Lenovo

Demonstrating the Memory RAS Features of Lenovo ThinkSystem Servers

Explains the memory RAS features of the Lenovo ThinkSystem servers

Shows how to enable the related features in UEFI

Provides the Linux kernel commands to set and check the RAS features

Shows the effect of MCA recovery, address range mirroing, and PFA

Neo Cui

LENOVO PRESS

Abstract

Reliability, availability and serviceability (RAS) is a computer hardware engineering term referring to the elimination of hardware failures to ensure maximum system uptime. The memory RAS features in Lenovo® ThinkSystem™ servers include Error Correcting Code (ECC), spare memory banks, page retirement and mirroring.

This document describes the memory RAS features in detail, explaining how to server availability is enhanced with the memory RAS features on Lenovo ThinkSystem servers running Linux.

At Lenovo Press, we bring together experts to produce technical publications around topics of importance to you, providing information and best practices for using Lenovo products and solutions to solve IT challenges.

See a list of our most recent publications at the Lenovo Press website:

http://lenovopress.com

Do you have the latest version? We update our papers from time to time, so check whether you have the latest version of this document by clicking the **Check for Updates** button on the front page of the PDF. Pressing this button will take you to a web page that will tell you if you are reading the latest version of the document and give you a link to the latest if needed. While you're there, you can also sign up to get notified via email whenever we make an update.

Contents

Introduction

The machine-check mechanism in Lenovo ThinkSystem servers allows the processor to detect and report a variety of hardware errors based on Machine Check Architecture (MCA) and Machine Check Exception (MCE). The hardware errors are classified by MCE. MCA is an Intel mechanism in which the CPU reports MCEs to the operating system (OS). The OS has a special handler to process the information contained in the MCA registers.

There are two major types of MCEs: a notice or warning error and a fatal exception. A warning will be logged by a “Machine Check Event logged” notice in the system logs, and can be viewed later using certain Linux utilities. A fatal MCE will cause the machine to stop responding and the details of the MCE will be printed out to the system’s console.

The most common errors in MCE events are:

- Memory errors or Error Correction Code (ECC) problems
- Inadequate cooling/processor overheating
- System bus errors
- Cache errors in the processor or hardware

The errors are classified into several MCE types, as shown in Table 1.

Table 1 MCE Types

Type of MCE	Description
Corrected Error (CE)	An error corrected by hardware
Uncorrected Error (UC)	Hardware could not correct the error. The processor context is corrupted and cannot continue to operate the system.
Uncorrected Recoverable Error (UCR):	
Software Recoverable Action Required (SRAR)	The error is detected and the processor already consumes the memory. System reboot is recommended.
Software Recoverable Action Optional (SRAO)	Some data in the memory are corrupted. But the data have not been consumed and system can perform a recovery action.
Uncorrected No Action Required (UCNA)	Some data in the memory are corrupted, but the data has not been consumed and the system may continue to operate.

Memory RAS features

This section introduces the main RAS features that ThinkSystem servers have.

MCA Recovery

The new Intel Xeon Scalable Family processors support recovery from some memory errors based on the Machine Check Architecture (MCA) Recovery mechanism. This requires the OS to declare a memory page “poisoned”, kill the processes associated with the page and avoid using the page in the future.

The MCA mechanism is used to detect, signal, and record machine fault information. Some of these faults are correctable, whereas others are uncorrectable. The MCA mechanism is intended to assist CPU designers and CPU debuggers in diagnosing, isolating, and

understanding processor failures. It is also intended to help system administrators detect transient and age-related failures, suffered during long-term operation of the server.

The MCA Recovery feature is a part of the fault tolerant capabilities of servers based on the Intel Xeon Scalable Family processors, such as the ThinkSystem portfolio of servers. These capabilities allow systems to continue to operate when an uncorrected error is detected in the system. If not for these capabilities, the system would crash and might require hardware replacement or a system reboot.

MCA Recovery handles the following errors:

- Software Recoverable Action Required (SRAR): There are two types of such errors – detected by Data Cache Unit (DCU) and detected by Instruction Fetch Unit (IFU).
- Software Recoverable Action Optional (SRAO): There are two types of such errors - detected by memory patrol scrub and detected by Last Level Cache (LLC) explicit writeback transaction.

Figure 1 shows the system error handling flow with a Linux operating system.

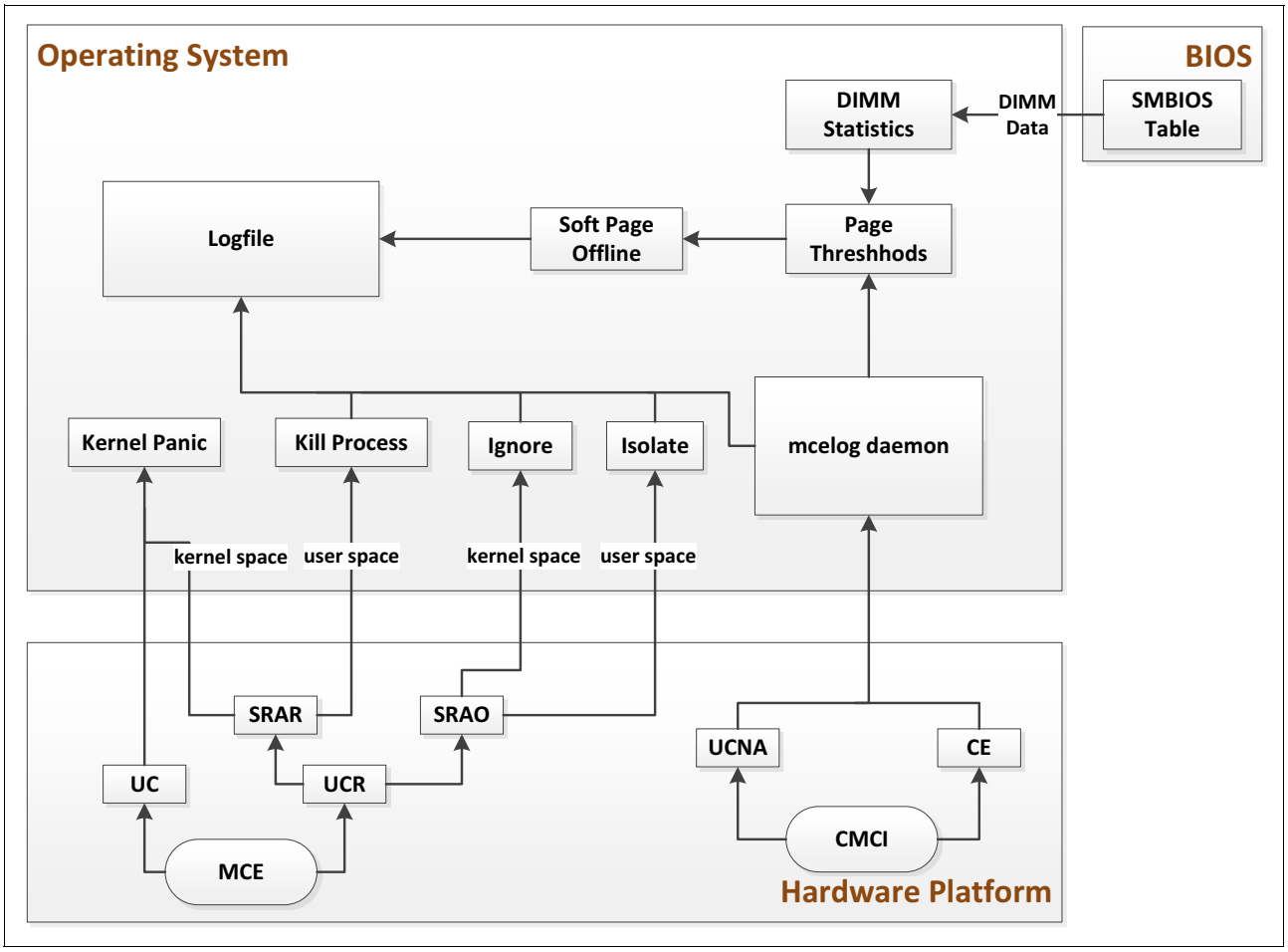

Figure 1 Linux Operating System Error Handling Flow

Primarily, hardware faults are reported to the OS using either Machine-check exception (MCE) or Corrected Machine Check Interrupt (CMCI). There are also other mechanisms that report error events, such as System Control Interrupt (SCI). The MCA Recovery feature implementation uses MCE to notify the OS when an SRAR or SRAO event is detected by the

hardware. Then, the OS analyses the log to verify if recovery is feasible. It then handles the affected memory page (default page size is 4KB) and logs the event in the mcelog.

