Ongoing evolution of Linux x86 machine check handling

Sept. 2009

Andi Kleen

LinuxCon 2009

What's a good error?

- User has to see it, of course
  - That can be surprisingly difficult
  - Also psychological barriers (“users don't read errors”)
- High level classification
  - Software error versus hardware error
  - Don't want a hardware error reported as a kernel bug
  - Still have low level details for experts (ideally separated)
- Identify affected component
  - Do not require low level knowledge to process
  - Works out of the box
- How serious is the error?
  - Do not upset the user unnecessarily

Error sources

- Machine checks from the CPU
- NMI
- PCI-Express Advanced Error Handling (AER)
- Chipset
- ACPI4 (APEI)
- Drivers
  - SATA errors
  - Ethernet
  - ...
- Presentation only covers machine check errors from the CPU
  - This includes memory on modern systems

What is a machine check?

- Machine check is a hardware error reported by the CPU
  - Not a software problem!
- Hardware corrects most problems, but sometimes it can fail
  - Memory, Interconnect, Cache, Internal errors
- Uncorrected errors raise exceptions (“MCEs”)
  - Better to stop than to continue with corrupted data
  - Otherwise the corrupted data could hit disk or give wrong results
  - If you aren't sure where it is, stop the machine ...
- Corrected errors are reported in the background
  - Either using a poll handler or with interrupts (“CMCI”)
  - Didn't cause corruption (yet)

Why are machine checks important?

- They report memory errors on modern systems
  - Memory error rate scaling roughly with memory sizes
  - Memory sizes are increasing quickly
    - More cores need more memory
    - Virtualization needs a lot of memory
  - This also means more memory errors
  - So good error handling is important
- On large clusters, errors are common
  - What's uncommon on a single system
  - … becomes common when you have a hundred of them
  - … and even very rare events become common on thousands of systems
- In general, good error diagnosis is useful
  - If you ever searched manually for a bad DIMM …
  - Saves time and hassle

MCE errors in practice today

- Error flows for uncorrected and corrected errors

- Assuming 64 bit or 2.6.31+ with CONFIG_X86_NEW_MCE
- Mcelog has to be installed

- Some of these flows are still somewhat clumsy
  - The future will be brighter, hopefully

Classic unrecoverable MCE error today

- System detects uncorrected error
  - Requires ECC for memory
  - Machine check exception happens
- Machine check handler collects error and prints it out
  - CPU x: Machine check exception ...
- System panics on unrecoverable error
  - On auto reboot (panic=30) can be logged after reboot on many systems
    - Available then decoded in /var/log/mcelog some time after boot
    - Trap: doesn't work with double reboot or with power switch
  - You will see the panic on the console (not in X) or on serial/netconsole
    - If you don't have auto reboot a logging console is very useful
    - Console output can be run through mcelog –-ascii to decode
- Analyze error, based on decoded output
  - For example, map to DIMM (“Memory DIMM ID of error: ...”)
  - If common, take corrective action

Corrected error flow today

- A data bit flips
  - Hardware detects error, using checksum, and corrects it and reports event
- CMCI interrupt happens or poll timer picks it up
  - Kernel logs it to internal buffer accessible through /dev/mcelog
    - Note this buffer can overflow
- Mcelog picks it up
  - Either as cronjob or in daemon mode or as trigger
    - In cronjob mode up to 10 minutes delay, worst case with polling
  - Mcelog decodes
  - Logged in /var/log/mcelog or syslog
- Analysis of the log entry
  - Identify component
  - Take corrective action if common

Low level MCE handler improvements

- General overhaul after comprehensive audit
  - A lot of small improvements, too many to list
- Monarch support: synchronization over all CPUs
  - Collect errors from all CPUs
  - Synchronize all CPUs
  - Process the most serious error first to avoid data corruption
    - When the hardware didn't contain the error shut down
- Bank sharing
  - Handle shared machine check banks correctly
- Corrected Machine Check Interrupt Support ("CMCI")
- Injector support for testing and a comprehensive test suite

MCA recovery

- New CPU feature in upcoming Nehalem-EX CPU
- Recovery from some memory uncorrected errors
  - For example, Patrol scrub memory error in the background
  - Required a lot of changes in the MCE handler to do reliable
  - When recovering it's much more important to handle all corner cases
- OS finds out what the corrupted page does
  - And attempts to get rid of it
- Machine check architecture has new status bits for recovery
  - Signalled, Action-Required
  - Different types of errors: Action-Optional, Action-Required, UCNA, Corrected

