- Julia Lawall, University of Copenhagen
- Gilles Muller, INRIA/LIP6-Regal
- Olaf Spinczyk, Technische Universität Dortmund

The program committee received eighteen short papers for consideration, and each was assigned to four members of the committee for review. Each paper from a committee member received an additional fifth review as well. One paper was submitted from the program chair's research group. For that paper, another member of the committee selected five reviewers, and the reviewers' identities were not revealed to the program chair. In total, 75 reviews were written, many with long and detailed comments.

Once all the reviews were in, the program committee met by teleconference to decide on the workshop program. Papers were evaluated based on technical quality, originality, relevance, and presentation. In the end, the committee selected eight papers for presentation and discussion at the workshop.

* http://www.plosworkshop.org/2011/

- **Welcome and Keynote**
  - Keynote Address: *The Role of Language Technology in Trustworthy Operating Systems*
    Gernot Heiser (University of New South Wales and NICTA)
- **Session 1a: Static Analyses**
  - *Finding Resource-Release Omission Faults in Linux*
    Suman Saha (LIP6-Regal), Julia Lawall (DIKU, University of Copenhagen), and Gilles Muller (INRIA/LIP6-Regal)
  - *Configuration Coverage in the Analysis of Large-Scale System Software*
    Reinhard Tartler, Daniel Lohmann, Christian Dietrich, Christoph Egger, and Julio Sincero (Friedrich-Alexander University)
- **Session 1b: Security**
  - *Rounding Pointers — Type Safe Capabilities with C++ Meta Programming*
    Alexander Warg and Adam Lackorzynski (Technische Universität Dresden)
  - *Preliminary Design of the SAFE Platform*
    André DeHon, Ben Karel (University of Pennsylvania), Thomas F. Knight, Jr. (BAE Systems), Gregory Malecha (Harvard University), Benoît Montagu (University of Pennsylvania), Robin Morisset (École Normale Supérieure Paris), Greg Morrisett (Harvard University), Benjamin C. Pierce (University of Pennsylvania), Randy Pollack (Harvard University), Sumit Ray (BAE Systems), Olin Shivers (Northeastern University), Jonathan M. Smith (University of Pennsylvania), and Gregory Sullivan (BAE Systems)
- **Session 2a: Dynamic Safety and Performance**
  - *Dynamic Deadlock Avoidance in Systems Code Using Statically Inferred Effects*
    Prodromos Gerakios, Nikolaos Papaspyrou (National Technical University of Athens), Konstantinos Sagonas (National Technical University of Athens and Uppsala University), and Panagiotis Vekris (National Technical University of Athens)
  - *Using Declarative Invariants for Protecting File-System Integrity*
    Jack Sun, Daniel Fryer, Ashvin Goel, and Angela Demke Brown (University of Toronto)
  - *Assessing the Scalability of Garbage Collectors on Many Cores*
    Lokesh Gidra, Gaël Thomas, Julien Sopena, and Marc Shapiro (Regal-LIP6/INRIA)
- **Session 2b: Reversible Debugging**
  - *URDB: A Universal Reversible Debugger Based on Decomposing Debugging Histories*
    Ana-Maria Visan, Kapil Arya, Gene Cooperman, and Tyler Denniston (Northeastern University)
- **Session 3: Demonstrations**
- **Session 4: Working Groups and Wrap Up**

Figure 1. Workshop Program

4. Program

The PLOS 2011 program, summarized in Figure 1, kicked off with an invited keynote by Gernot Heiser. Following the keynote, the first paper session focused on static program analyses (two papers) and techniques for implementing secure systems (two papers). In the second paper session, the workshop participants discussed dynamic safety and performance (three papers) and reversible debugging (one paper). Following the paper presentations, in the third session, several authors demonstrated the systems that they had just described. This allowed workshop participants to interact closely and ask detailed questions. Finally, during the last workshop session, participants organized themselves into working groups to explore three topics that had been raised earlier in the day. These topics were (1) "systems-extensible" programming languages, (2) filesystem invariants, and (3) hardware-software interfaces and verification.

4.1 Keynote

Gernot Heiser from the University of New South Wales and NICTA started the workshop with a keynote that examined the role of programming-language technology in the implementation of trustworthy systems software. More specifically, he challenged the trend toward using type-safe and managed languages for building dependable systems.

