

# Implementing High-performance In-kernel Network Services With WYKIWYG\*

Sapan Bhatia Charles Consel

LaBRI/INRIA  
Domaine Universitaire  
33400 Talence, FRANCE  
{bhatia,consel}@labri.fr

## Abstract

This paper introduces the design philosophy and underlying principles of WYKIWYG, a language to implement high performance network services. A WYKIWYG compiler, unlike compilers for traditional languages, is empowered with the knowledge of underlying OS mechanisms such as task management, memory management, the device I/O interface *etc.* generates code which is specifically optimized for these mechanisms, and can even go as far as modifying or extending them in a controlled manner. Preliminary experiments with manually compiled code show that such code can improve the throughput of network services by up to an order of magnitude.

## 1 Context

Over the decades, one of the primary goals of OS design has been to allow the safe deployment of untrusted code. This strategy enables rapid deployment of new application code while at the same time ensuring that anomalies resulting from programming errors or malicious intent do not affect the system as a whole. The most common implementation of this separation, which is commonly assisted by modern processors, is to run untrusted applications in non-overlapping virtual name spaces. In this way, application processes have separate virtual memory address spaces, a separate set of file descriptors *etc.* which are distinct from the name spaces in which critical OS code is implemented.

Although it yields powerful abstractions, this separation of *user space* from *kernel space* has its disadvantages. Isolating applications using virtual spaces not only interposes the overhead of *mapping dilemmas* [8] (the effort needed to

---

\*This research has been partially funded by the Conseil Régional d'Aquitaine.

reify generic behavior), but also closes OS mechanisms off to applications. Applications, thus, need to invoke OS mechanisms as atomic, immutable entities, and can at best configure them using coarse grained policing interfaces. There have been many efforts to build systems that allow applications to tune kernel mechanisms to their needs to various degrees [5, 2, 13, 6, 1].

## 2 Our approach

In our approach, we use a strategy to push critical functionalities of applications into privileged *kernel-space* using a domain-specific language [4]. We are defining a language named WYKIWYG (WYKI, henceforth), for writing network services, which is designed to facilitate high performance network servers, while at the same time being safe. The following paragraphs discuss its salient features.

**Fast servers with WYKI.** A WYKI compiler is empowered with intimate knowledge of OS mechanisms, and generates highly optimized code that blends into these mechanisms by extending them to use platform-specific optimizations, bypassing interfaces and levels of abstraction when necessary, performing domain-specific optimizations, *etc.*

**Program safety with WYKI.** Like many other Domain Specific Languages (DSLs) [11, 9], WYKI is designed to be safe. WYKI-generated code is guaranteed to preserve certain program properties, such as termination, type safety and memory safety, which make it possible to run WYKI-generated code as privileged OS code.

**OS consistency with WYKI.** Since a WYKI compiler is OS-aware, the code produced respects certain properties of the underlying OS mechanisms, such as fairness. This maintains the consistency of underlying OS mechanisms, and preserves the overall consistency of the OS.

## 3 Illustration by example

This section introduces select optimizations used in WYKI. These optimizations are facilitated by domain specific knowledge of application behavior and the fact that OS mechanisms are open and fully exposed to a WYKI compiler.

### 3.1 Memory management

WYKI is designed to use its domain-specific knowledge of service-level memory management to optimize the allocation and use of memory. When possible, superpages [7] are used as units of allocation for effective TLB utilization. Since

the superpage allocator in WYKI is application specific, it circumvents several issues associated with using superpages [12], such as rapid fragmentation. Application knowledge also enables more effective cache utilization.

### 3.2 Thread management

Most network servers are implemented using threads, in order to support concurrent sessions. The creation, scheduling and synchronization of threads carries an overhead. WYKI optimizes away this overhead by doing away with threads by collapsing them into language-level control structures. Properties associated with threads, such as fairness and preemptability are imposed by the WYKI compiler, in a cooperative spirit.

