


# **“Otherworld” - giving applications a chance to survive OS kernel crashes**

*Alex Depoutovitch and Michael Stumm  
University of Toronto*



# Software faults

- That faults in software are an unavoidable fact that we have to cope with
- From 30 to 50 percent of computer systems TCO is spent on recovering from and preparing for faults  
(Patterson et al.)
- IBM and Microsoft calls for switching focus from faster systems to more reliable systems

# Kernel faults - particularly severe

- Kernel crash causes all application data to be lost
- Techniques that minimize consequences of fault:
  - Checkpointing
    - Introduces overhead
  - Redundant calculations
    - Increase system cost and complexity
  - Micro-kernels, software fault isolation
    - Not directly applicable to commodity OSes, introduces overhead

# Consequences of kernel faults

- Kernel state is corrupted and can't be trusted
  - Kernel state corruption is mostly restricted to faulty modules
  - 70-85% of errors are introduced by faulty drivers
- All running application are affected
- Application state is rarely corrupted in kernel faults:
  - Application memory corrupted in less than 18% of cases
  - Application-specific recovery reduces this number to 1-2%

**In 98% of the kernel faults application would be able restore its data if it is given a chance**

# Key idea of “Otherworld”

## OS kernel:

- Just component of a software system
- Logically well isolated from other components

It should be possible to reboot the kernel without destroying everything else running on the same system.

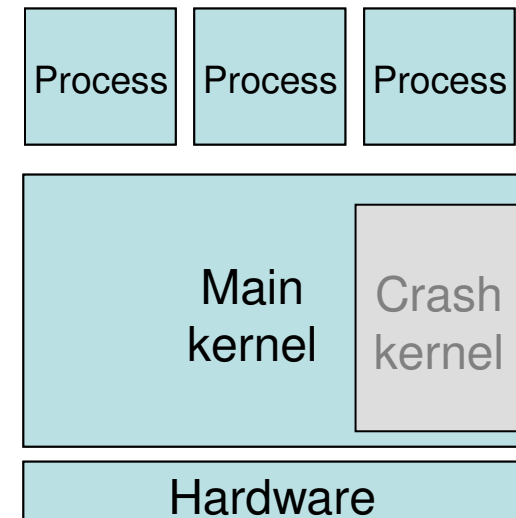
(kernel reboot vs. system reboot)

# Challenges of the kernel reboot

- Kernel contains data critical for all applications
  - Physical memory pages
  - Location of paged-out data
  - Open files
  - Network sockets
  - etc.
- Need to have software component, which is able to manage system after the fault

# “Otherworld” architecture

- Two kernels
  - Main – active
  - Crash – dormant, protected, uninitialized
- In case of a fault control transferred to Crash kernel
- Crash kernel initializes itself
- Gets information about processes
- Continues to run processes



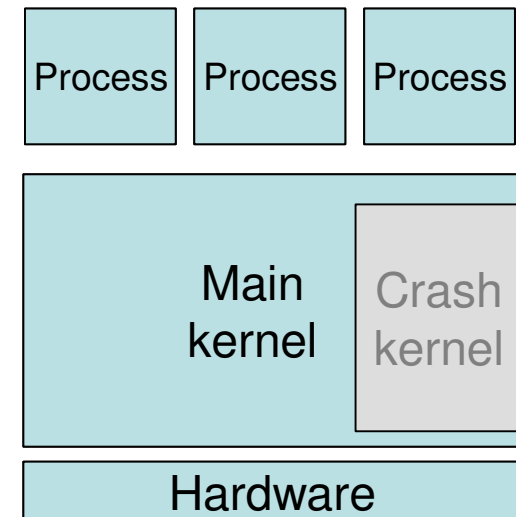
# Benefits of “Otherworld”

- No run-time overhead
- Fixed and small memory overhead
- Applicable to monolithic and microkernel OSes
- Small changes to OS and applications code
- No specialized or redundant hardware
- Restored state is the most recent
- Amount of state that can be restored is not limited



