

# The Singularity Project

Ryan Yates

12-8-2011

# The Singularity Project

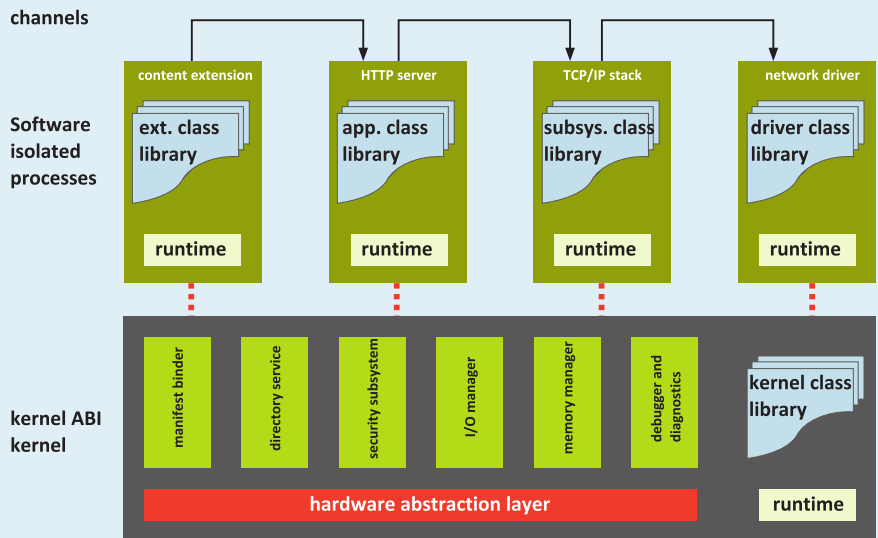
- ▶ Microsoft Research Project 2004–2006
- ▶ Jim Larus and Galen Hunt

# The Singularity Project

## Goals of the Singularity Project[6]:

- ▶ Use safe high-level programming languages to the greatest extent possible.
- ▶ Software failure should not lead to system failure.
- ▶ Systems should be self-describing at all levels of abstraction.

figure 1. Structure of a Singularity system.



# Software Isolated Process

- ▶ Instead of using hardware protection mechanisms handle protection in software.
- ▶ Use programming languages as an aid.

# Software Isolated Process

- ▶ Statically ensure that bad things cannot happen
  - ▶ Prove it cannot happen
  - ▶ Check that it is not about to happen
- ▶ Verify code before running

# Software Isolated Process

- ▶ What are the normal hardware and software costs in memory protection?

# Software Isolated Process: Creation

Process create and start

# Software Isolated Process: Creation

Process create and start

OS	Cycles
Singularity	353,000
freeBSD 5.3	1,030,000
Linux 2.6.11 (Red Hat FC4)	719,000
Windows XP (SP2)	5,380,000

# Software Isolated Process: Syscall

Minimum Kernel API Call

# Software Isolated Process: Syscall

## Minimum Kernel API Call

OS	Cycles
Singularity	91
freeBSD 5.3	878
Linux 2.6.11 (Red Hat FC4)	437
Windows XP (SP2)	627

# Software Isolated Process: Context Switch

Thread context switch

# Software Isolated Process: Context Switch

## Thread context switch

OS	Cycles
Singularity	346
freeBSD 5.3	911
Linux 2.6.11 (Red Hat FC4)	906
Windows XP (SP2)	753

# Software Isolated Process: IPC

Message Request/Reply [1]

# Software Isolated Process: IPC

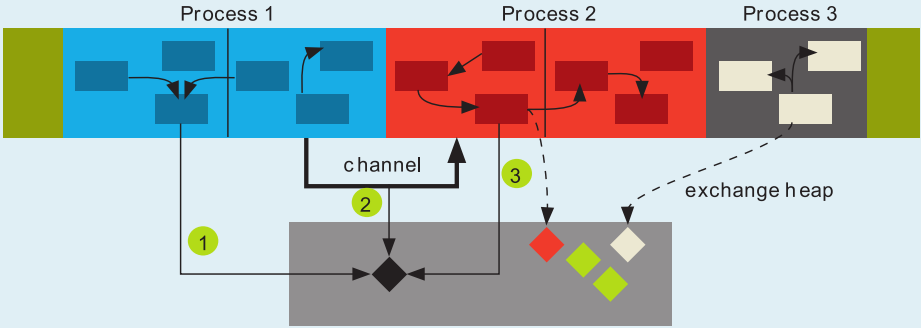
Message Request/Reply [1]

OS	Method	Cycles
Singularity	Channel	803
freeBSD 5.3	Socket	13,300
Linux 2.6.11	Socket	5,800
Windows XP	Named Pipe	6,340

## Contract-Based Channels [3]

- ▶ Contract protocols are statically checked.
- ▶ Messages in channels are well typed.
- ▶ Memory allocated in exchange heap is only referenced by a single process at a time.

figure 2. Singularity process objects reside on a dedicated collection of pages.



# C# Spec# Sing#

- ▶ Type-safe
- ▶ Garbage Collection

## Spec#: Method Contracts

- ▶ Requirements can be met statically (using a theorem prover [2] [7]) or by inserting runtime checks (with a failure path that does not violate protection) [3].