### 3.3 Socket management

Sockets are communications end-points that processes use to transfer data with other processes (local or remote). The use of sockets not only results in overheads [3] but also entails signaling between the application and kernel, which can be expensive [10]. WYKI generates code which collapses the sockets implementation, in much the same way as it collapses threads, by using program-level conditionals and interacting with the kernel scheduler, instead of using high-level mechanisms such as signals and system calls.

## 4 Conclusion and current position

WYKI is a domain-specific language used to implement safe in-kernel servers. WYKI compilers are OS-specific, and use their intimate knowledge of underlying OS mechanisms to generate code, which is not only optimized itself, but also optimizes these mechanisms by extending, altering, or in certain cases, bypassing them. Preliminary experiments show that services produced using such techniques are up to an order of magnitude faster than industry standard implementations thereof.

## References

- [1] Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, Nathan C. Burnett, Timothy E. Denehy, Thomas J. Engle, Haryadi S. Gunawi, James A. Nugent, and Florentina I. Popovici. Transforming policies into mechanisms with infokernel. In *SOSP03*, 2003.
- [2] B.N. Bershad, C. Chambers, S. Eggers, C. Maeda, D. McNamee, P. Pardyak, S. Savage, and E. Gün Sirer. SPIN – an extensible microkernel for application-specific operating system services. Technical Report 94-03-03, Department of Computer Science and Engineering, University of Washington, February 1994.

- [3] S. Bhatia, Consel C., LeMeur A.F, and C. Pu. Tool-based specialization of protocol stacks in OS kernels. Research report, LaBRI, November 2003.
- [4] C. Consel and R. Marlet. Architecturing software using a methodology for language development. In C. Palamidessi, H. Glaser, and K. Meinke, editors, *Proceedings of the 10th International Symposium on Programming Language Implementation and Logic Programming*, volume 1490 of *Lecture Notes in Computer Science*, pages 170–194, Pisa, Italy, September 1998.
- [5] D. Engler. *The Exokernel Operating System Architecture*. PhD thesis, MIT, Cambridge, MA, USA, 1998.
- [6] B. Ford, G. Back, G. Benson, J. Lepreau, A. Lin, and O. Shivers. The Flux OSKit: A substrate for kernel and language research. In *Proceedings of the 16th ACM Symposium on Operating Systems Principles*, pages 38–51, St-Malo, France, October 1997.
- [7] N. Ganapathy and C. Schimmel. General purpose operating system support for multiple page sizes. In *USENIX In Proceedings of the USENIX 1998 Annual Technical Conference, Berkeley, CA*, 1998.
- [8] Gregor Kiczales, John Lamping, Chris Maeda, David Keppel, and Dylan McNamee. The need for customizable operating systems. In *Workshop on Workstation Operating Systems*, pages 165–169, 1993.
- [9] Julia L. Lawall, Gilles Muller, and Luciano Porto Barreto. Capturing OS expertise in a modular type system: the Bossa experience. In *Proceedings of the ACM SIGOPS European Workshop 2002 (EW2002)*, pages 54–62, Saint-Emilion, France, September 2002.
- [10] D. McNamee, J. Walpole, C. Pu, C. Cowan, C. Krasic, C. Goel, C. Consel, G. Muller, and R. Marlet. Specialization tools and techniques for systematic optimization of system software. *ACM Transactions on Computer Systems*, 19:217–251, May 2001.
- [11] F. Méryllon, L. Réveillère, C. Consel, R. Marlet, and G. Muller. Devil: An IDL for hardware programming. In *Proceedings of the Fourth Symposium on Operating Systems Design and Implementation*, pages 17–30, San Diego, California, October 2000.
- [12] Juan Navarro, Sitaram Iyer, Peter Druschel, and Alan Cox. Practical, transparent operating systems support for superpages. In *OSDI00*, 2002.
- [13] M. I. Seltzer, Y. Endo, C. Small, and K. A. Smith. Dealing with disaster: Surviving misbehaved kernel extensions. In *Proceedings of the 2nd Symposium on Operating Systems Design and Implementation*, pages 213–227, Seattle, Washington, October 1996.