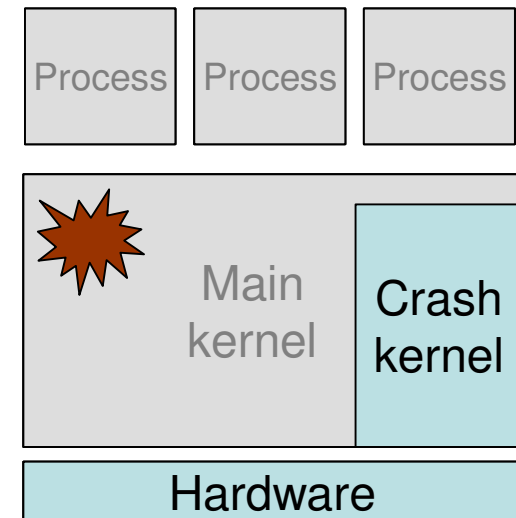
# Normal functioning

- Main kernel boots
- Reserves space for Crash kernel
- Loads Crash kernel image
- Runs processes
- Process register Crash Procedure



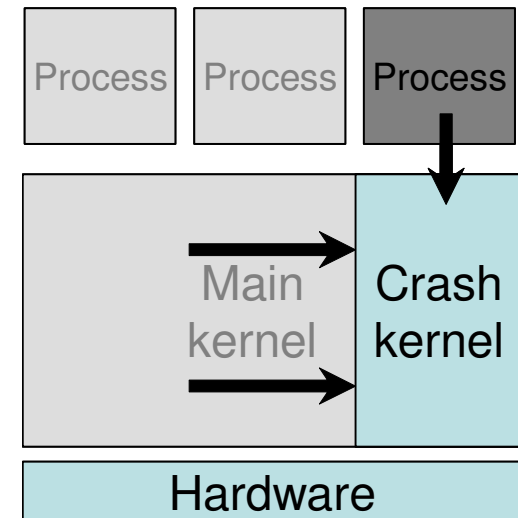
# Kernel fault occurs

- Control is passed to Crash Kernel
- Crash kernel initializes itself
- Memory is restricted to reserved region
- Result:
  - Initialized undamaged kernel
  - Main kernel and processes memory preserved and accessible



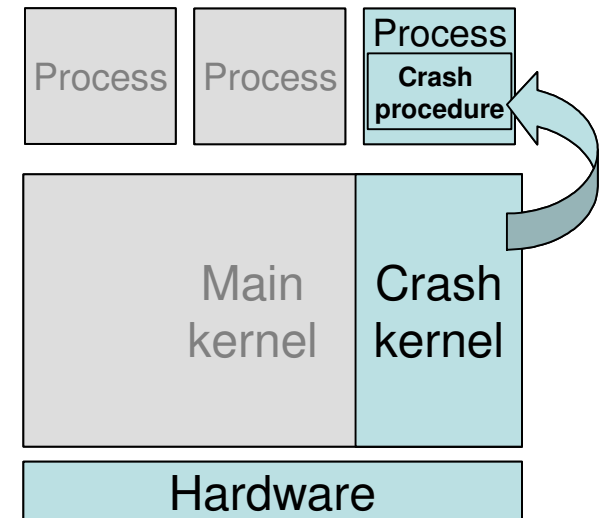
# Retrieving information

- Crash kernels recovers process information for each process
- Checksums and data redundancy can be used for corruption detection
- New process is created
- Address space is copy of Main kernel process
- Resources are restored



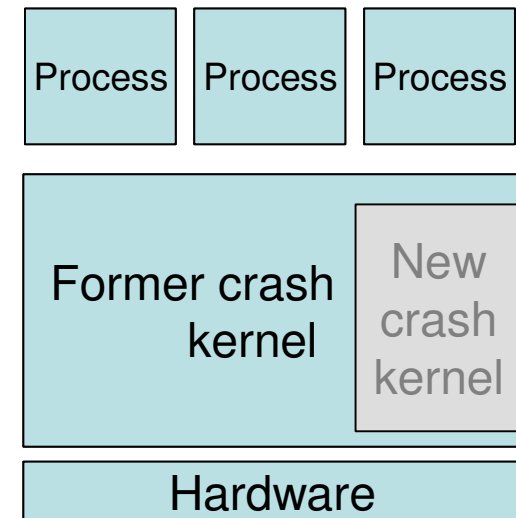
# Running crash procedure

- Crash kernel restores process and calls Crash procedure (Resurrection)
- Crash procedure saves information to disk or continues application execution
- Crash procedure - analog of application exception handler



# Post-recovery steps

- After application resurrection is complete:
  - Reclaims all remaining memory
  - Loads another crash kernel
  - Continues regular activities
- Fully functional system



# Automatic resources resurrection

Automatically restored:

- Application physical memory pages
- Pages swapped to disk
- Memory mapped files
- Open files

Not restored:

- Network connections
- Pipes
- Screen content
- Threads

# Evaluation

- Platform
  - Linux 2.6.18
    - (KDump is used for loading crash kernel)
- Applications
  - JOE text editor
  - MySQL database server
  - Apache/PHP
  - Berkeley Lab Checkpoint/Restart
- Kernel crashes
  - Triggering existing BUGON() asserts in kernel

# JOE – text editor

- Terminal based, multidocument text editor
  - 30,000 lines of code
- Crash procedure
  - 25 lines of code
  - Goes through list of opened documents
  - Calls existing save function for every document
  - Restarts text editor with the documents
- For text editor user kernel fault is transparent



# MySQL – database server

- Popular open source database server
  - 700,000 lines of code
  - Supports memory-resident tables
  - Amount of required changes: 75 lines of code
- Crash procedure (50 line)
  - Calls MySQL functions to retrieve in-memory data
  - Saves the data to disk
  - Restarts the server
- Start-up code (25 line)
  - Reads saved data
  - Populates in-memory tables
  - Continues normal server execution

# Applications of “Otherworld”

- Servers:
  - Reliable in-memory databases
    - 1.5-140 times faster than disk resident
  - Reliable in-memory web session data
    - 25% faster than storing sessions on disk
- Scientific applications:
  - Checkpointing without overhead
  - Reliable in-memory checkpointing
- Interactive application:
  - Editors that restores the document up to the last symbol entered

# Open problems

- Detection and prevention of data corruption
- Uninterrupted application execution
- Resurrection of other resources (e.g. sockets)
- Resurrection of a group of interacting programs

# Conclusion

“Otherworld” is a fault recovery technique:

- Allows applications and application data to survive kernel crash
- Requires only minor changes to the kernel and applications
- No run-time overhead
- Applicable to wide range of applications

# Questions



# Probability of application data corruption

OS	Application data corruption	Bug types	Comments
MVS <sup>[1]</sup>	6%	Real	Sample of 240 error reports out of 3000
BSD 4.x <sup>[2]</sup>	2%	Real	
Linux <sup>[3]</sup>	10%	Artificial	35,000 injected bugs
Linux <sup>[4]</sup>	Application generic: interactive <b>18%</b> non-interactive <b>2%</b>  Application specific: interactive <b>2%</b> non-interactive <b>1%</b>	Artificial	400 injected bugs
SunOS <sup>[5]</sup>	8%	Artificial	500 injected bugs

## Conclusions:

- After OS failure: application memory corrupted in 1-18% of cases
- Non-interactive applications have lower chance of memory corruption
- Data corruption rarely occur outside OS kernel component that contains bug

[1] M. Sullivan and R. Chillarege, Software defects and their impact on system availability: A study of field failures in operating systems.

[2] M. Baker et al., Non-volatile memory for fast, reliable file systems.

[3] W. Gu et al., Characterization of linux kernel behavior under errors.

[4] S. Chandra and P. M. Chen, The impact of recovery mechanisms on the likelihood of saving corrupted state.

[5] W. Kao et al., FINE: A Fault Injection and Monitoring Environment for Tracing the UNIX System Behavior Under Faults

# Joe text editor

- Crash procedure:

```
int ow_crash_procedure(void* unused) {
    B *b;
    int saved=0, size=10;
    char** params=(char**)malloc(sizeof(char*)*size);
    params[0]=program_location;
    params[1]=NULL;
    b=&bufs;
    do {
        if (b->name) {
            bsave(b->bof,b->name,b->eof->byte,1);
            params[++saved]=b->name;
            params[saved+1]=NULL;
            if (saved>=size-2) {
                params=(char**)realloc(params,size*2);
                size*=2;
            }
        }
        b = b->link.next;
    } while (b->link.next!=&bufs);
    execl(program_location,params);
    return 0;
}
```

# Comparison to CuriOS

	Otherworld	CuriOS
Applicable to existing OSes	+	-
No run-time overhead	+	-
Reduction of error propagation	-	+
Recovery speed	+/-	+
Application transparency	-	+/-

- Once error propagates to other subsystems CuriOS is difficult to recover
- Otherworld starts with new clean kernel



# Comparison to checkpoints

- Otherworld has no overhead
- Otherworld has always the latest state
- Both approaches have probability of application data corruption

# Objectives of “Otherworld”

- Allow applications and application data to survive OS crashes
- Desired properties:
  - Can be used with existing OS architecture
  - No significant changes to OS or applications code
  - Negligible run-time overhead
  - No specialized or redundant hardware