HWPoison handling in the VM

- VM finds out who owns a page and stops using it
  - Pages with copy on disk can be just dropped
  - Or application is killed, if data has no copy
  - IO error for dirty file cache pages
  - Free pages will be ignored on allocation
- Difficult because error can come in at any time
  - Can disrupt normal page livecycle
  - Error code has to be very careful
  - Also testing is difficult
- Put page on bad page list and never reuse it again
- Page table entry of any mappings is poisoned
- Early kill versus late kill mode

Virtualization

- Virtualization requires a lot of memory
- Many eggs in a single basket (“servers on a single machine”)
- Error handling and error containment is important
  - If there's a problem only kill single guest, not all
- KVM guests act like processes
  - So uses the per process infrastructure in standard hwpoison
- Uncorrected recoverable errors can be forwarded to a guest
  - Guest can inject the error as machine check (or KVM kills)
  - Guest with MCA recovery support can recover (or panic)
- Similar code developed for Xen
  - No forwarding for Xen currently

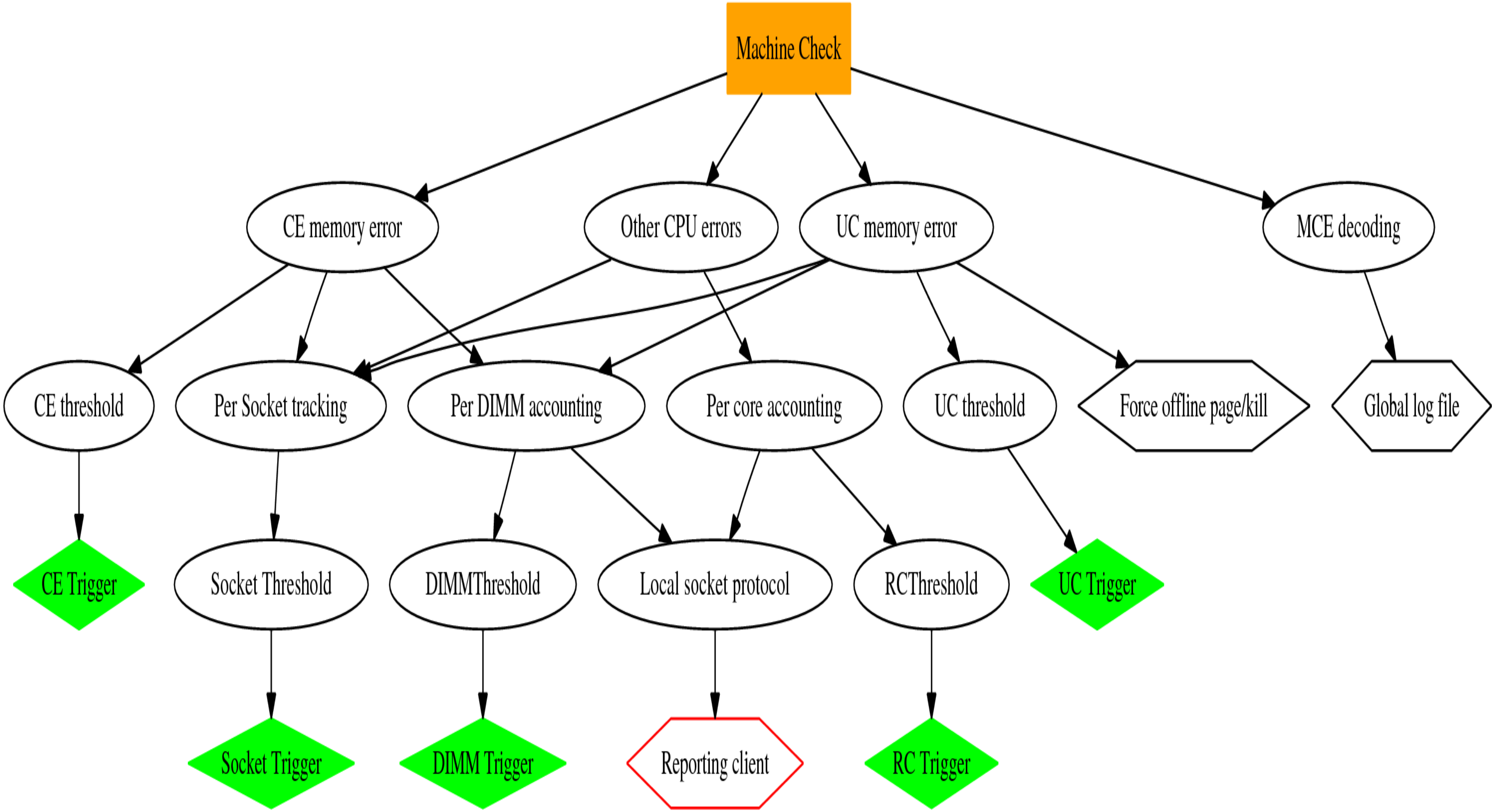
Memory error application interface

- Applications can catch memory errors, which are signals
  - Was needed for KVM, but can be used by others too
- Applications have often cache, which they can drop
- Expect only a few specialized applications to use it
- SIGBUS with an address can be caught
  - BUS_MCEERR_AO Action-Optional error in process
    - Get rid of page specified by si_addr, si_addr_lsb
    - Take out of free list or similar
  - BUS_MCEERR_AR Action-Required error in current execution context
    - Need to abort right now (siglongjmp etc.)
- Prctl to set early kill vs late kill for the process
  - Late kill is typically better for error aware applications

Mcelog

- User space backend that decodes and processes MCE errors
  - Also identifies components with some firmware help
- Traditionally on 64 bit x86, now on 32 bit too
- Traditionally cronjob every 5 minutes, future daemon
  - Daemon mode allows to keep state about errors in memory with query interface and triggers
  - Some old attempts using an on-disk database proved difficult
- Can run shell scripts on specific events (“triggers”)
  - Notify administrator, offline component, ...
- Long term goal: high level errors in syslog
  - Some steps into this direction

Mcelog error flow



Error accounting

- Possible in mcelog daemon mode
- Often most interesting is which component the error affects
  - DIMM, memory, PCI card, etc.
  - To see trends and replace the right component quickly if needed
- Individual errors are often not that interesting
