Building a Self-Healing Operating System

Francis David Roy Campbell

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

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

The problem seems to be caused by the following file: SPCMDCON.SYS

PAGE_FAULT_IN_NONPAGED_AREA

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:

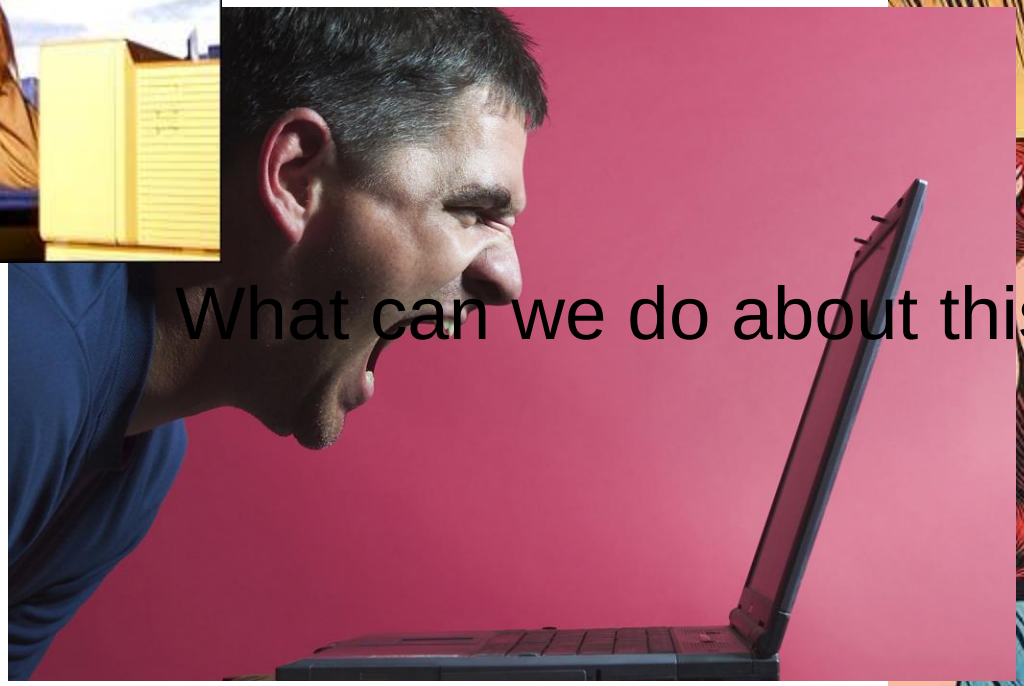
*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)

