Singularity: Designing Better Software (Invited Talk)

James R. Larus

Microsoft Research
One Microsoft Way
Redmond WA 98052
larus@microsoft.com
http://research.microsoft.com/~larus

Five years ago, frustrated by the never-ending process of finding bugs that developers had cleverly hidden throughout our software, I started a new project with Galen Hunt to rethink what software might look like if it was written, from scratch, with the explicit intent of producing more robust and reliable software artifacts. The Singularity project [1] in Microsoft Research pursued several novel strategies to this end. It has successfully encouraged researchers and product groups to think beyond the straightjacket of time-tested software architectures, to consider new solutions that cross the bounds of academic disciplines such as programming languages, operating systems, and tools.

Singularity built a new operating system using a new programming language, new software architecture, and new verification tools. The Singularity OS incorporates a system architecture based on software isolation of processes. Sing#, the programming language is an extension of C# that provides pre- and post-conditions; object invariants; verifiable, first-class support for OS communication primitives; and strong support for systems programming and code factoring.

From its start, the Singularity project was driven by the question of what would a software platform look like if it was designed with the primary goal of improving the reliability and robustness of software? To this end, we adopted three strategies. First, Singularity is almost entirely written in a safe, modern programming language, which eliminates many serious defects such as buffer overruns. Second, the system architecture limits the propagation of runtime errors by providing numerous, inexpensive, well-defined failure boundaries, thereby making it easier to achieve robust and correct system behavior, even in the presence of imperfect software. Finally, Singularity was designed from the start to facilitate the widespread use of sound program verification tools, with the belief that these tools could provide strong guarantees that entire classes of errors were eliminated.

The success of Singularity raises the possibility that it is time to rethink the traditional design, architecture, and construction practices for software in light of its increasingly central role in the world and the unprecedented threats to its security and integrity. It also poses interesting questions about today's balance

A. Gupta and S. Malik (Eds.): CAV 2008, LNCS 5123, pp. 1–2, 2008.

of effort between finding defects in existing software and developing the next generation of languages and tools, which could make a qualitative improvement in software robustness. The advent of parallel programming, occasioned by the Multicore revolution, makes these changes even more relevant, as this sea change opens the door for other radical changes in software.

References

1. Hunt, G., Larus, J.: Singularity: Rethinking the Software Stack. Operating System Review 41, 37–49 (2007)