Gernot decried the growing tendency to conflate type safety with stronger notions such as trustworthiness and security—i.e., to think that type safety implies other kinds of safety, which is most definitely not true! Instead, Gernot said that "trustworthiness is best achieved through functional correctness." For proving functional correctness, he noted that managed languages can sometimes get in the way because of the complexity of their runtimes. This complexity can also be a barrier to achieving high performance. To emphasize this point, Gernot compared the published IPC performance of the L4 and Singularity kernels and noted that L4's IPC was significantly faster—while also having a smaller trusted computing base (TCB) and having proven functional correctness. Finally, Gernot noted the practical need to incorporate (unsafe) legacy code in many systems. Insisting that everything be written in a type-safe language gets in the way of building useful things.

In contrast to the trend toward using type-safe languages for all system components, Gernot championed an approach that uses different languages for different parts of a system. He suggested that low-level languages—C and assembler—be used for implementing trustworthy bottom layers. Used correctly, such implementations can be amenable to reasoning and proofs of correctness. This is the approach taken by seL4 [9], which has a low-level implementation with associated proofs for functional correctness and integrity. Using the multi-language approach, the seL4 team is continuing to prove more properties about seL4, such as confidentiality. Gernot noted that this approach yields a trustworthy base for higher layers of a software stack, such as managed-language runtimes and applications written in managed code.

To conclude, Gernot said that "we should stop kidding ourselves" about the contribution of type-safe languages in the construction of dependable systems.

4.2 Sessions 1a and 1b: Static Analyses and Security

Suman Saha from LIP6-Regal started the first paper session with a presentation about resource-release omission faults in Linux. Such faults occur when an acquired resource is not properly freed along some execution path. Suman and his coauthors implemented a static analysis to detect such errors in the Linux code base. It works by (1) identifying the set of resource-releasing operations in a function's error-handling code, (2) comparing the error-handling blocks with each other to find missing operations, and (3) applying heuristics to decide if omitted operations correspond to resource-release faults. This approach accounts for the fact that in Linux, the correct choice of a resource-releasing operation is sometimes context sensitive. Using their analyzer, Suman and his colleagues found more than 100 faults in Linux 2.6.34 drivers.

In the second talk, Reinhard Tartler discussed "configuration coverage." Systems written in C are statically configured using a preprocessor (CPP), and for large systems, configuration can be both complex and problematic. For example, the Linux kernel has more than 11,000 configurable features and the standard "allyesconfig" kernel configuration includes only 78% of all the selectable code blocks in Linux 2.6.35. To make the remaining code blocks more readily available to analyzers and testers, Reinhard and his colleagues extended their UNDERTAKER toolchain to find a set of kernel configurations that maximize the number of blocks included in at least one configuration. It does this by using a SAT solver to find values for CPP symbols that control code inclusion and exclusion. Using this approach, they were able to include 94% of all selectable code blocks in their configuration set.

The second half of the session focused on security. Alexander Warg started this half by describing techniques for representing kernel-protected object capabilities as C++ "smart pointers." Smart pointers allow a C++ programmer to invoke operations on kernel objects through the usual method-call syntax. In Warg's system, the internal representation of a smart pointer to a kernel object is just an integer—not a memory address. This is an efficient representation of capabilities, but it is tricky to implement because of the assumptions that C++ compilers make about pointers and objects' memory layouts. Warg presented the details of making the smart-pointer abstraction work with all the machinery of C++. Warg's presentation was a motivator for the working group on systems-extensible programming languages, which was held in the afternoon.

The last talk of the session was given by Benoît Montagu, who described SAFE: a security-focused, clean-slate redesign of the entire computer stack, from hardware to applications. SAFE is motivated by two modern developments in computing. First, formal methods have matured and are now capable of proving properties about real systems, such as complete instruction-set architectures (ISAs). Second, hardware resources are so abundant that it makes sense to (1) reconsider complicated features that make it difficult to reason about complete systems, such as virtual memory, and (2) devote hardware to ensuring the correct operation of the complete system, including adherence to security policies. The SAFE hardware, for example, includes a tag-management unit that can enforce information-flow rules on every instruction. The SAFE project started only recently. Most of the effort so far has focused on the design of Breeze, a new programming language for SAFE, and the design and semantics of the new hardware ISA.

4.3 Sessions 2a and 2b: Safety, Performance, and Debugging

In the afternoon paper session, Konstantinos Sagonas described a new tool for implementing dynamic deadlock avoidance in systems code. The first part of the tool is a static analyzer for C programs that use pthreads: for each lock operation and function call, the analyzer computes a "continuation effect," which is a sequence of future lock and unlock events. The program is then instrumented with the effects and compiled. The second part of the tool is a runtime system—a pthread library replacement—that uses the continuation effects to avoid deadlocks. When a thread attempts to take a lock L, the runtime computes the "future lockset" of L: this is the set of locks that may be acquired before L is released. The thread can acquire L only if all the locks in its future lockset are available. Experiments show that this technique imposes only a modest overhead.

