We Crashed, Now What?

Cristiano Giuffrida Lorenzo Cavallaro
Andrew S. Tanenbaum

Vrije Universiteit Amsterdam

6^{th} Usenix Workshop on Hot Topics in System Dependability
October 3, 2010, Vancouver, BC, Canada

A problem has been detected and Windows has been shut down to prevent damage
to your computer.

DRIVER_IRQL_NOT_LESS_OR_EQUAL

If this is the first time you've seen this Stop error screen,
restart your computer, If this screen appears again, follow
these steps:

Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.

If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.

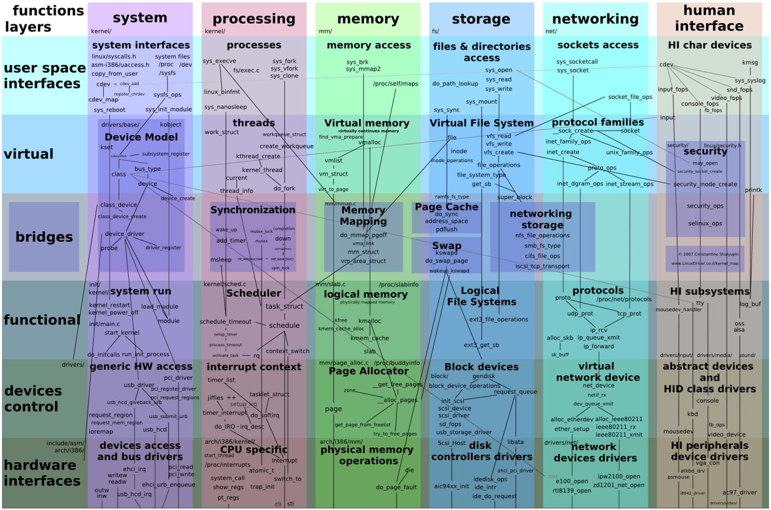
Technical information:

*** STOP: 0x000000D1 (0x0000000C,0x00000002,0x00000000,0xF86B5A89)

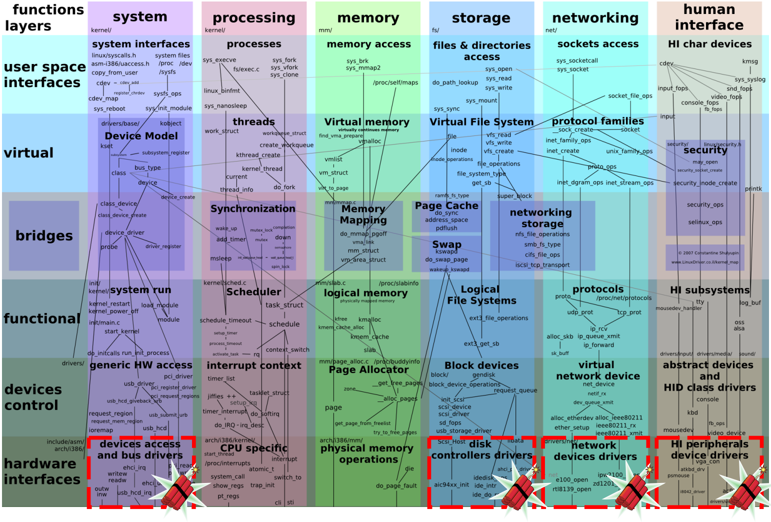
*** gv3.sys - Address F86B5A89 base at F86B5000, DateStamp 3dd991eb

Beginning dump of physical memory
Physical memory dump complete.
Contact your system administrator or technical support group for further
assistance.