  - Errors often come in bursts and individual errors in a burst are not interesting
  - Large clusters can generate a lot of data, which is difficult to process
- Mcelog moving towards accounting errors per component
  - Only reports "n errors in last x hours on component k"
  - Triggers when thresholds are exceeded
  - Discovers component names with firmware help
  - Can disable individual error logging for less data

Open issues

- Crashdump handling
- More testing is always useful
  - Stress test suite under way
  - Any contributions welcome
- Better error reporting in general
  - More high level errors better presented
- More error sources in mcelog
- Intelligent error handling in mcelog
  - If you have ideas, feel free to contact me

Resources

- http://halobates.de/mce.pdf
  - Old paper about Linux machine checks
- Intel Software Developer's Manual: 3A/B System Programming Guide
  - Description of the x86 Machine check architecture
- git://git.kernel.org/pub/scm/linux/kernel/git/ak/linux-mce-2.6
  - MCE development tree (hwpoison, mce4)
- git://git.kernel.org/pub/scm/utils/cpu/mcelog.git
  - Mcelog development repository
- ftp://ftp.kernel.org/pub/linux/utils/cpu/mce/
  - Mcelog releases
- git://git.kernel.org/pub/scm/utils/cpu/mce-test.git
  - MCE test suite (or now part of LTP to (http://ltp.sourceforge.net)
- git://git.kernel.org/pub/scm/utils/cpu/mce-inject.git
  - MCE injector

Backup

Mcelog versus EDAC

- EDAC old style driver for chipset memory controllers
  - Exposes a lot of low level details
  - New model: memory controller in CPU
  - Memory errors integrated with machine checks
  - Handled by standard MCE subsystem
- EDAC needs driver for each platform
  - And often accesses “non stable” registers that could change even with steppings
  - Mcelog uses standardized interfaces
- No integration with software
  - Requires special configuration for each board to identify components
- Cannot do a lot of things that user space (mcelog) can do

Testing

- Testing machine checks is difficult
  - Normal operation doesn't have enough errors
  - So standard Linux community testing model doesn't work very
  - Needs special injection support and test suites
- Various injectors on software level (without hardware support)
  - Low level machine check injector
  - Page error injector for process and for arbitrary page
- New injectors, test suite mce-test for testing MCEs
  - Testing low level handler with mce-inject
  - Testing hwpoison VM code in process context
    - Bring pages into specific states and test to see if they can be poisoned
- Ongoing work to get the best test coverage

Legal Information

intel®

Leap ahead™