In the second talk, Jack Sun from the University of Toronto presented Recon, a framework for protecting the integrity of filesystem metadata. Recon operates between a filesystem and an underlying block-storage layer: it imposes a set of rules at this interface to enforce filesystem consistency at run time. Jack's talk focused on the programming-language aspects of Recon, which are implemented in Datalog. As changes occur to the filesystem, Recon computes change records and represents these as facts in a Datalog environment. Consistency rules are also written in Datalog: they are predicates that evaluate to true when a consistency violation occurs. Recon is currently implemented for a test user-level filesystem, and their experience with Datalog has been promising. In the future, Jack and his coauthors intend to integrate Recon into a hypervisor.

Lokesh Gidra turned the workshop focus toward performance—in particular, toward the performance scalability of concurrent garbage collectors. Lokesh and his colleagues studied the behavior of the concurrent collectors in the OpenJDK 7 Java Virtual Machine running on a NUMA machine with 48 cores (8 nodes with 6 cores each). From their experiments, they concluded that the existing collectors do not scale with the number of GC threads. Lokesh identified three issues that hindered scalability. The first was the cost of remote scanning, which was frequent and not NUMA-aware. The second was the cost of remote object copying, which was also frequent and expensive because of a lack of node affinity between GC threads and their local allocation buffers. The third issue was the implementation of load balancing between GC threads.

The last talk was given by Ana-Maria Visan from Northeastern University, who presented URDB: a universal reversible debugger. A reversible debugger allows one to "go backward" in the execution of the software under examination. URDB is layered on top of existing debuggers to add reversibility; Visan and her colleagues have done this for GDB, Python, MATLAB, and Perl, and they claim that reversibility can be added to a debugger in less than a day. URDB's reversibility is based on checkpointing, restarting, and re-executing debugger-session histories. It also requires decomposing debugger actions so that commands such as "reverse step" can work when the previous command was not "step." Visan concluded with experimental results that show that URDB runs at acceptable, interactive speeds. On one benchmark, URDB was 5,200 times faster than the target-record mode of GDB 7.2.

4.4 Demonstrations

Following the paper presentations, the PLOS attendees turned their attention to demonstrations. Four of the systems described in the paper sessions were demonstrated during the workshop. Suman Saha presented his tool that analyzes Linux code for resource-release omission faults. At the same time, Reinhard Tartler demonstrated the UNDERTAKER toolchain and showed how he and his coauthors used it to obtain the configuration-coverage measures in their paper. Konstantinos Sagonas demonstrated his group's system for avoiding deadlocks. Using a small test program, he showed that the test was prone to deadlocks when run with the normal pthread library, and that the test always ran without deadlocks when it used his new runtime. Finally, Ana-Maria Visan presented URDB and used it to run test programs in reverse.

4.5 Working Group: Systems-Extensible Languages

After the demonstrations, participants organized themselves into three working groups to discuss interesting topics that had emerged earlier in the workshop. The first working group discussed the idea of systems-extensible programming languages. As evidenced by Alexander Warg's presentation earlier in the day (and previously by others [6, 16]), systems programmers often need to control the low-level implementation details of programming-language abstractions. Therefore, mechanisms that provide machine-specific information to compilers that help and guide their static analyses are necessary. Such information can be expressed, for instance, in the form of custom compiler attributes, pointer annotations, or even in the form of small domain-specific languages. Additionally, system programmers sometimes want to add new abstractions to existing languages [2, 5, 8], especially related to concurrency primitives. Support for compile-time introspection could be a good step in this direction. The working group discussed the state of the art and opportunities for future work in these areas.

4.6 Working Group: Filesystem Invariants

Motivated by Jack Sun's presentation, the second working group focused on filesystem invariants. The group members organized their discussion around four increasingly difficult goals for using invariants in the context of filesystems. The first was to detect and eliminate bugs in filesystems: this is the current focus of the Recon system presented earlier in the day. Recon only examines metadata; the group discussed how rules could protect file data as well. The second goal was to isolate untrusted filesystems. In contrast to the first goal, where the filesystem is generally trusted but may have bugs, the group discussed the idea of using rules to contain untrusted filesystem code. The third goal was to use invariants to implement an existing filesystem: i.e., to synthesize an implementation of a current filesystem from a specification. Work toward this goal could be informed by previous work on driver specification [13, 15]. The fourth and final goal was to use invariants to specify and implement new filesystems. Such filesystems might be open to managing application-defined consistency properties, such as those explored by Featherstitch [7].

