Singularity

Hack the Planet linked to an interesting MS Research project called Singularity. From the blurb:

> Singularity is a research project in Microsoft Research that started with the question: what would a software platform look like if it was designed from scratch with the primary goal of dependability? Singularity is working to answer this question by building on advances in programming languages and tools to develop a new system architecture and operating system (named Singularity), with the aim of producing a more robust and dependable software platform. Singularity demonstrates the practicality of new technologies and architectural decisions, which should lead to the construction of more robust and dependable systems.

The result (so far) looks pretty interesting, particularly in light of my own recent comments about microkernels. Their design is definitely that of a microkernel, with components represented by Software Isolated Processes or SIPs, which run in a single address space. The kernel (and, I assume, a core set of SIPs) is written in a "safe" language called Sing#, which is derived from another MSR project called Spec#. In fact, leveraging other MSR projects seems to be quite a theme in Singularity; they also use the Boxwood filesystem, for example. Spec# looks like a very interesting language, with built-in support for verification of program properties way beyond the mere type-checking you find in most languages. There are specific features for null-pointer handling, safe exceptions, and detailed specification of Eiffel-style "contracts" between functions. The specific difference between Sing# and Spec# is that it extends this validation even to protocol-level interactions between components (e.g. "if I send you an X message I expect either a Y or Z in response") which is very cool. To a large degree these features are implemented via static code analysis rather than run-time checks, so the impact on performance should be small.

In many ways Singularity looks like the kind of OS I might have designed. They pretty much prove my point about a microkernel design implemented within a single address space with lightweight RPC, for starters. Their work also addresses my point about safe languages. Such languages are great, but actually implementing a kernel in an interpreted/JIT-compiled fashion would be insane. Singularity's approach of relying on a big dose of static verification to enforce safety properties in code that then runs in fully compiled form seems like very much the right way to go about things, and the limitations placed on SIPs make perfect sense in a context of keeping the system reliable.

There are only two things that I would consider disappointing about Singularity. If they're going to all this trouble to create an OS geared toward dependability, it seems odd that they're not more actively working toward a distributed system that can survive hardware as well as software faults. Maybe they just decided that the scope of the project was large enough already and they'll leave that part for later, which is fine, but it seems like it should merit at least a mention. The other thing is not so much a disappointment as an area where further improvement can be made. The protocol-level checks in Sing# are great, but incomplete in two ways. One, which the authors admit, is that the verifier checks for invalid state transitions but not deadlock/livelock. The other is that, while the protocol contract addresses the issue of running but possibly malfunctioning components, it does little to address what should (or should not happen) when a component outright fails and is removed from the system. The behavior of other components that were interacting with the failed one should also be specified and enforced, though the exact details of that specification are obviously up for debate.

Quibbles aside, though, Singularity looks like a very interesting piece of work. I look forward to seeing what directions they take it from here.