In the case of an SRAO event, the OS recovers and resumes normal operation. In the case of an SRAR-IFU event, the OS reloads the 4KB page containing the instruction to a new physical page and resumes normal operation. In the case of an SRAR-DCU event, the OS triggers a “SIGBUS” event to notify the application for further recovery action. The application has a choice to either reload the data and resume normal execution, or kill the application to avoid crashing the entire system.

Memory Address Range Mirroring

Address Range Mirroring is a new memory RAS feature on the Intel Xeon Scalable Family platform that allows greater granularity in choosing how much memory is dedicated for redundancy.

Memory mirroring implementations (full mirror mode, partial mirror mode, and address range mode) are designed to allow mirroring of critical memory regions to increase the stability of physical memory. Dynamic (without reboot) failover to the mirrored memory is transparent to the OS and applications.

An illustration of Address Range Mirroring is shown in Figure 2. It is similar to partial memory mirroring and can be enabled selectively for individual physical machines. On each physical machine on which Address Range Mirroring is enabled, the size (range) of the primary and secondary mirrors can be defined using 64MB intervals.

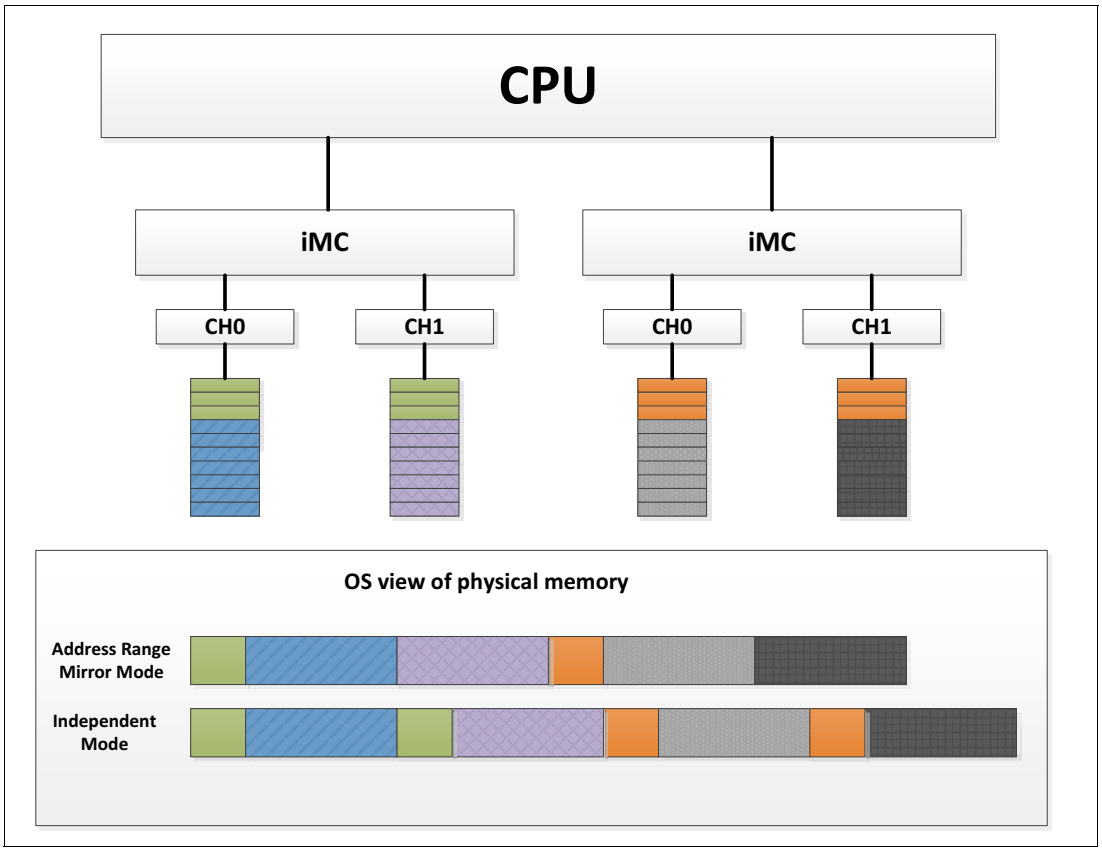

Figure 2 Address Range Mirroring

The Intel Xeon processor with SKU level upper than Sliver supports up to two mirror ranges, one mirror range per integrated Memory Controller (iMC). The range is defined by the value programmed in the Target Address Decoder 0 (TAD0) register for the server. The TAD0 defines the size of the primary and secondary mirror ranges. The secondary mirror range is reserved for redundancy and not reported in the total memory size. To enable Address Range Mirroring, there is a Control and Status Register (CSR) bit that enables TAD0 use for mirroring.

Address Range Mirroring offers the following benefits:

- Provides further granularity to memory mirroring by allowing the firmware or OS to determine a range of memory addresses to be mirrored, leaving the rest of the memory in the socket in non-mirror mode.
- Reduces the amount of memory reserved for redundancy.
- Improves high availability, avoiding uncorrectable errors in the kernel memory of the Linux system by allocating all kernel memory from the mirrored memory.

Address Range Mirroring has the following OS and firmware requirements:

- Requires OS support to fully utilize Address Range Mirror. The OS must be aware of mirrored region.
- Requires a firmware-OS interface. The UEFI firmware on Lenovo ThinkSystem servers has implements the following interfaces:
  - UEFI Variables -- A method to request the amount of mirrored memory
  - UEFI Memory map -- Presents mirrored memory range on the platform

Memory Read/Write strategy

Address Range Mirroring improves the read/write efficiency using an effective channel interleave method, illustrated in Figure 3 on page 7. In the figure, **N+*n*** represents the data in the memory. The Memory Reads are interleaved between all four channels for the mirrored and non-mirrored areas. Memory Writes are interleaved between two channels for the mirrored areas, and interleaved between four channels for non-mirrored areas.