4.7 Working Group: HW/SW Interfaces and Verification

The third working group discussed the role that the hardware-software interface plays in the formal verification of systems. This topic was motivated by Benoît Montagu's presentation of the SAFE platform and by Gernot Heiser's keynote.

The working group discussed the potential for both hardware and software innovation to improve system verifiability. A major concern was the performance costs incurred by past systems that used specialized hardware to implement capabilities. The group recommended investing effort into making any such hardware mechanisms competitive with modern TLBs. Nevertheless, the group felt it less important to produce a working prototype that was as fast as commercial hardware in all respects; the goal should be for researchers to demonstrate the potential for competitive performance. As examples of recent hardware research to achieve industry attention, the group pointed out virtualization [1], transactional memory [4], and the OpenFlow extensible routing architecture [12]. Such examples demonstrate the continuing possibility of innovation in hardware features without compromising performance.

5. Conclusion

Like previous editions of PLOS, the Sixth Workshop on Programming Languages and Operating Systems was a great success. Almost forty attendees participated in a daylong program of research presentations, focused discussions, system demonstrations, and working groups. The organizers believe that they achieved their goal of providing a venue for emerging research at the intersection of OS development and programming language development. They hope to see you at a future edition of PLOS!

Acknowledgments

Many people contributed to the success of PLOS 2011. We thank everyone who submitted papers and everyone who participated in the workshop—without their efforts and interest, there would be no workshop at all. We also thank the members of the program committee for their careful work in selecting this year's workshop program. Chris Hawblitzel and Konstantinos Sagonas assisted with the working-group summaries in this report. We also owe special thanks to the SOSP 2011 organizing committee. Ramakrishna Kotla and Rodrigo Rodrigues handled the myriad details of the workshop program overall, and Andrew Birrell performed many tasks in support of the workshops and the publication of the workshops' proceedings. Finally, we thank ACM SIGOPS for its sponsorship.

References

[1] K. Adams and O. Agesen. A comparison of software and hardware techniques for x86 virtualization. In *Proc. ASPLOS*, pages 2–13, Oct. 2006.

[2] Z. Anderson et al. SharC: Checking data sharing strategies for multithreaded C. In *Proc. PLDI*, pages 149–158, June 2008.

[3] Y. Chen et al. Surviving sensor network software faults. In *Proc. SOSP*, pages 235–246, Oct. 2009.

[4] D. Dice et al. Early experience with a commercial hardware transactional memory implementation. In *Proc. ASPLOS*, pages 157–168, Mar. 2009.

[5] M. Fähndrich et al. Language support for fast and reliable message-based communication in Singularity OS. In *Proc. EuroSys*, pages 177–190, Apr. 2006.

[6] D. Frampton et al. Demystifying magic: High-level low-level programming. In *Proc. VEE*, pages 81–90, Mar. 2009.

[7] C. Frost et al. Generalized file system dependencies. In *Proc. SOSP*, pages 307–320, Oct. 2007.

[8] D. Gay et al. The nesC language: A holistic approach to networked embedded systems. In *Proc. PLDI*, pages 1–11, June 2003.

[9] G. Klein et al. seL4: Formal verification of an OS kernel. In *Proc. SOSP*, pages 207–220, Oct. 2009.

[10] P. Levis et al. T2: A second generation OS for embedded sensor networks. Technical Report TKN–05–007, Telecommunication Networks Group, Technische Universität Berlin, Nov. 2005.

[11] D. Lohmann et al. Aspect-aware operating system development. In *Proc. AOSD*, pages 69–80, Mar. 2011.

[12] N. McKeown et al. OpenFlow: Enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74, Mar. 2008.

[13] F. Mérillon et al. Devil: An IDL for hardware programming. In *Proc. OSDI*, pages 17–30, Oct. 2000.

[14] N. Palix et al. Faults in Linux: Ten years later. In *Proc. ASPLOS*, pages 305–318, Mar. 2011.

[15] L. Ryzhyk et al. Automatic device driver synthesis with Termite. In *Proc. SOSP*, pages 73–86, Oct. 2009.

[16] J. Shapiro. Programming language challenges in systems codes: Why systems programmers still use C, and what to do about it. In *Proc. PLOS*, Oct. 2006.

[17] J. Yang and C. Hawblitzel. Safe to the last instruction: Automated verification of a type-safe operating system. In *Proc. PLDI*, pages 99–110, June 2010.

[18] F. Zhou et al. SafeDrive: Safe and recoverable extensions using language-based techniques. In *Proc. OSDI*, pages 45–60, Nov. 2006.