# mcelog: memory error handling in user space

Andi Kleen
Intel®Corporation
andi@firstfloor.org

## 1 Abstract

Memory error handling is an increasingly important topic. The paper describes memory error handling in user space on Linux systems using the mcelog daemon. It describes features like bad page offlining or cache error handling that improve the uptime of server systems.

## 2 Introduction

Servers and high-performance computing systems contain more and more memory to handle bigger data sets. But with more and larger memory modules, and more transistors in them, combined with larger clusters of systems, the rate of memory errors in operation is also increasing.

Modern server systems generally use ECC memory and other ways to detect and correct many memory errors in the hardware. When the hardware corrects an error it generates corrected error events. These events can be also used by specialized software to prevent future failures.

mcelog is a daemon for handling and reporting hardware errors. It is able to use trends in corrected error reports to implement specific error prevention algorithms.

## 3 Basic error architecture

A modern x86 server CPU with an integrated memory controller memory and ECC memory constantly checks for memory data errors and corrects and reports them. For this it uses techniques like patrol scrubbing and demand scrubbing and also computes the checksum of all data read from memory.[8]. Errors will be usually reported to the operating system using the standard x86 machine check architecture (**MCA**) registers. Uncorrected memory errors – that is data corruption – are reported using a machine check exception and handled directly by the kernel, for example by killing the affected process or shutting down the system down.

Corrected errors are polled by a timer or interrupt handler[1] and reported to the mcelog daemon. The mcelog daemon decodes the error and does accounting and other book keeping. The rest of the paper will discuss the various actions implemented in the mcelog daemon.

Traditionally mcelog was run as a cronjob, but in newer distributions it supports a daemon mode started by an init script. It is recommended to ensure the daemon is always running on every system with ECC memory. [2]

More information on the basic architecture can be found in [1] and the hardware architecture is described in [3].

[1] Older system check with a regular timer, newer Intel systems have support for a special **CMCI** interrupt for faster notification.

[2] At least one popular distribution ships the init script, but does not enable mcelog. It is recommended to enable it manually, otherwise no memory errors will be logged.

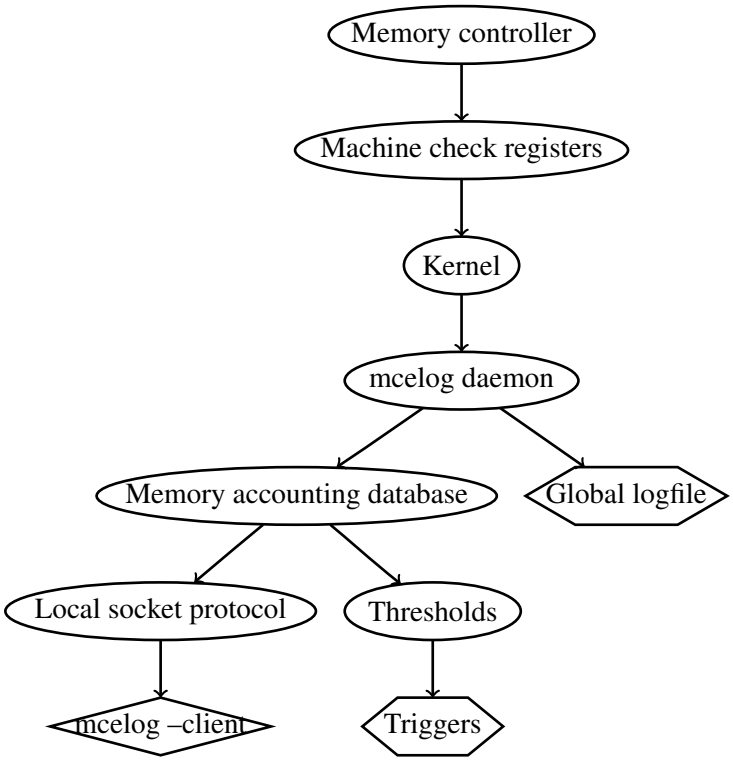


Figure 1: mcelog basic error flow

### 3.1 EDAC

Linux also has an alternative memory error reporting infrastructure called [9]. It consists of separate drivers for specific platforms that use hardware facilities to do memory error counting and DIMM topology discovery. While it supports basic memory error counting and some logging, it does not implement any of the advanced features described here. EDAC is not covered in this paper.