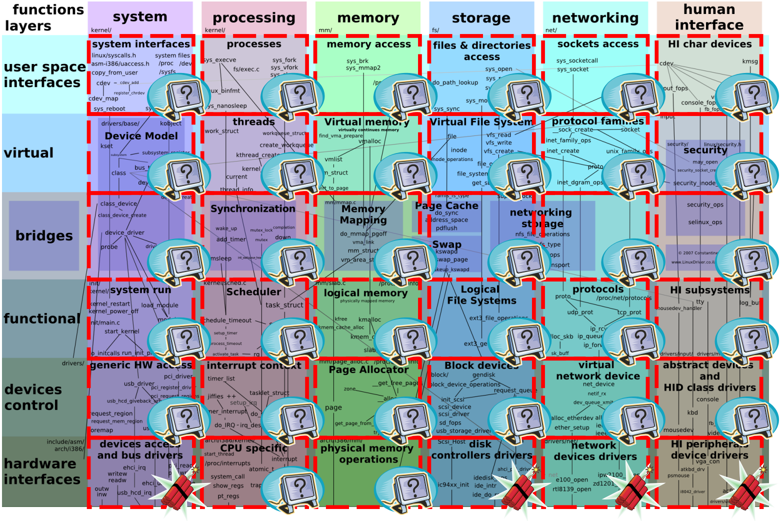
OS Dependability Threats



OS Dependability Threats



OS Dependability Threats



Are Core Components Safe?

Are Core Components Safe?

"We're getting bloated and huge.

Are Core Components Safe?

*"We're getting bloated and huge.
Yes, it's a problem.*

Are Core Components Safe?

"We're getting bloated and huge.
Yes, it's a problem.
[. . .] *I'd like to say we have a plan."*

Are Core Components Safe?

"We're getting bloated and huge.
Yes, it's a problem.
[...] *I'd like to say we have a plan."*

Linus Torvalds on the Linux kernel, 2009

High-coverage Crash Recovery

- Rapid evolution and huge size cause more bugs
- Crash recovery solution with smaller TCB needed
- Whole-OS crash recovery

High-coverage Crash Recovery

- Rapid evolution and huge size cause more bugs
- Crash recovery solution with smaller TCB needed
- Whole-OS crash recovery

How?

High-coverage Crash Recovery

- Rapid evolution and huge size cause more bugs
- Crash recovery solution with smaller TCB needed
- Whole-OS crash recovery

How?

1. Extend existing work on isolated subsystems to the entire OS

High-coverage Crash Recovery

- Rapid evolution and huge size cause more bugs
- Crash recovery solution with smaller TCB needed
- Whole-OS crash recovery

How?

1. Extend existing work on isolated subsystems to the entire OS
2. Design a new high-coverage crash recovery infrastructure

Isolated Subsystems $\overset{?}{\rightsquigarrow}$ Entire OS

- Work on extensions and drivers
  - e.g., *Safedrive*, *Nooks*, *Minix 3*
- Filesystems
  - e.g., *Membrane*
- Assume isolated untrusted parties with well-defined interfaces
- Several recoverer-recoveree pairs to scale to the entire OS
  - Complex and hard-to-maintain recovery infrastructure
- High exposure of the recovery code to the programmer

Isolated Subsystems $\overset{?}{\leadsto}$ Entire OS

- Work on extensions and drivers
  - e.g., *Safedrive, Nooks, Minix 3*
- Filesystems
  - e.g., *Membrane*
- Assume isolated untrusted parties with well-defined interfaces
- Several recoverer-recoveree pairs to scale to the entire OS
  - Complex and hard-to-maintain recovery infrastructure
- High exposure of the recovery code to the programmer

. . . it is like a dog chasing its tail!

Emerging High-coverage Solutions

Shadow kernel **vs** **Pure instrumentation**
e.g., Otherworld **e.g., Recovery Domains**

Emerging High-coverage Solutions

Shadow kernel **vs** **Pure instrumentation**

e.g., Otherworld **e.g., Recovery Domains**

↗

- Best-effort
  (weak failure model)

Emerging High-coverage Solutions

Shadow kernel **vs** **Pure instrumentation**

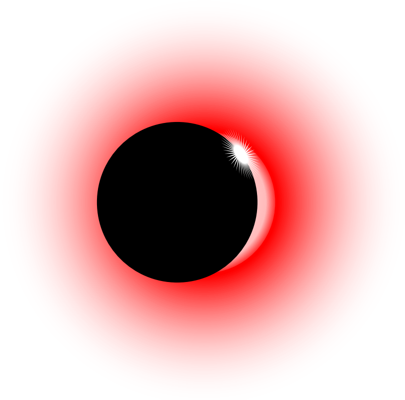
e.g., Otherworld **e.g., Recovery Domains**