## Spec#: Method Contracts

```
int ISqrt(int x)
  requires 0 <= x;
  ensures result*result <= x
    && x < (result+1)*(result+1);
{
  int r = 0;
  while ((r+1)*(r+1) <= x)
    invariant r*r <= x;
  {
    r++;
  }
  return r;
}
```

See: <http://rise4fun.com/SpecSharp>

## Sing#: Contract-Based Channels

Example contract: [5]

```
contract NicEvents
{
    enum NicEventType {
        NoEvent, ReceiveEvent,
        TransmitEvent, LinkEvent
    }
    out message NicEvent(NicEventType e);
    in message AckEvent();
    state READY: one {
        NicEvent! -> AckEvent? !READY;
    }
}
```

figure 3. hybrid hardware-software isolation using SIPs and domains.

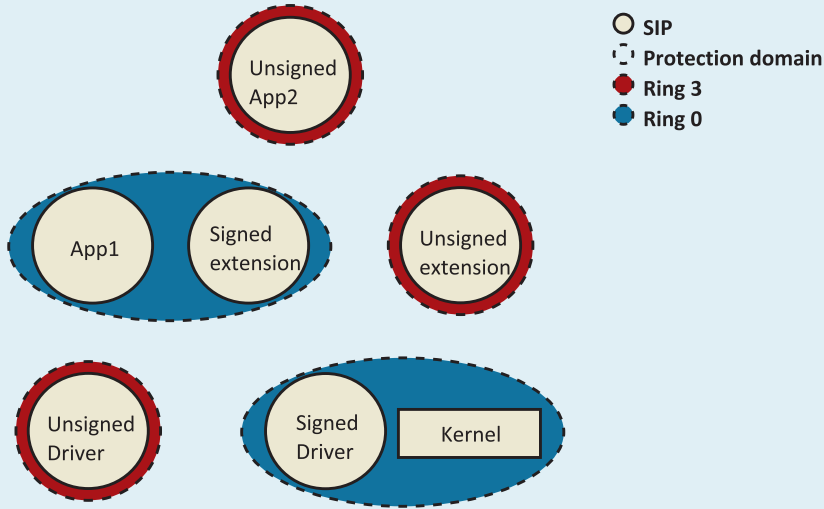
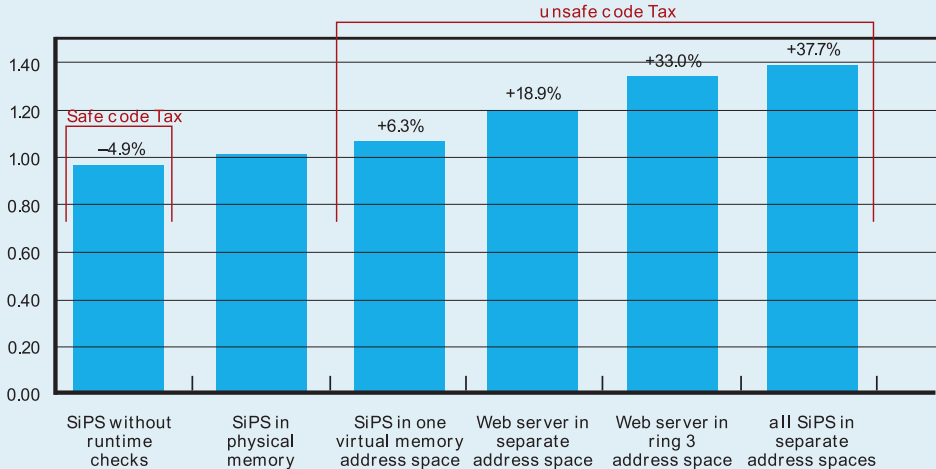


figure 4. normalized execution time comparing the overhead cost of software and hardware process isolation mechanisms for a Web server running on Singularity. our experiments ran on a 1.8Ghz amd athlon 64 3000+ system, starting with a pure software-isolated version of Singularity, progressively adding hardware address-space protection.



# Drawbacks

- ▶ Programming model
  - ▶ Multiple type systems
  - ▶ Porting required for benefits
- ▶ Kernel GC

# More Developments

- ▶ Verve — Mechanically verified operating system [8].
  - ▶ Type-safe
  - ▶ Memory safe
  - ▶ Practical verified mixing of high-level code and untyped assembly.
- ▶ Boogie — Intermediate verification language [7].
- ▶ Contracts — Get some of these benefits with out invasive language changes [4].

- [1] Mark Aiken, Manuel Fähndrich, Chris Hawblitzel, Galen C. Hunt, and James R. Larus.  
Deconstructing process isolation.  
In *Memory System Performance*, pages 1–10, 2006.
- [2] L. M. de Moura and N. Bjørner.  
Z3: An efficient smt solver.  
In *Tools and algorithms for the construction and analysis of systems.*, pages 337–340, 2008.
- [3] Manuel Fähndrich, Mark Aiken, Chris Hawblitzel, Orion Hodson, Galen Hunt, James R. Larus, and Steven Levi.  
Language support for fast and reliable message-based communication in singularity os.  
*SIGOPS Oper. Syst. Rev.*, 40:177–190, April 2006.
- [4] Manuel Fähndrich, Michael Barnett, and Francesco Logozzo.  
Embedded contract languages.  
In *ACM Symposium on Applied Computing*, pages 2103–2110, 2010.

- [5] Galen C. Hunt and James R. Larus.  
Singularity: rethinking the software stack.  
*SIGOPS Oper. Syst. Rev.*, 41:37–49, April 2007.
- [6] James R. Larus and Galen C. Hunt.  
The singularity system.  
*Communications of The ACM*, 53:72–79, 2010.
- [7] K. Rustan M. Leino.  
This is boogie 2, 2008.
- [8] Jean Yang and Chris Hawblitzel.  
Safe to the last instruction: automated verification of a  
type-safe operating system.  
In *SIGPLAN Conference on Programming Language Design  
and Implementation*, pages 99–110, 2010.