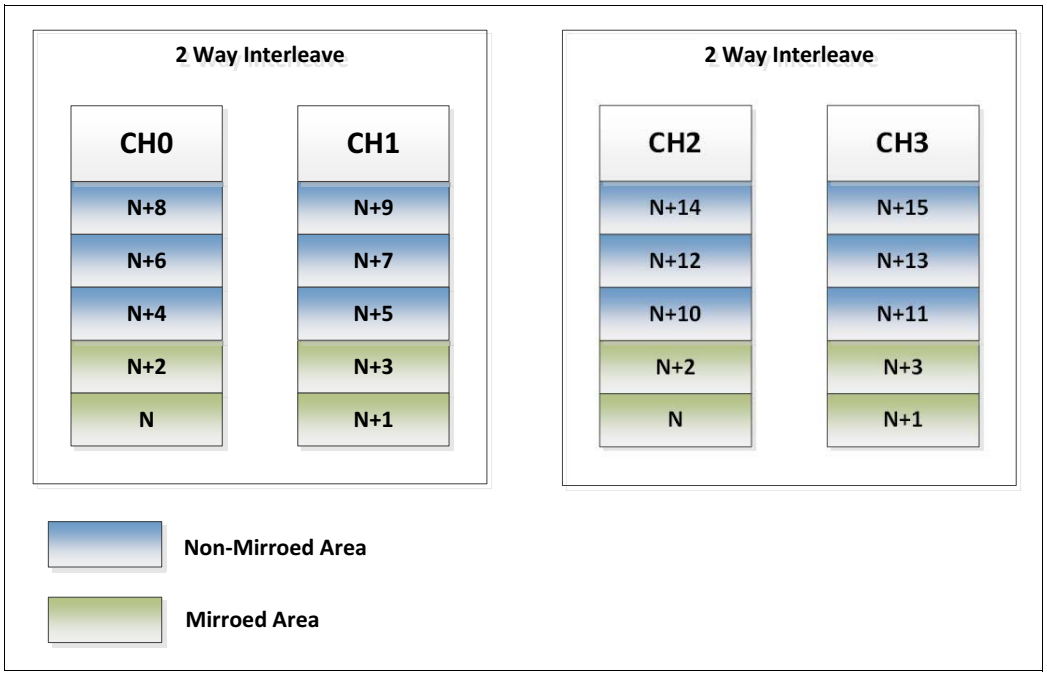

Figure 3 Interleave in Address Range Mirroring

Memory Error Recovery

The Address Range Mirroring improves the memory fault tolerance and error correction capabilities of the system. The uncorrected errors in the mirrored memory region can be downgraded to corrected errors to avoid system corruption. The error recovery workflow is illustrated in Figure 4 on page 8.

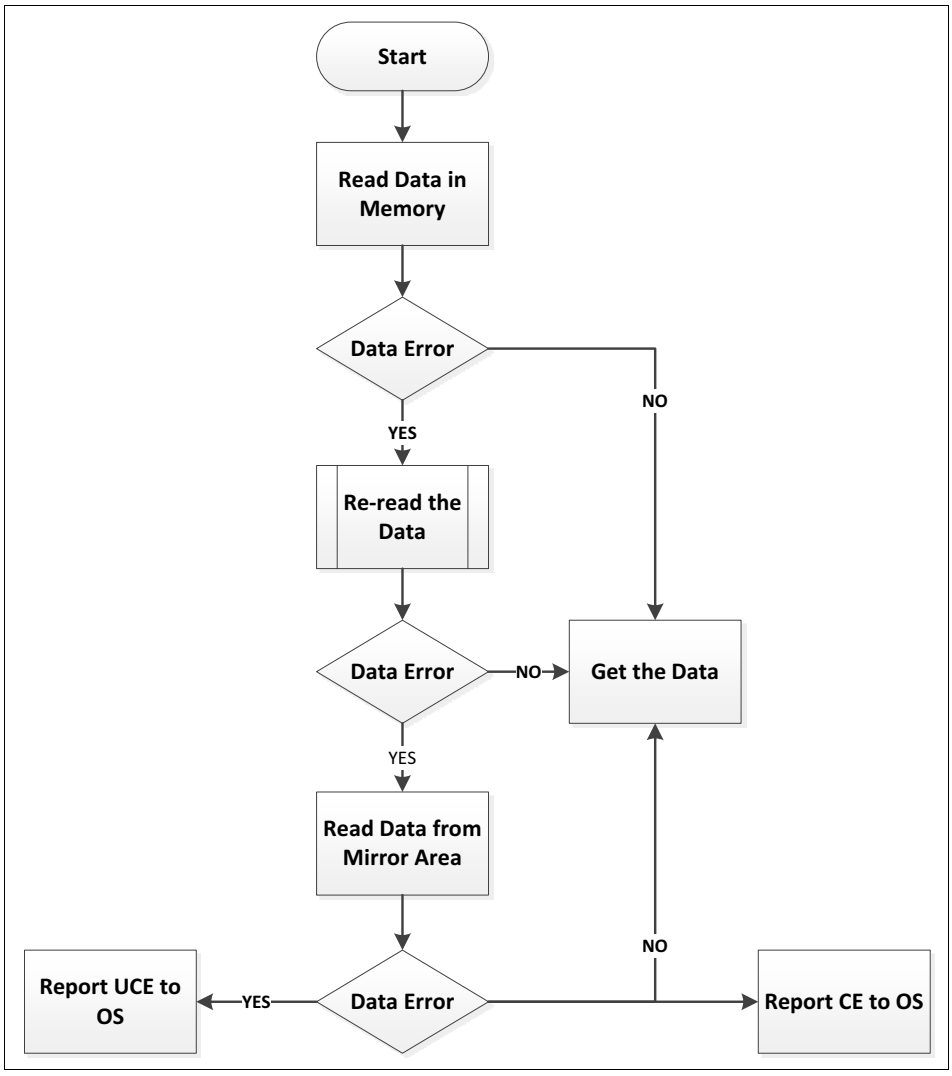

Figure 4 Memory Error Recover Workflow

Linux support for Address Range Mirroring

There are two ways to manage physical memory in Linux: memblock and Zone Allocator[1].

Memblock manages memory blocks during the early bootstrap period, but is discarded after initialization and this function is taken over by Zone Allocator. Every memory block consists of two arrays – memblock.memory and memblock.reserved.

As illustrated in Figure 5 on page 9, a memory block is marked as “reserved” if it has been allocated or used. Memory mirror support in memblock has been merged into Linux kernel version 4.3.

[1] Taku Izumi. Linux Conference 2016: Address Range Memory Mirroring

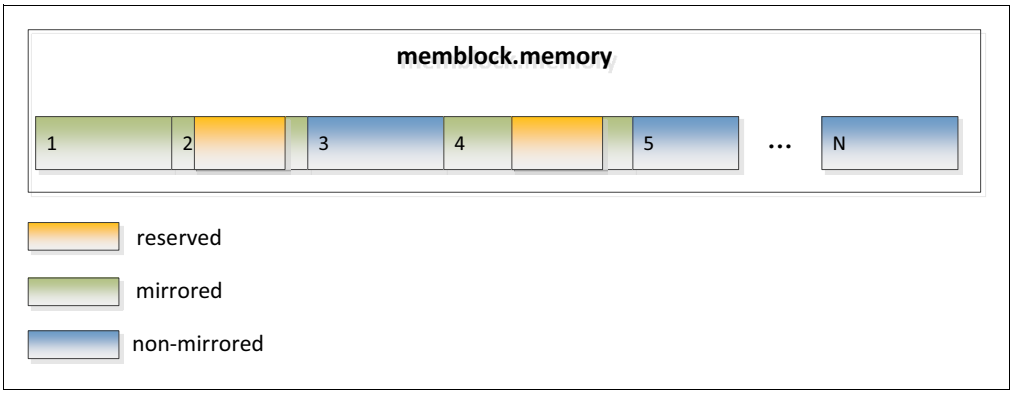

Figure 5 Memory Mirror Support in the Memblock

The Zone Allocator is the usual memory management method in Linux kernel. The total system memory is partitioned in different zone types in this model. The Zone Allocator manages these memory zones. There are three types of zones on an x86_64 architecture server, which are ZONE_DMA, ZONE_DMA32, and ZONE_NORMAL.

The Zone Allocator is a suitable method for implementing Address Range Mirroring on a Linux system. Kernel version 4.6 and later already support Address Range Mirroring. The implementation of this feature based on Zone Allocator is shown in Figure 6.