## 4 Soft errors versus hard errors

When a corrected error – or soft error – occurs in a system this is not necessarily a problem. In fact on systems with long uptime there is an expected soft error rate that will be reported. The hardware platform uses error correcting codes and redundancy to handle soft errors. This is why they are called corrected errors. Unlike an uncorrected (hard) error – that is data corruption – soft errors do not directly require software reaction. Also since there is an expected soft error rate for each system some soft errors are expected to occur. A small number of soft errors in a given time frame is generally not a problem.

The actual soft error rate depends on many factors, including quality of memory, size of memory (the more transistors in memory the more errors), quality of the motherboard and other factors. Memory soft errors can also occur in interconnections (like PCI-Express or QPI) and in CPU caches and other hardware components. Above all on clusters of multiple systems the total error rate of the cluster is of course rising, too. The more components a cluster contains the more – but usually corrected – errors.

The goal of correct error trend analysis is thus not to react to the first reported error, but look at trends, and when the error rates exceed specific thresholds to warn the administrator or take other corrective action. These error thresholds are never absolute numbers[3], but an error threshold defined for a specific time period.

For more details on error rates on realistic systems see [2] and [4].

[3] Any absolute number could be exceeded by a system with long enough uptime.

# 5 Logging

The simplest action in mcelog is to log the error to disk or syslog. The error will be decoded into an ASCII representation containing all information reported by the hardware, like the error address or the type of the error. The default log file is */var/log/mcelog*.

mcelog will also attempt to determine the DIMM associated with the error if possible. This requires CPU specific support in mcelog. It will also attempt to determine the motherboard label and part number associated with the DIMM. This requires correct SMBIOS tables in the BIOS.

In some use cases – like combined error reporting for a large cluster– full soft error logging can generate a large amount of information. An alternative is to turn off logging and only rely on threshold counters. This is supported with the *filter-memory-errors* option in *mcelog.conf*.

# 6 Error accounting

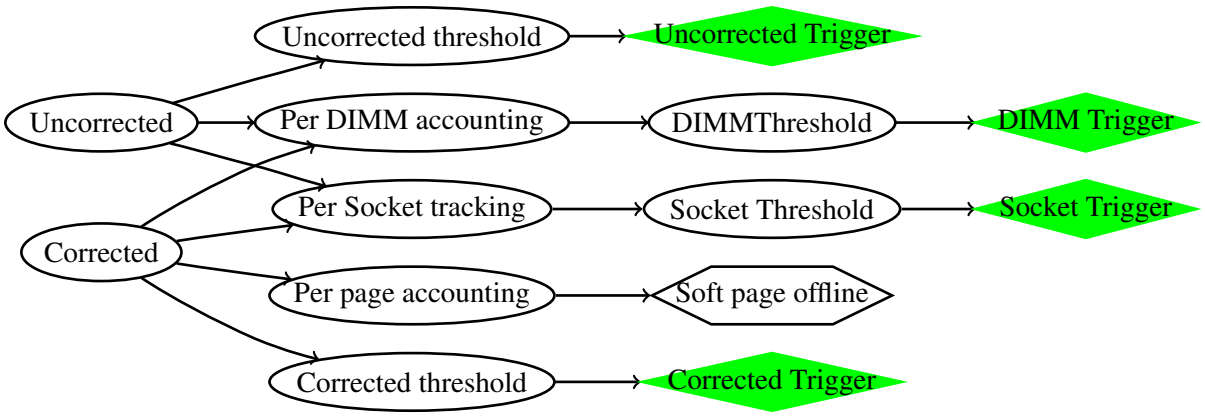


Figure 2: mcelog thresholds

mcelog accounts all memory errors reported. The error accounting can be per DIMM[4], per page and per socket. In some cases when the DIMM cannot be determined it will fall back to accounting per DDR channel or per CPU socket. The error accounting information will be stored in a in memory error database.

In addition to a global counter mcelog also maintains thresholds using a *leaky-bucket* algorithm. When the number of errors in a specific time window exceeds a preconfigured threshold a *trigger* will be executed. Triggers usually are shell scripts in */etc/mcelog*, but can be also other internal actions. Thresholds and triggers can be configured in *mcelog.conf*

## 6.1 mcelog client

The error database of a running daemon can be queried with *mcelog –client*. The client will query the daemon using a local unix socket and print the current global error counts and threshold states.