- Best-effort
  (weak failure model)

- Heavyweight
  (high complexity)
  (poor performance)
  (poor scalability)

WWW: What We Want

WWW: What We Want



High coverage

WWW: What We Want

Low complexity

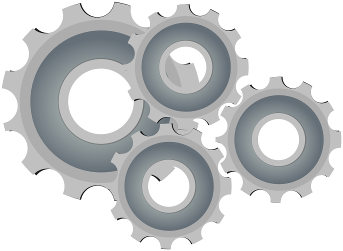
WWW: What We Want

Reasonable performance and scalability

WWW: What We Want

Good maintainability

WWW: What We Want



Address the many challenges of the crash recovery problem

The Crash Recovery Problem — I

Crash detection

- Detect crashes proactively or reactively
- Isolate crashes so they do not disrupt the recovery process

The Crash Recovery Problem — I

Crash detection

- Detect crashes proactively or reactively
- Isolate crashes so they do not disrupt the recovery process

State transfer

- Create a new execution context to restart execution
- Transfer the state from the old execution context

The Crash Recovery Problem — I

Crash detection

- Detect crashes proactively or reactively
- Isolate crashes so they do not disrupt the recovery process

State transfer

- Create a new execution context to restart execution
- Transfer the state from the old execution context

State consistency

- Restore a stable and consistent state in the new context
- Allow for deterministic execution upon restart

The Crash Recovery Problem — II

State dependency tracking

- Preserve state dependencies among different contexts
- Allow for a globally coherent state upon restart

The Crash Recovery Problem — II

State dependency tracking

- Preserve state dependencies among different contexts
- Allow for a globally coherent state upon restart

State corruption

- Detect arbitrary data corruption
- Attempt to recover from arbitrary data corruption

The Crash Recovery Problem — II

State dependency tracking

- Preserve state dependencies among different contexts
- Allow for a globally coherent state upon restart

State corruption

- Detect arbitrary data corruption
- Attempt to recover from arbitrary data corruption

Restart

- Determine a safe execution point to resume operation
- Attempt to avoid further crashes

Our Approach

Combine OS design and lightweight instumentation

Our Approach

Combine OS design and lightweight instumentation

OS Design

- Reduce complexity at recovery time
- Good performance and scalability

Our Approach

Combine OS design and lightweight instumentation

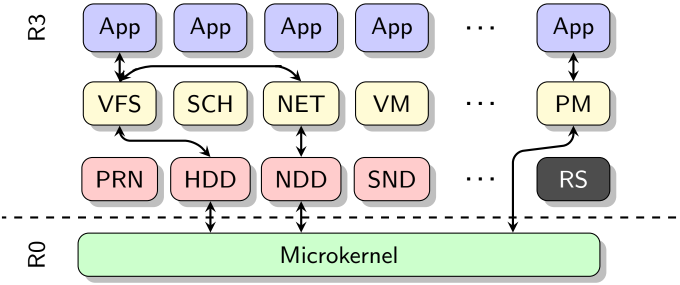
OS Design

- Reduce complexity at recovery time
- Good performance and scalability

Lightweight Compiler-based Instrumentation

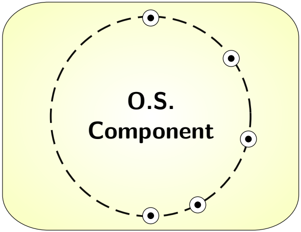
- High coverage and component-agnostic recovery
- Good maintainability and evolvability

OS Architecture



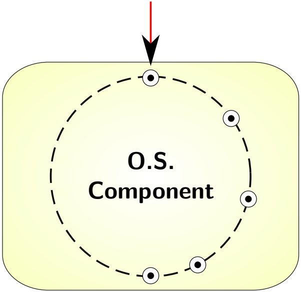
- We break down the OS into several userspace components
- Multiserver microkernel architecture based on message-passing