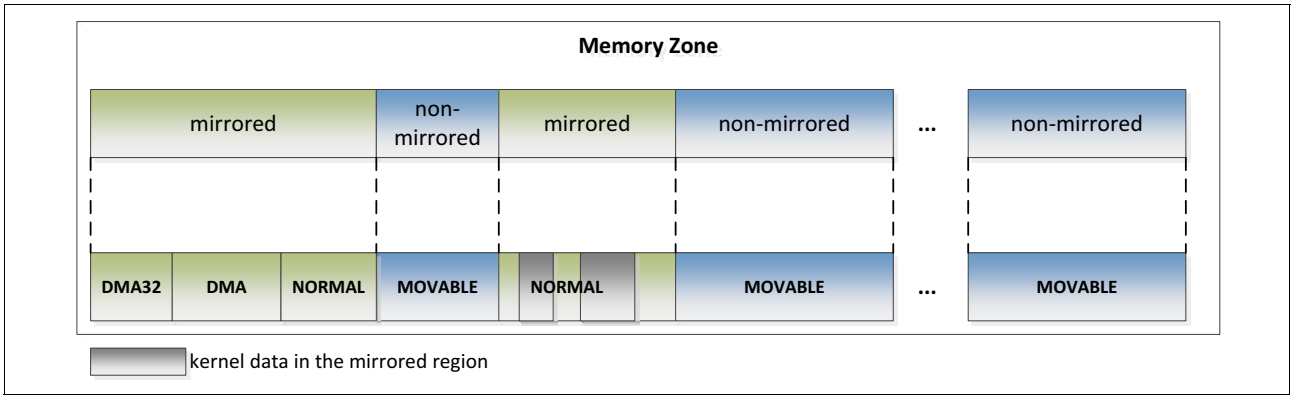

Figure 6 Address Range Memory Mirroring in Linux Kernel

If you want to allocate kernel memory requested from the mirrored region, you need to specify `kernelcore=mirror` in the kernel boot parameter. As demonstrated in Figure 6, the non-mirrored region will be allocated to ZONE_MOVABLE and memory used by the kernel will only be allocated from the mirrored region.

Memory Predictive Failure Analysis

One of the most important service that an OS provides to applications is the management of memory. The OS allows processes to allocate memory as pages. These pages can be of varying sizes (typically 4KB) depending on the hardware's capabilities, and are be backed by a combination of main memory or disk space. The actual content of the pages may be copied in multiple places, such as the processor cache, main memory, swap space, or in a file.

If a correctable fault occurs in the memory, we don't need to perform any recovery action on the OS. The platform hardware uses Error Correcting Code (ECC) and redundancy to handle correctable errors. However, if we continue to see correctable faults, then perhaps the memory is failing. To avoid the possibility of future uncorrectable faults in the same page, we

can copy the data to a different page and mark the page as offline (retired). This is the mechanism used by Memory Predictive Failure Analysis (PFA).

Introduction to PFA

The PFA technique itself is quite simple: if a physical memory page is believed to be affected by an underlying hardware fault (e.g., a weak cell or faulty row in a memory chip or DRAM), the affected page can be retired by relocating its content to another physical page, and placing the retired page on a list of physical pages that should not be subsequently allocated by the virtual memory system.

If the underlying fault manifests itself as one or more Corrected Error (CE) to the OS, the page retirement can be completed immediately by copying its content to another physical page and updating the virtual memory page translation tables. If the underlying fault manifests as an Uncorrected Error (UE) on a clean page, the page can be retired and the OS can subsequently allocate a new physical page and re-read the contents of the page from the associated backing object.

If a UE affects a dirty page and the UE is detected upon cache writeback, the OS marks the page as having a UE but defers action until the page is subsequently accessed (hoping the page will instead be freed). Finally, if a UE affects a dirty page and the error is detected upon access, the OS forcibly terminates the affected process, retires the affected page, and then restarts the affected service. Therefore, only pages that are not relocatable at all, such as those used within particular regions of the kernel itself, cannot be retired.

Linux support for PFA

Linux kernel version 2.6.33 (and some 2.6.32 kernels with backports) implements PFA based on mcelog and page soft-offline. 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. This capability is called soft-offlining because it never kills or otherwise affects any application, in contrast to the hard-offlining that is performed when an uncorrected recoverable data error occurs.

Mcelog records and counts MCEs. Mcelog is required by the Linux kernel to record MCEs and should run on all Linux systems that require error handling. When the number of errors in a specific time window (usually 24 hours) exceeds a pre-configured threshold, a trigger will be executed. Triggers are usually shell scripts in the /etc/mcelog directory, but can also be other internal actions. Thresholds and triggers can be configured in mcelog.conf.

For more information about mcelog, see the following website:

http://www.mcelog.org/

Figure 7 on page 11 demonstrates the mcelog mechanism for different error types in the memory:

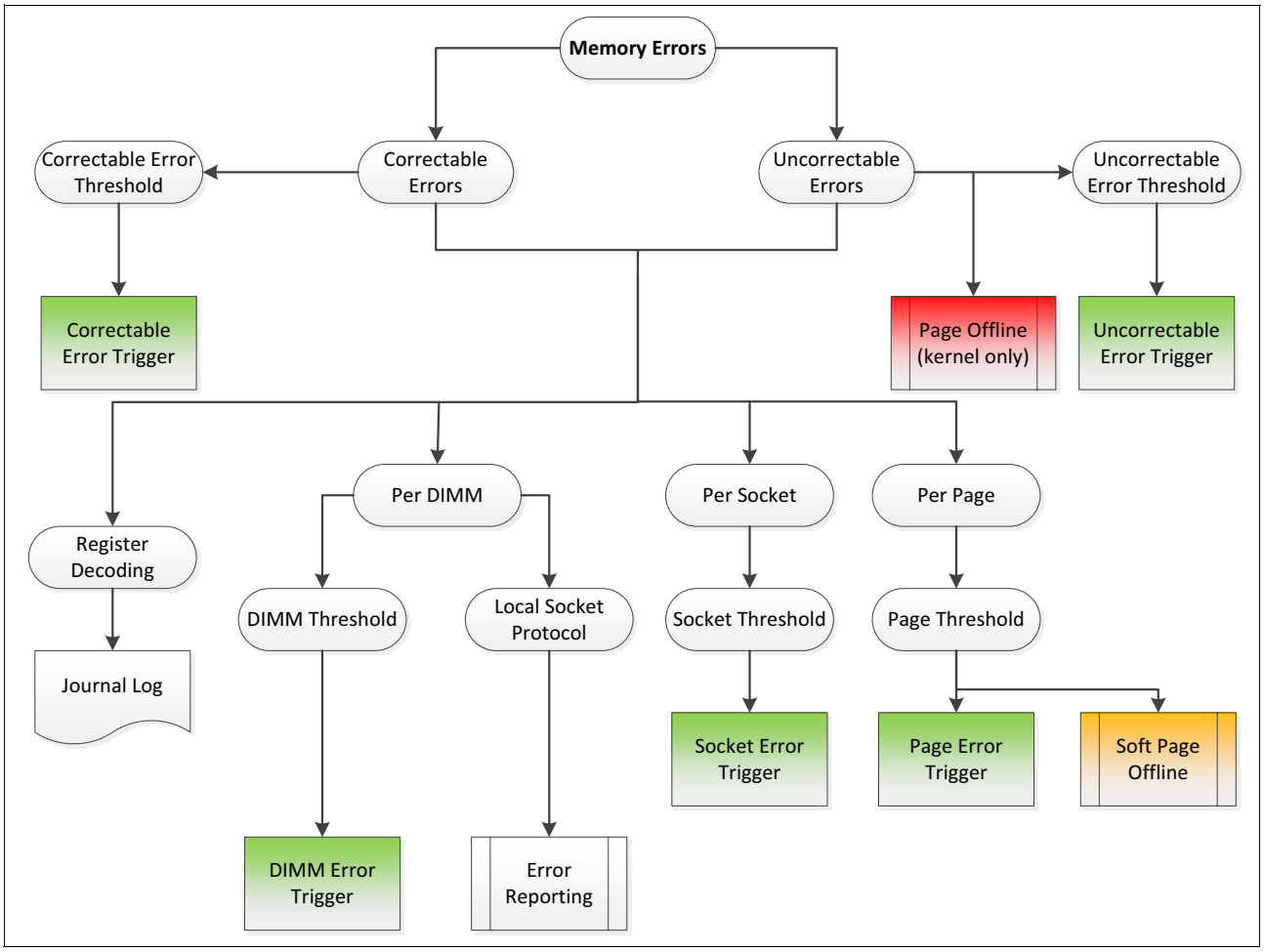

Figure 7 Mcelog Mechanism

The command **`mcelog --client`** can be used to query a running daemon to perform error reporting. The daemon can also execute triggers when configured error thresholds are exceeded. This is used to implement a range of automatic PFA algorithms, including bad page offlining and automatic cache error handling. User-defined actions can also be configured. All errors are logged to /var/log/mcelog or the system log or system journal.