*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

```
program smartd is using a deprecated SCSI ioctl, please convert it to SG_IO
program smartd is using a deprecated SCSI ioctl, please convert it to SG_IO
program smartd is using a deprecated SCSI ioctl, please convert it to SG_IO
program smartd is using a deprecated SCSI ioctl, please convert it to SG_IO
program smartd is using a deprecated SCSI ioctl, please convert it to SG_IO
program smartd is using a deprecated SCSI ioctl, please convert it to SG_IO
program smartd is using a deprecated SCSI ioctl, please convert it to SG_IO
program smartd is using a deprecated SCSI ioctl, please convert it to SG_IO
program smartd is using a deprecated SCSI ioctl, please convert it to SG_IO
program smartd is using a deprecated SCSI ioctl, please convert it to SG_IO
BUG: unable to handle kernel paging request at virtual address 78bc78bb
 printing eip:
eabb8e00
*pde = 2fd3a001
*pte = 00000000
Oops: 0002 [#1]
SMP
Modules linked in: drbd ipmi_si ipmi_devintf ipmi_msghandler
CPU:    2
EIP:    0060:[<eabb8e00>]    Not tainted VLI
EFLAGS: 00010286   (2.6.18-pe-sm-tmm-20061030-vs #1)
EIP is at 0xeabb8e00
eax: eabb8e00   ebx: c5833900   ecx: c5927a40   edx: d742a000
esi: eabb8e00   edi: 00000007   ebp: 00000000   esp: d742beec
ds: 007b   es: 007b   ss: 0068
Process readpgenv (pid: 5839[#0], ti=d742a000 task=f08ba030 task.ti=d742a000)
Stack: c0129cc3 eabb8e00 c5833980 00000000 00000008 c0129eaa c011facf 00000000
       00000001 c068c328 00000008 c011f743 c068c328 00000002 00000000 00000046
       b7f8c4f8 c0110910 0805ae70 c011f7fd d742bfbc c01032d7 d742bfbc 0000007b
Call Trace:
 [<c0129cc3>] __rcu_process_callbacks+0xfd/0x162
 [<c0129eaa>] rcu_process_callbacks+0x20/0x40
 [<c011facf>] tasklet_action+0x55/0xb0
 [<c011f743>] __do_softirq+0x79/0xfd
 [<c0110910>] do_page_fault+0x0/0x582
 [<c011f7fd>] do_softirq+0x36/0x3a
 [<c01032d7>] apic_timer_interrupt+0x1f/0x24
 [<c0110910>] do_page_fault+0x0/0x582
 [<c011007b>] early_vga_write+0x51/0x102
 [<c0110a68>] do_page_fault+0x158/0x582
 [<c012421c>] sys_rt_sigaction+0x68/0x7e
 [<c0110910>] do_page_fault+0x0/0x582
 [<c0103389>] error_code+0x39/0x40
Code: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
 00 00 <c4> 79 65 c9 c9 8f 15 c0 00 00 00 00 00 00 00 00 40 04 60 c0 00
EIP: [<eabb8e00>] 0xeabb8e00 SS:ESP 0068:d742beec
 <0>Kernel panic - not syncing: Fatal exception in interrupt
```

The All-Important OS

- All applications are dependent on the OS
- When the OS dies, all running applications are lost



What can we do about this?

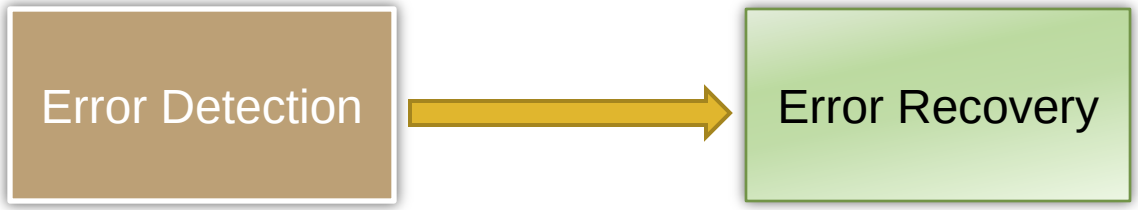
Outline

- Terminology
- Error Detection in Existing Operating Systems
- Error Recovery in Existing Operating Systems
- The Choices Operating System
- Error Signaling and Confinement
- Error Detection and Recovery
- Concluding Remarks

Terminology

- Fault – Defect or flaw in hardware or software
- Error – Deviation from correct state
- Failure – Inability to perform expected task

- How to prevent system failures from occurring?



Error Detection in Existing OSs

- Custom Error Detection Code in OSs
  - Linux – Deadlock Detection, Soft Lockup Detection etc
  - Windows – Deadlock Detection etc

- Hardware Memory Protection – MMU
- Software Memory Protection – SafeDrive, XFI
- Periodic Consistency Checks – EROS
- Watchdog Timers – Linux, Windows etc

Error Recovery in Existing OSs

- Linux – Oops! and recovery by terminating thread
- Restart Failed Component
  - Windows Vista – Example: Video Card Driver
  - Minix3
  - Chorus
  - Linux+Nooks
  - IBM z/OS
- Hardware Redundancy
  - Process Pairs: HP NonStop Servers
- Reboot Entire System