The Programming Model



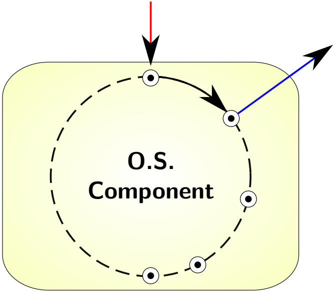
We rely on an event-driven model

The Programming Model



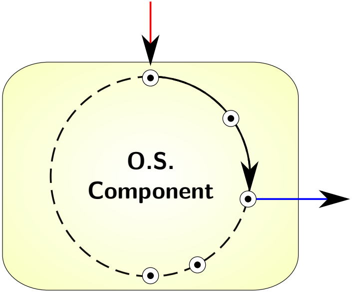
Events trigger execution of the task loop

The Programming Model



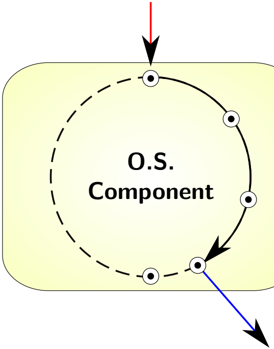
Idempotent messages possible within the task loop

The Programming Model



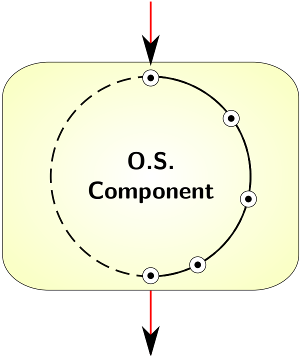
Idempotent messages possible within the task loop

The Programming Model



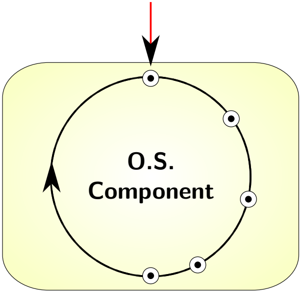
Idempotent messages possible within the task loop

The Programming Model



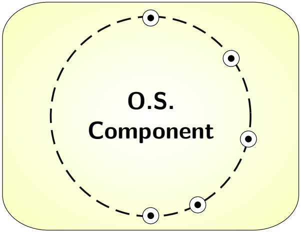
Push non-idempotent messages to the end

The Programming Model



Back to the top of the loop!

The Programming Model



Pending interactions are remembered in the state

State Management

- Identify state of data and state of execution
- Both well-defined and consistent at the top of the task loop
- The top of the loop is a local stable state point
- Global state consistency by design

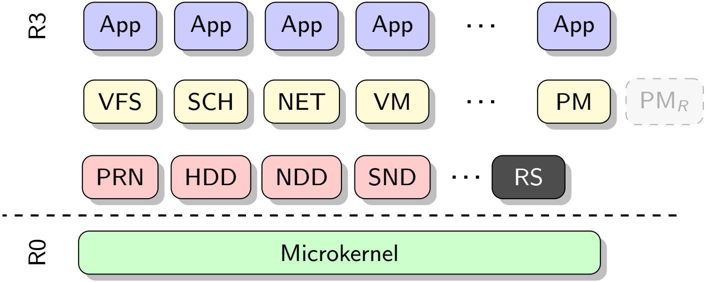
Instrumentation-based Recovery

- The task loop is the recovery window
- Lightweight instrumentation to track local state changes
- Used by the recovery code to revert to the last stable state
- Different strategies possible