Demonstrating memory RAS features on ThinkSystem servers

For our demonstration of the memory RAS features, we will be using the ThinkSystem SR650 server running Red Hat Enterprise Linux 7.3. The server uses the Intel Xeon Gold 6150 Processor with eight 16GB 1Rx4 TruDDR4™ DIMMs.

The demonstration tool we are using is EINJ which provides a hardware error injection mechanism. It is very useful for debugging and testing APEI and RAS features in general. In this demo, we will use EINJ to trigger and validate the RAS feature.

For details about the EINJ tool, see the following EINJ documentation:

https://www.kernel.org/doc/Documentation/acpi/apei/einj.txt

MCA demonstration

Before we can demonstrate how MCA is implemented in Linux with the use of the EINJ error injection tool, we need to enable Machine Check Recovery in UEFI.

Setting up UEFI

The steps to setup UEFI to enable Machine Check Recovery are as follows:

1. Power on the server and press F1 to enter ThinkSystem UEFI setup menu, XClarity Provisioning Manager.
2. From the left navigation mention, click **System Settings** → **Recovery and RAS** as shown in Figure 8.

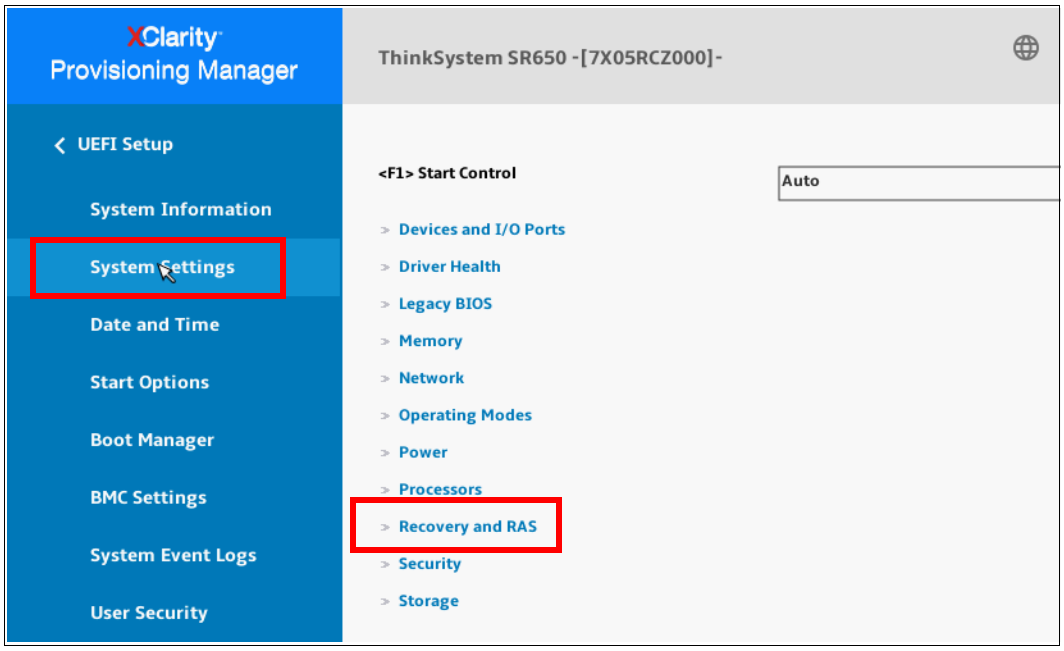

Figure 8 System Settings

3. Select Advanced RAS as shown in Figure 9 on page 13.

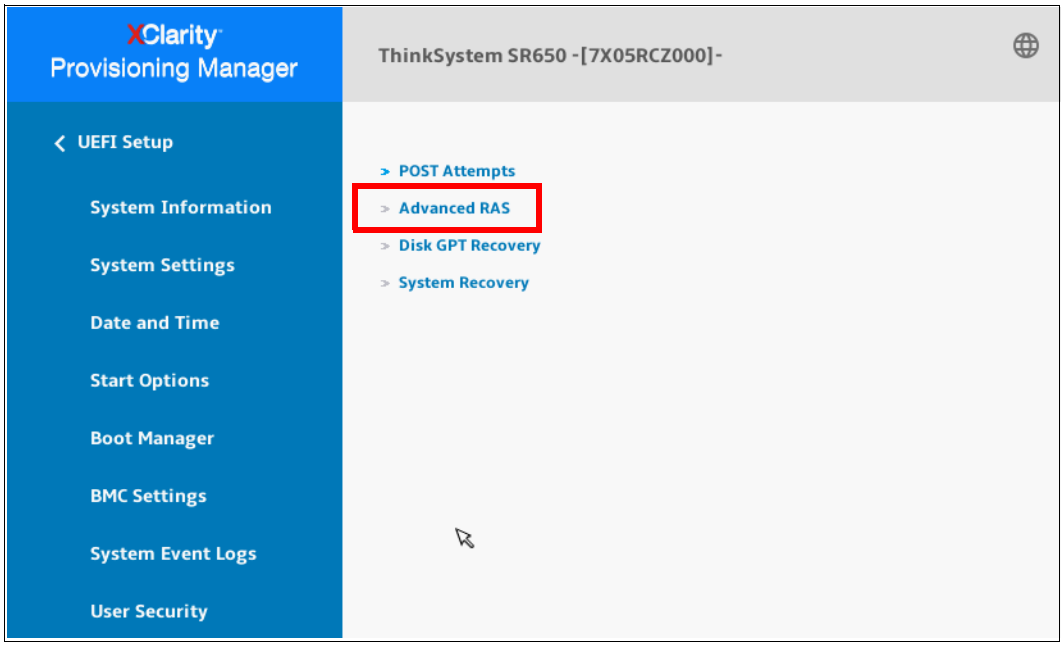

Figure 9 Recovery and RAS menu

4. Enable **Machine Check Recovery** as shown in Figure 10

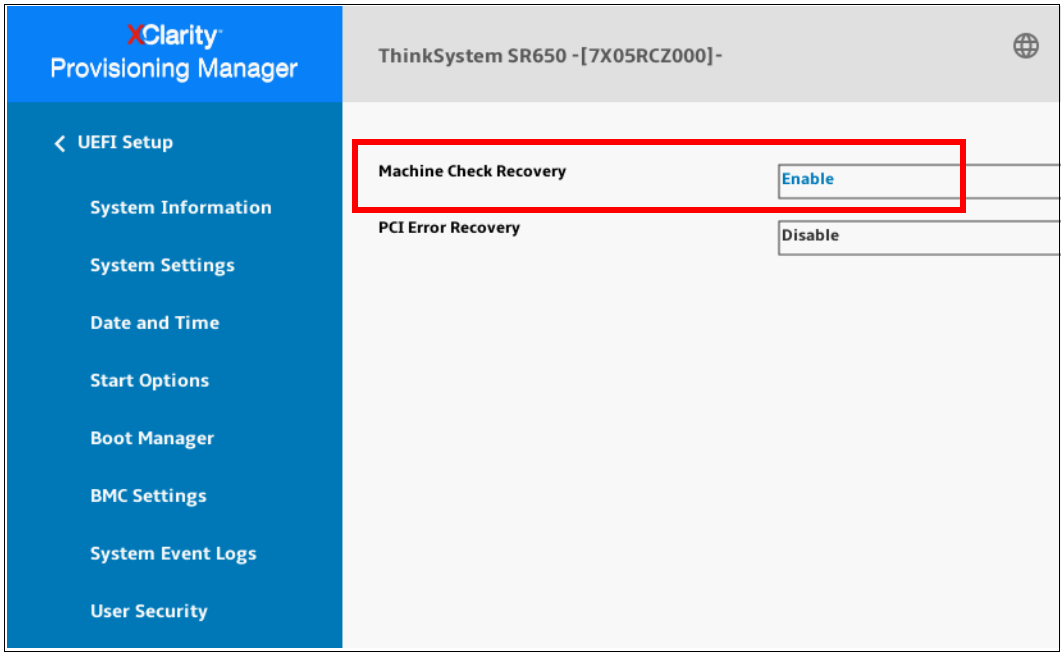

Figure 10 Advanced RAS

5. Save the configuration and exit the UEFI setup menu.

MCA Recovery feature validation

In this section we show the MCA Recovery by injecting a hardware error and showing how the application recovers successfully. We show two event types, SRAR and SRAO.