Listing 1: mcelog –client output example

```
# mcelog --client
Memory errors
SOCKET 0 CHANNEL 2 DIMM 1
corrected memory errors:
        3 total
        3 in 24h
uncorrected memory errors:
```

[4]If the CPU or BIOS supports determining the DIMM

```
        0 total
        0 in 24h

SOCKET 0 CHANNEL 0 DIMM 0
corrected memory errors:
        3 total
        3 in 24h
uncorrected memory errors:
        0 total
        0 in 24h
```

### 6.2 mcelog triggers

When a error threshold is exceeded mcelog calls a *trigger*. A trigger is normally a simple shell script, located in */etc/mcelog*, that is called with a set of predefined environment variables. The administrator can put own actions in there, like notifying the administrator or logging to a central cluster server. By default any trigger will log a summary message into the system log. It is recommended to not modify the original scripts, but put any site specific actions into a .local subscript that is called by the original trigger.

Listing 2: Environment passed tp the DIMM error trigger

```
# THRESHOLD       human readable threshold status
# MESSAGE         Human readable consolidated error message
# TOTALCOUNT      total count of errors for current DIMM of CE/UC depending on
#                 what triggered the event
# LOCATION        Consolidated location as a single string
# DMI_LOCATION    DIMM location from DMI/SMBIOS if available
# DMI_NAME        DIMM identifier from DMI/SMBIOS if available
# DIMM            DIMM number reported by hardware
# CHANNEL         Channel number reported by hardware
# SOCKETID        Socket ID of CPU that includes the memory controller with
                  the DIMM
# CECOUNT         Total corrected error count for DIMM
# UCCOUNT         Total uncorrected error count for DIMM
# LASTEVENT       Time stamp of event that triggered threshold (in time_t
                  format, seconds)
# THRESHOLD_COUNT Total number of events in current threshold time period
                    of specific type
```

## 7 Bad page offlining

One reasonably common class of memory errors is a single "stuck bit" in the DIMM[4], [2]. The bit stays stuck in a specific state and will result in future read errors. Other bits in the same DIMM or on the same channel are not affected. With ECC DIMMs this error can be corrected: it is not immediately a fatal problem. But when another nearby bit gets corrupted for some reason this could develop into an uncorrected 2-bit error. In addition the stuck bit will generate regular continuous corrected error reports when the memory scrubber scrubs it again. Handling these reports takes some time and may drown error thresholds for other purposes. It does not actually tell anything new.

The best strategy is to simply stop using the bit. The only entity which has reasonable fine control over that is the operating system. The OS manages memory by pages (typically $4K$ of size) and it's possible to offline the page containing the stuck bit and stop using it. Linux has had a page migration capability for some time – originally developed for NUMA tuning purposes – that was easily repurposeable for offlining. Similar strategies have been used by other operating systems for some time[5].

When running in daemon mode mcelog keeps track of corrected memory errors per 4K pages and maintains error counters for each page. This is controlled using the *[page]* section in mcelog.conf mcelog defaults to page tracking enabled by default (if the CPU supports it) with offlining of a specific page when a threshold of 10 errors per 24 hours is crossed. This threshold is a reasonable conservative threshold for today's **DDR3** memory subsystems.

Maintaining these statistics has a memory overhead (64 bytes including metadata per 4k page on a 64bit system, that is roughly 1.5% of memory). Since mcelog is running in user space this memory can be swapped out when needed. On typical systems the number of errored pages is small and memory usage is not a problem. Mcelog has support for limiting the maximum memory

Starting with kernel versions *2.6.33* (and in some *2.6.32* kernels with backports) Linux supports a page soft-offlining capability. That is the contents of the page are copied somewhere else (or dropped if not needed) and the original page is removed from the normal operating system memory management and not used anymore.

The capability is called soft-offlining because it never kills or otherwise affects any application, in contrast to the "hard-offlining" that is done when an uncorrected recoverable data error happens. The memory is just copied to a new page and transparently remapped.

One caveat is that offlining does not work for all pages, but only for pages in the Linux page cache (that is containing application memory or file data) and free pages. Hugepages are also currently not supported[5] With common workloads (if they are not hugepages heavy) the majority of memory can be usually soft-offlined. There is ongoing work to improve the soft offliner for more page types.

The *page-types* tool included with the kernel source can be used to display page types. This is the current state of the author's work station. All the page types in the top ten, mostly consisting of *LRU* pages in the page cache and free pages (*nopage*), can be offlined if they should develop a stuck bit.

```
# page-types  | awk '{ printf "%8s %8s %15s %30s\n",$2,$3,$4,$5 }' |
  sort -rn +1  | head
 1310720     5120                                        # total
  885248     3458  ____________________________________
  262144     1024  ____________________n_______________                  nopage
   36400      142  __RU_l_____________________________      referenced,uptodate,lru
   31530      123  ___U_l_____________________________              uptodate,lru
   28024      109  ___U_lA____________________________          uptodate,lru,active
   23955       93  ___U_lA____Ma_b____________________ uptodate,lru,active,mmap,anonymous,
                                                       swapbacked
   17105       66  __RU_lA____________________________ referenced,uptodate,lru,active
    7787       30  __R__l_____________________________            referenced,lru
    5534       21  _______S___________________________                    slab
```