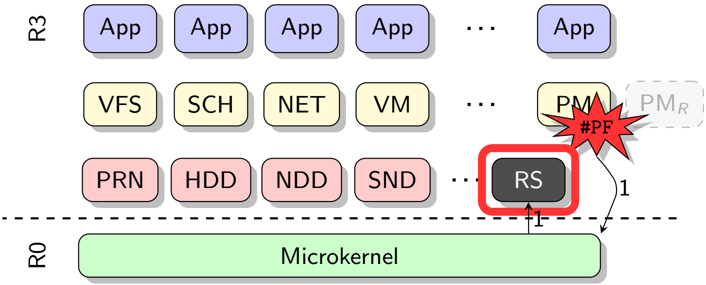
Our Implemented Instrumentation

- Maintain shadow state regions
- Track dynamic memory allocations
- Track changes on state objects
- Use alias analysis to detect changes at the object granularity
- Automatically commit changes at the end of the task loop
  (i.e., it synchronizes shadow and main state regions)

The Crash Recovery Process



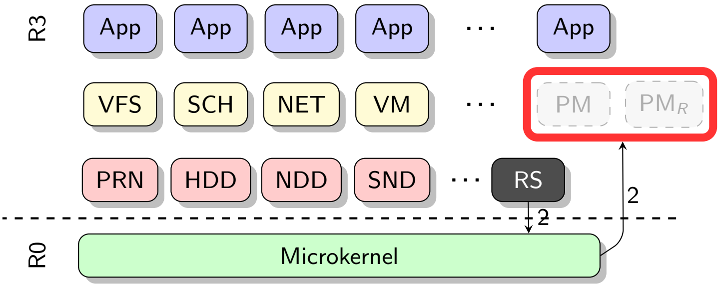
The Crash Recovery Process



An OS component crashes: the system manager detects the crash and initiates recovery

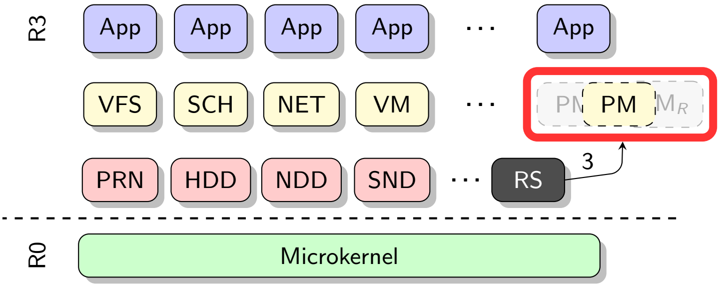
(the microkernel actually signals the system manager)

The Crash Recovery Process



The system manager selects a new replica
and tells the microkernel
(virtual ids make transparent recovery possible!)

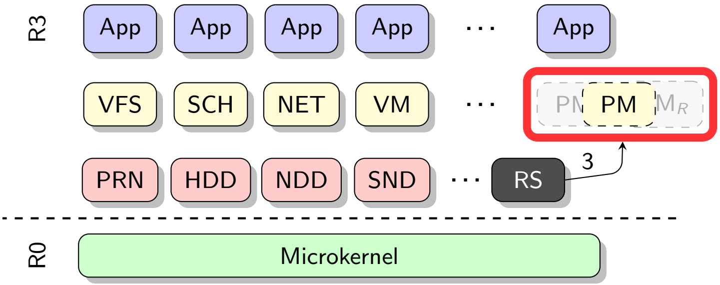
The Crash Recovery Process



The system manager yields control to the new replica for state transfer. . .

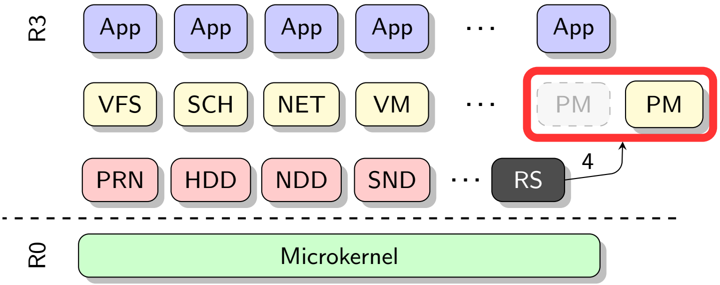
(libary-based recovery code starts executing. . .)

The Crash Recovery Process



. . . the component is brought back to
the last stable state and resumes operation
(shadow and main state regions are synced!)