SRAO Errors

We can get Software Recoverable Action Optional (SRAO) information from the /var/log/messages log file if we inject a matching error to trigger an SRAO. The SRAO output is shown in Figure 11.

```
kernel: mce: [Hardware Error]: Machine check events logged
kernel: mce: Uncorrected hardware memory error in user-access at 1cf6cce000
kernel: MCE 0x1cf6cce: dirty LRU page recovery: Recovered
mcelog: Hardware event. This is not a software error.
mcelog: MCE 0
mcelog: CPU 33 BANK 1 TSC 7f26fb4fe4cc
mcelog: RIP 33:400d5d
mcelog: MISC 86 ADDR fb7c85000
mcelog: TIME 1499943947 Thu Jul 13 07:05:47 2017
mcelog: MCG status:RIPV EIPV MCIP LMCE
mcelog: MCi status:
mcelog: Uncorrected error
mcelog: Error enabled
mcelog: MCi_MISC register valid
mcelog: MCi_ADDR register valid
mcelog: SRAO
mcelog: MCA: MEMORY CONTROLLER RD_CHANNELunspecified_ERR
mcelog: STATUS bd80000000100134 MCGSTATUS f
mcelog: MCGCAP f000c14 APICID 3 SOCKETID 0
mcelog: CPUID Vendor Intel Family 6 Model 85
```

Figure 11 SRAO Output

The message highlighted in red in Figure 11 shows the message “kernel: MCE 0x######: dirty LRU page recovery: Recovered” in the log.

SRAR Errors

We can get Software Recoverable Action Required (SRAR) information from the messages log if we inject a matching error to trigger an SRAR. The SRAR information is shown in Figure 12:

```
kernel: mce: [Hardware Error]: Machine check events logged
kernel: mce: Uncorrected hardware memory error in user-access at ff4dd8000
kernel: MCE 0xff4dd8: corrupted page was clean: dropped without side effects
kernel: MCE 0xff4dd8: clean LRU page recovery: Recovered
mcelog: Hardware event. This is not a software error.
mcelog: MCE 0
mcelog: CPU 36 BANK 0 TSC 8150da1bc5c6
mcelog: RIP 33:400ffe
mcelog: MISC 86 ADDR ff4dd8000
TIME 1499945081 Thu Jul 13 07:24:41 2017
mcelog: MCG status:RIPV EIPV MCIP LMCE
mcelog: MCi status:
mcelog: Uncorrected error
mcelog: Error enabled
mcelog: MCi_MISC register valid
mcelog: MCi_ADDR register valid
mcelog: SRAR
mcelog: MCA: Instruction CACHE Level-0 Instruction-Fetch Error
mcelog: STATUS bd8000000000c0150 MCGSTATUS f
mcelog: MCGCAP f000c14 APICID 9 SOCKETID 0
mcelog: CPUID Vendor Intel Family 6 Model 85
```

Figure 12 SRAR Output

You can find the message `kernel: MCE 0x######: clean LRU page recovery: Recovered` in the log. You may also find messages such as `MCE 0x######: Killing einj_mem_uc:3902 due to hardware memory corruption` if recovery action is needed to kill the process.

Address Range Mirroring

In this section we show how to enable Address Range Mirroring on a ThinkSystem server running Linux.

UEFI Setup

In mirroring mode, each DIMM in a pair must be identical in size and architecture. The channels are grouped in pairs with each channel receiving the same data. One channel is used as a backup of the other, which provides redundancy. So a specific DIMM population rule should be followed to enable the memory mirroring feature.

You can find a detailed description of the rules regarding DIMM population in the Maintenance Manual for each ThinkSystem server, available from the ThinkSystem Information Center:

http://thinksystem.lenovofiles.com/help/index.jsp

For example, you can find the mirroring mode DIMM population rule for the SR650 at the following web page:

http://thinksystem.lenovofiles.com/help/topic/7X05/dimm_installation_rules.html#dimm_installation_rules__memory_mirroring

In our lab environment, we installed eight DIMMs referring to mirroring mode of DIMM population rule in the SR650 server.

The steps we followed to enable memory mirroring are as follows:

1. Power on the server and press the F1 key to enter the UEFI Setup menu. Select **System Settings** → **Memory**.

2. Set Mirror Mode to **Partial** and enable **Mirror below 4GB** if you would like mirroring memory below 4GB. In our testing, we enabled Mirror below 4GB as shown in Figure 13 on page 16.

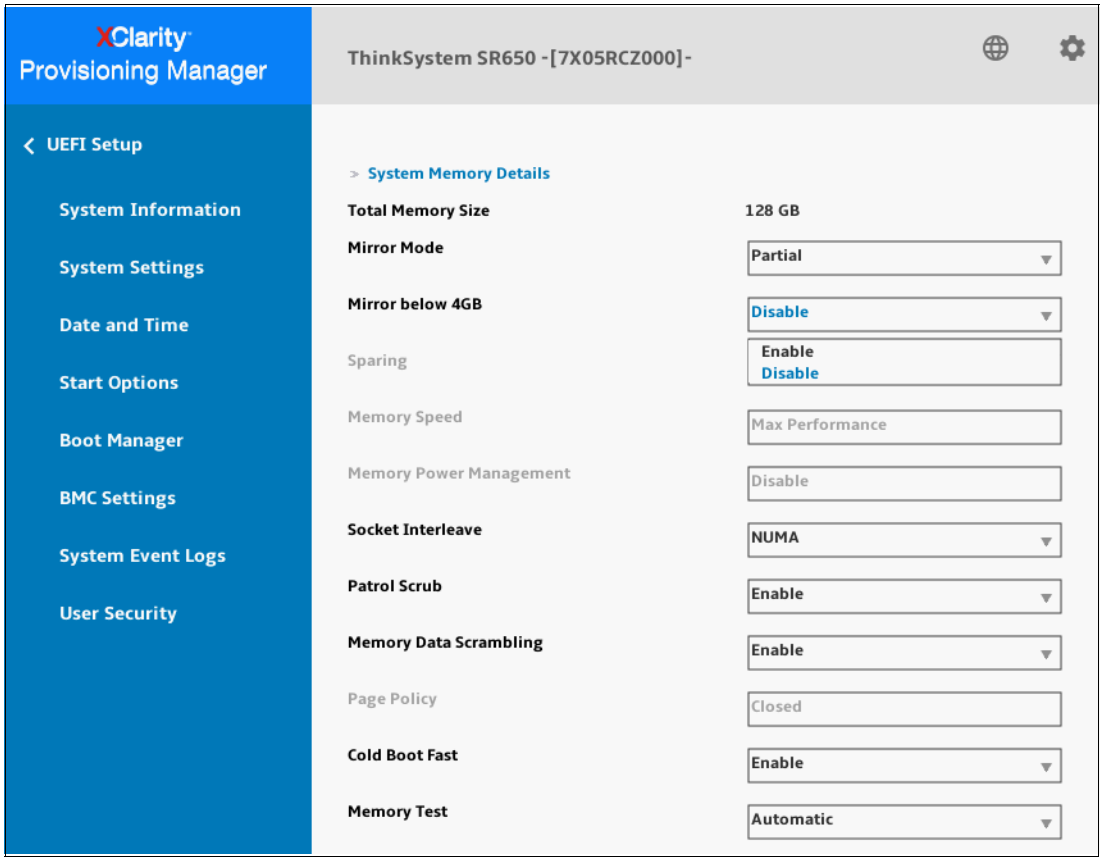

Figure 13 Memory Mirror Settings

3. Save the configuration and exit the UEFI setup menu.
4. The memory information of the memory mirror is shown on the system boot screen. The usable memory capacity is reduced according to the configuration that was set in UEFI. Figure 14 on page 17 shows the memory independent mode on the left side and Address Range Mirroring mode on the right side.