The Choices Operating System

- Object Oriented
- Written in C++
- OS Components are Objects
  - Kernel
  - CPU
  - Process
  - etc.
- Runs on Sparc, x86, ARM
- Will focus on ARM hardware in this work

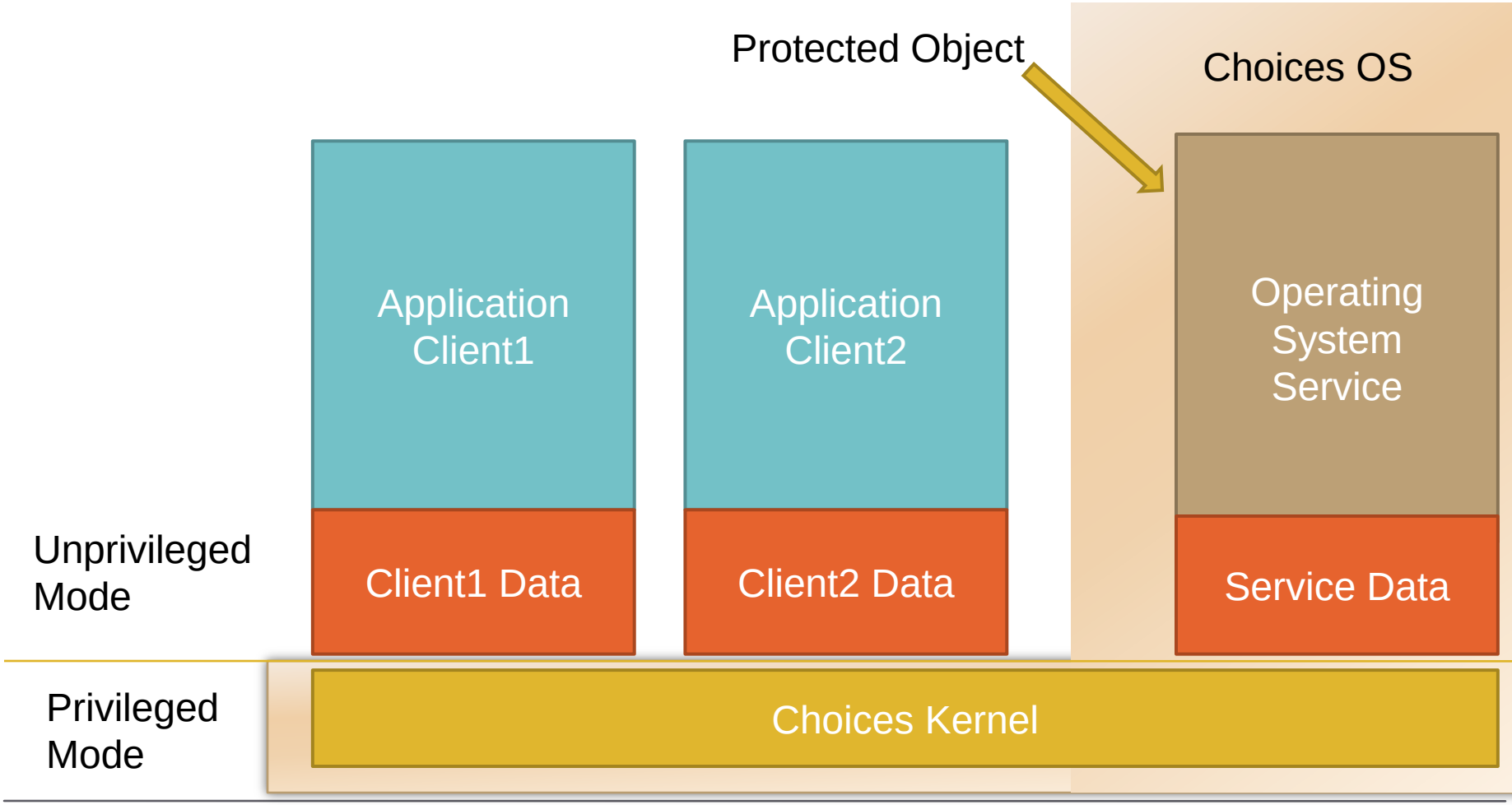
Error Signaling

- C++ Exception Handling for Unified Error Signaling
  - Developer defined exceptions
  - Processor exceptions
- Benefits of mapping processor exceptions to language exceptions
  - Local error recovery using C++ “catch” statements
  - Generic handlers for all types of exceptions
- Normal run-time performance overhead is negligible