Bad page offlining works on CPUs that provide a physical address on corrected memory machine check errors. These are usually CPUs with integrated memory controller and ECC memory support. On some CPUs the address can only be retrieved with help from the BIOS through an ACPI **GHES**[7] driver. The BIOS has to be configured to "firmware first" mode. The **GHES** driver is available since Linux 2.6.34 or in backports as loadable modules for 2.6.32 based distribution kernels[6].

Bad page offlining state is currently only kept in memory and not saved over reboots. This greatly simplifies the implementation, otherwise the operating system would need to detect when DIMMs have changed. The drawback is that the page with the stuck bit will be used for some time again after the next reboot unless enough corrected errors have accumulated to offline it again.

# 8 Cache error handling

Modern x86 CPUs consist mostly of cache memory. Checking the health of the cache is a good way to check the health of the CPU.

[5] hugepage soft-offline support is in development and scheduled for the Linux 2.6.37 kernel.

Recent Intel®CPUs have support for enhanced cache error reporting to report on the health of the CPU caches for corrected machine checks. To quote the Intel Architecture Software Developer's Manual (Volume 3, 15.4)[3]

> A processor that supports enhanced cache error reporting contains hard- ware that tracks the operating status of certain caches and provides an indi- cator of their "health". The hardware reports a "green" status when the number of lines that incur repeated corrections is at or below a pre-defined threshold, and a "yellow" status when the number of affected lines exceeds the threshold. Yellow status means that the cache reporting the event is operating correctly, but you should schedule the system for servicing within a few weeks.

When mcelog sees a "yellow" state on a cache it attempts to offline the cores owning that cache. This will prevent the cache from being used further. This allows the system to continue operating, but with reduced computing capability.

This feature can be configured using the options in the *[cache]* section of *mcelog.conf*. When a cache error occurs the trigger script configured with *cache-threshold-trigger* is executed. By default mcelog calls the */etc/mcelog/cache-error-trigger* script which offlines the affected CPUs through the CPU offline interface of sysfs and logs a message to the system log. Stopping scheduling processes on these CPUs will stop using the caches owned by these CPUs. Thus the system can continue operating with reduced computing capacity. There is currently a limitation that the first CPU core cannot be offlined.

## 9 Deployment

The earlier non-daemon versions of mcelog have been deployed in major 64bit Linux distributions for many years as a simple cron job. Since the Linux 2.6.32 kernel it also started supporting 32bit distributions. The new versions with daemon and error database functionality are starting to get deployed now, but are already shipping in some recent distributions. The eventual goal is to have mcelog running by default for error handling on all Linux servers.

## 10 Conclusion

Handling memory errors well is important for server reliability. As shown mcelog can improve the reliability of a system by doing predictive failure analysis on corrected memory errors. This can result both in alerting the system administrator before a system fails and in some automatic actions that improve system reliability.

## References


[1] "Machine check handling on Linux", Kleen, Linux Kongress 2004 http://halobates.de/mce.pdf

[2] "DRAM Errors in the Wild: A Large-Scale Field Study", Schroeder, Pinheiro, Weber, SIGMETRICS, 2009 http://research.google.com/pubs/archive/35162.pdf

[3] "Intel®64 and IA-32 Architectures Software Developer's Manual Volume 3A: System Programming Guide Part 1", Intel Corporation, 2010 http://www.intel.com/products/processor/manuals/

[4] "A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility", Li, Huang, Shen, Chu: Usenix Annual Tech Conference 2010 http://www.cs.rochester.edu/~kshen/papers/usenix2010-li.pdf

[5] "Assessment of the Effect of Memory Page Retirement on Systems RAS against Hardware Faults", Tang, Arruthers, Totari, Shapiro: Proceedings of the 2006 International Conference on Dependable Systems and Networks.

[6] ftp://ftp.kernel.org/pub/linux/kernel/people/ak/apei/

[7] "Advanced configuration and power interface specification 4.0a" http://www.acpi.info/

[8] "Advanced Reliability for Intel®Xeon®Processor-based servers", Kumar, Demshki, Shiveley: http://software.intel.com/sites/oss/pdfs/Intel_Xeon_Processor_7500_Series_RAS.pdf

[9] Linux EDAC project on sourceforge. http://bluesmoke.sourceforge.net/