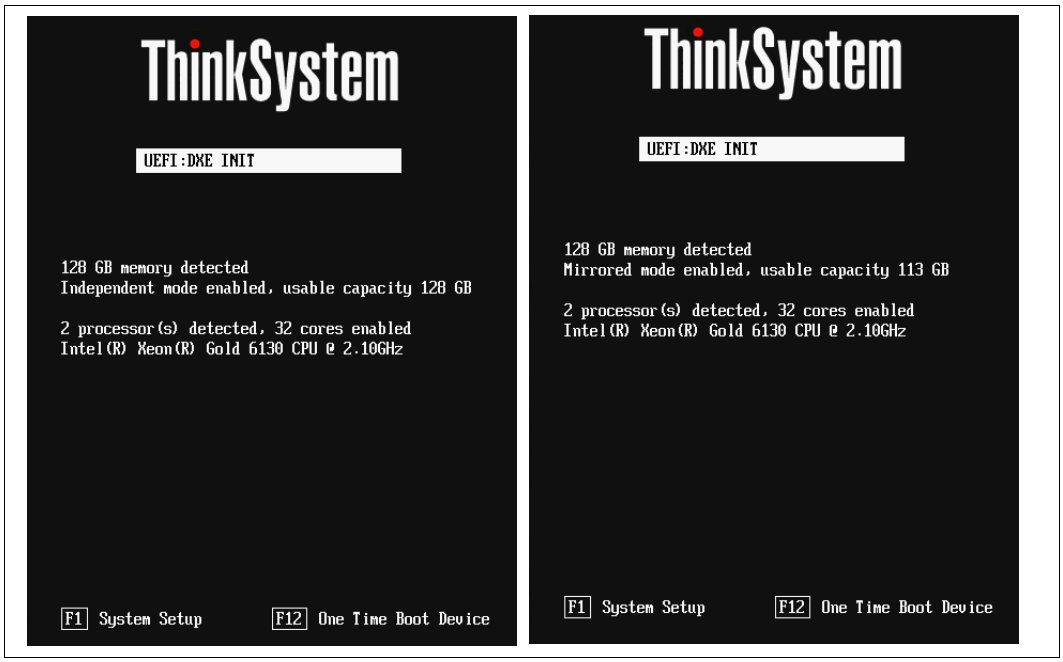

Figure 14 Memory Independent Mode (left) and Mirror Mode (right)

Address Range Mirroring feature validation

This section describes the method to validate Address Range Mirroring on your Linux system. First we describe how to request memory mirror size on Linux system by using efibootmgr tool. Then we introduce the solution for mirror support of zone allocator in the Linux kernel.

Linux-based Address Range Mirroring request

Two UEFI variables are used to configure the mirrored memory size for Address Range Mirroring:

- MirrorRequest: Written by the OS to request a new mirror configuration on the next system reboot.
- MirrorCurrent: Written by the firmware and read by the OS to communicate the current status of Address Range Mirroring.

The latest version of the Linux tool efibootmgr supports UEFI variables for Address Range Mirroring with command options `m` and `M`. The `m` parameter is used to enable mirror memory below 4GB. The `M` parameter represents the percentage of memory to mirror above 4GB.

For example, running the command **`efibootmgr -m t -M 20`** means that we enable mirror memory below 4GB and request 20% of the mirrored region above 4GB. This command is shown in Figure 15.

```
root@linux-pz68:~# efibootmgr -m t -M 20
BootOrder: 0000,0001,0002,0003
Boot0000* Red Hat Enterprise Linux
Boot0001* CD/DVD Rom
Boot0002* Hard Disk
Boot0003* Network
MirroredPercentageAbove4G: 0.00
MirrorMemoryBelow4GB: false
RequestMirroredPercentageAbove4G: 20.00
RequestMirrorMemoryBelow4GB: true
```

Figure 15 Memory Address Range Mirroring

As a result of this command, the memory information on the system boot screen will show the changed memory mirror percentage when we reboot the system, as shown in Figure 16.

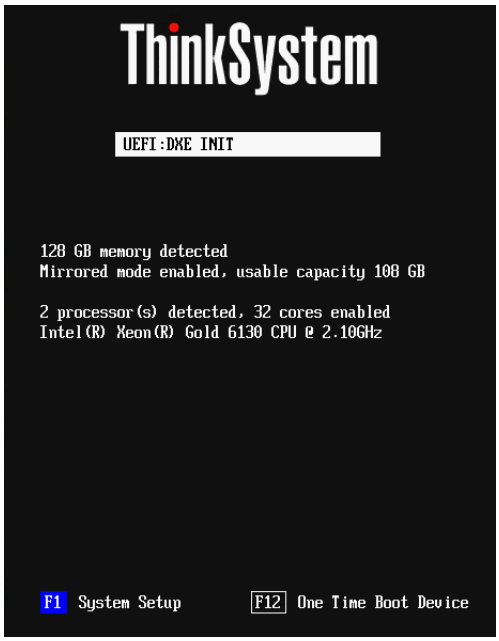

Figure 16 Memory Address Range Mirroring

The memory Address Range Mirroring information is passed via EFI memory map. We can run the **mmap** command in the UEFI shell to check the memory mirror information. The EFI_MEMORY_MORE_RELIABLE attribute (0x10000) in the EFI memory descriptor indicates the mirrored range. You can find that the attribute bit is set as 1, which marked in red in Figure 17.

```
Type       Start            End              # Pages          Attributes
BS_code    0000000000000000-0000000000007FFF  0000000000000008 000000000001000F
available  0000000000008000-000000000003FFFF  0000000000000038 000000000001000F
BS_code    0000000000040000-000000000009FFFF  0000000000000060 000000000001000F
available  0000000000100000-00000000453A0FFF  00000000000452A1 000000000001000F
BS_data    00000000453A1000-00000000454A0FFF  0000000000000100 000000000001000F
....
available  0000000FC0000000-00000019BFFFFFFF  0000000000A00000 000000000000000F
reserved   00000000000A0000-00000000000FFFFF  0000000000000060 0000000000000000
reserved   00000000AF800000-00000000BFFFFFFF  0000000000010800 0000000000000000
MemMapIO   00000000C0000000-00000000CFFFFFFF  0000000000010000 8000000000000001
MemMapIO   00000000FD000000-00000000FE7FFFFF  0000000000001800 800000000000100D
MemMapIO   00000000FED20000-00000000FED44FFF  0000000000000025 800000000000100D
MemMapIO   00000000FF000000-00000000FFFFFFFF  0000000000001000 800000000000100D

  reserved  :  76,128 Pages (311,820,288)
  LoaderCode:     261 Pages (1,069,056)
  LoaderData:      25 Pages (102,400)
  BS_code   :   4,488 Pages (18,382,848)
  BS_data   : 304,800 Pages (1,248,460,800)
  RT_code   :     432 Pages (1,769,472)
  RT_data   :   6,450 Pages (26,419,200)
  available : 26,310,440 Pages (107,767,562,240)
  ACPI_recl :     168 Pages (688,128)
  ACPI_NVS  :   2,720 Pages (11,141,120)
  MemMapIO  : 108,589 Pages (444,780,544)
Total Memory: 104,022 MB (109,075,595,264) Bytes
```

Figure 17 mmap Information (red highlight indicates the mirrored range)

Enable Address Range Mirroring support in the Linux kernel

If you want to allocate kernel memory requested from the mirrored region, you need to specify kernelcore=mirror in the kernel boot parameter. The non-mirrored region will be allocated to ZONE_MOVABLE and memory used by the kernel will only be allocated from the mirrored region.