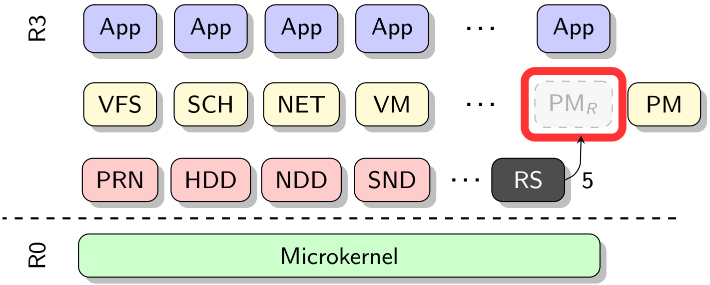
The Crash Recovery Process



The system manager cleans up the dead replica

(the new replica may even be involved in the process!)

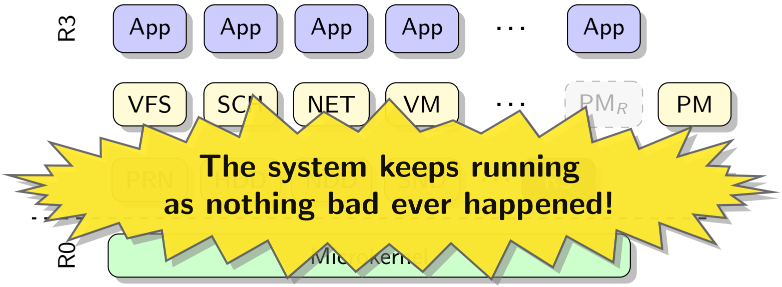
The Crash Recovery Process



The system manager spawns a new replica (if needed)

(per-component recovery policies apply)

The Crash Recovery Process



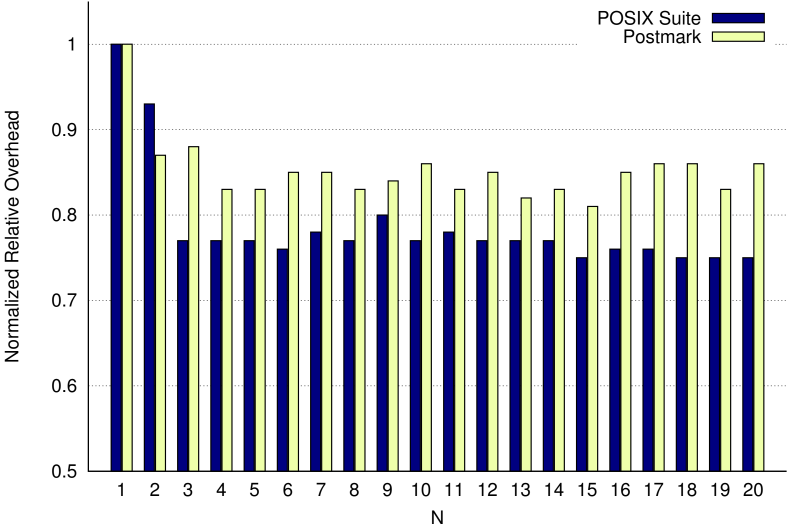
The system manager spawns a new replica (if needed)

(per-component recovery policies apply)

Prototype

- Implemented on top of MINIX 3
- Restructured OS processes to fit our event-driven model
- Instrumentation implemented as a series of LLVM passes
- Successfully recovered even the most critical components
- Early experiments confirmed key properties of our design

Scalability Properties



Summary

- A new high-coverage approach to OS crash recovery
- Combines OS design and compiler-based instrumentation
- Low complexity, good performance, scalability, maintainability
- No heavy burden for the OS programmer
- Addresses many of the crash recovery challenges efficiently

Future Work

- Finer-grained instrumentation to track the state
- Realistic fault injection scenarios
- Experiment and evaluate restart strategies
- Recover from state corruption
- Per-component recovery policies

We Crashed, Now What?

Thank you!
Any questions?

Cristiano Giuffrida, Lorenzo Cavallaro, Andy Tanenbaum
{giuffrida,sullivan,ast}@cs.vu.nl

Vrije Universiteit Amsterdam