Error Confinement

- Isolate OS components
- Used by some microkernels: L4, Minix3
- Nooks: Device Driver Isolation in Linux
- Objects in Choices can be placed in separate memory protection domains
- Implemented using wrappers which inherit from target Classes
- Example Protected Objects: Serial Port Driver, FileSystem Inodes, Timer Driver

Choices Protected Components



Error Detection & Recovery

- Code Reloading
- Component Micro-Reboots
- Transactional Components
- Automatic Service Restarts
- Watchdog-based Recovery
- Process-level Recovery

Code Reloading

- Fault: Corruption of OS code by software bugs or hardware bit-flips (Single Event Upsets)
- Proactive Recovery: Periodically checksum OS code and reload corrupted pages from stable storage
- Reactive Recovery: If undefined instruction exception is raised, reload relevant OS page from stable storage
- Simple fault-injection experiments show 89% recovery

Component Micro-Reboots

- Error: Unhandled Exceptions in Components
- Recovery: Similar to component restarts in existing systems
- Involves destroying & re-creating C++ object
- Re-create in-place for references to remain valid
- After “micro-reboot”, internal state may be error free
- Request is re-tried after micro-reboot
- Works well for stateless components

Transactional Components

- Error: Unhandled Exceptions in Component
- Recovery: Use transactional object support to roll back object state
- Alternative to micro-reboot that does not lose state
- Investigated by deploying Software Transactional Memory (STM) support in Choices
- Simple fault injection experiments show 100% recovery
- Drawbacks: More expensive than micro-reboot in terms of performance & memory usage

Automatic Service Restarts

- Error: Unhandled Exception in a Process
- Recovery: Automatically restart process
- Used when component level restarts fail or if error occurs outside components (framework code)
- Fault injection experiments show 78.9% recovery for process dispatcher (idle thread)

Watchdog-based Recovery

- Error: Lockups inside OS
- Recovery: Terminate locked up thread or dispatch exception
- Thread termination explored on Linux
- Exception Handling explored in Choices
- Exceptions allow possible local recovery without any information loss (in contrast with thread termination)
- Lockup fault injection experiments about 70% recovery

Process-level Recovery

- What to do when OS error recovery is not possible?
- Last Resort
  - Ensure minimal working subsystems – disk, recovery code
  - Save individual process state
  - Restore processes after full reboot
- Explored on Linux
- Re-use code for process checkpointing/migration support
- Can recovery from arbitrary OS corruption that does not affect user process state

Our (New) Contributions

- Exception Handling for unified error signaling
- Code-Reloading for proactive & reactive OS Recovery
- Watchdog-based Recovery
- Process-level Recovery

Future Work

- Working on OS restructuring to reduce error propagation and prevent state loss during component micro-reboots
- Framework for developer specified policies to govern micro-reboots and service restarts

Concluding Remarks

- Self-Healing Operating Systems may be built by incorporating a variety of recovery techniques to address different fault models
- Recovery may be combined with filesystem snapshot technology if data corruption issues are a concern

For More Information

- Website: http://choices.cs.uiuc.edu
- Related Papers

  Exception Handling in the Choices Operating System

  Advanced Topics in Exception Handling, Springer-Verlag, 2006

  Exploring Recovery from Operating System Lockups

  USENIX 2007

  Improving Dependability by Revisiting Operating System Design

  HotDep 2007