The steps to enable address range mirroring are as follows:

1. To allocate requested kernel memory from the mirrored region, add `kernelcore=mirror` to the boot parameter by modifying grub.cfg file as shown in Figure 18.

```
linuxefi /vmlinuz-3.10.0-514.el7.x86_64 root=/dev/mapper/rhel-root ro
crashkernel=auto rd.lvm.lv=rhel/root rd.lvm.lv=rhel/swap rhgb quiet LANG=en_US.UTF-8
kernelcore=mirror
```

Figure 18 Modify grub.cfg

2. Reboot the OS and check the dmesg by running the command **`dmesg | grep -i mirror`**. The memory mirror information output as shown in Figure 19.

```
root@linux-pz68:~# dmesg | grep -i mirror
efi: Memory: 26359M/104744M mirrored memory
```

Figure 19 Checking mirroring status

You can also find that a specific zone type, Movable, is created in the Linux kernel, as shown in Figure 20.

```
On node 0 totalpages: 12988525
DMA zone: 64 pages used for memmap
DMA zone: 23 pages reserved
DMA zone: 3999 pages, LIFO batch:0
DMA32 zone: 10372 pages used for memmap
DMA32 zone: 663758 pages, LIFO batch:31
Normal zone: 45056 pages used for memmap
Normal zone: 2883584 pages, LIFO batch:31
Movable zone: 147456 pages used for memmap
Movable zone: 9437184 pages, LIFO batch:31
Initmem setup node 1 [mem 0xcc0000000-0x19bfffffff]
On node 1 totalpages: 13631488
Normal zone: 49152 pages used for memmap
Normal zone: 3145728 pages, LIFO batch:31
Movable zone: 163840 pages used for memmap
Movable zone: 10485760 pages, LIFO batch:31
```

Figure 20 Zone Allocation Information

The system should work without any downtime when uncorrected errors occur in the memory mirror region. We can inject several uncorrected fatal errors into the memory mirror region and check the messages log.

The error information is shown in Figure 21. All the uncorrected errors have been downgraded to corrected errors as indicated by the message highlighted.

```
kernel: mce_notify_irq: 14 callbacks suppressed
kernel: mce: [Hardware Error]: Machine check events logged
mcelog: Running trigger `dimm-error-trigger'
mcelog: Hardware event. This is not a software error.
mcelog: MCE 0
mcelog: CPU 0 BANK 7
mcelog: MISC 620000c020006086 ADDR 220000000
mcelog: TIME 883618763 Thu June  1 09:39:23 2017
mcelog: MCG status:
mcelog: MCi status:
mcelog: Error overflow
mcelog: Corrected error
mcelog: Error enabled
mcelog: MCi_MISC register valid
mcelog: MCi_ADDR register valid
mcelog: MCA: MEMORY CONTROLLER RD_CHANNEL0_ERR
mcelog: Transaction: Memory read error
mcelog: M2M: MscodDataRdErr
mcelog: STATUS dc00050001010090 MCGSTATUS 0
mcelog: MCGCAP f000c14 APICID 0 SOCKETID 0
mcelog: CPUID Vendor Intel Family 6 Model 85
```

Figure 21 Memory Error Information

Memory PFA

This feature is not currently implemented in the UEFI firmware on ThinkSystem servers, however there are plans to introduce it in 2018. Linux currently implements the feature based on mcelog.

Memory PFA feature validation

In this section, we list the steps needed to configure and launch the mcelog daemon on Linux, and how to check that memory PFA works correctly.

1. Check that the mcelog daemon works smoothly with the following command

   ps -ef | grep -i mcelog

   The output shown in Figure 22 means that the mcelog daemon is already launched in your system.

```
root@linux-pz68:~# ps -ef | grep -i mcelog
root        1092       1          0  2017 ?         00:00:00     /usr/sbin/mcelog
--ignorenodev --daemon --syslog
```

Figure 22 Check status of mcelog daemon

   if the mcelog daemon isn't running on the system, start it manually with the following command:

   `mcelog --ignorenodev --daemon --syslog`

2. Check the mcelog.conf configuration under the /etc/mcelog directory. You can set the soft-offline threshold in mcelog.conf and restart the mcelog daemon process to apply it. See the lines in red shown in Figure 23 for an example.

```
root@linux-pz68:~# cat /etc/mcelog/mcelog.conf | gerp -i threshold
# errors per DIMM exceeds the threshold
uc-error-threshold = 1 / 24h
ce-error-threshold = 10 / 24h
mem-uc-error-threshold = 100 / 24h
mem-ce-error-threshold = 100 / 24h
cache-threshold-trigger = cache-error-trigger
cache-threshold-log = yes
# Try to soft-offline a 4K page if it exceeds the threshold
memory-ce-threshold = 10 / 24h
```

Figure 23 Contents of the mcelog.conf file

3. Inject corrected errors into one memory page. Note that the quantity of injected errors should exceed the soft-offline threshold in your mcelog.conf file. The error information is logged in the dmesg as shown in Figure 24.

```
mce: [Hardware Error]: Machine check events logged
{3}[Hardware Error]: Hardware error from APEI Generic Hardware Error Source: 0
{3}[Hardware Error]: It has been corrected by h/w and requires no further action
{3}[Hardware Error]: event severity: corrected
{3}[Hardware Error]:  Error 0, type: corrected
{3}[Hardware Error]:   section_type: memory error
{3}[Hardware Error]:   physical_address: 0x0000001cef1dc000
{3}[Hardware Error]:   physical_address_mask: 0x00003fffffffffc0
{3}[Hardware Error]:   node: 2 card: 0 module: 0 rank: 0 column: 744
{3}[Hardware Error]:   error_type: 2, single-bit ECC
{3}[Hardware Error]:   DIMM location: not present. DMI handle: 0x0000
```

Figure 24 Error Information in dmesg

4. You can find the offlining information in the output of the mcelog client, where the failed memory page has been retired and set to offline status. See the lines highlighted in red in Figure 25.

```
root@linux-pz68:~# mcelog --client
Memory errors
SOCKET 0 CHANNEL any DIMM any
corrected memory errors:
   40 total
   40 in 24h
uncorrected memory errors:
   1 total
   1 in 24h
SOCKET 0 CHANNEL 0 DIMM any
corrected memory errors:
   40 total
   40 in 24h
uncorrected memory errors:
   1 total
   1 in 24h
SOCKET 1 CHANNEL any DIMM any
corrected memory errors:
   52 total
   52 in 24h
uncorrected memory errors:
   0 total
   0 in 24h
SOCKET 1 CHANNEL 0 DIMM any
corrected memory errors:
   48 total
   48 in 24h
uncorrected memory errors:
   0 total
   0 in 24h
Per page corrected memory statistics:
e971ea000: total 1 seen "1 in 24h" online
ea6184000: total 20 seen "20 in 24h" offline triggered
1cef1dc000: total 16 seen "16 in 24h" offline triggered
1d239b7000: total 11 seen "11 in 24h" offline triggered
```

Figure 25 Output of the mcelog command

The offlining information should also be logged in the messages log as shown in Figure 26.

```
mcelog: Corrected memory errors on page 1cef1dc000 exceed threshold 10 in 24h: 10 in
24h
mcelog: Location SOCKET:1 CHANNEL:0 DIMM:? []
mcelog: Running trigger `page-error-trigger'
mcelog: Offlining page 1cef1dc000
```

Figure 26 Output of the messages log

Summary

This white paper introduces the main memory RAS features on the Lenovo ThinkSystem Platform: MCA recovery, address range mirroring and Predictive Failure Analysis. It also shows the method to check if the RAS feature functioned well on your system by reading the key message in Linux log file.

Learn more

For more information, see these resources:

- Intel paper, *Address Range Partial Memory Mirroring*

  https://software.intel.com/en-us/articles/address-range-partial-memory-mirroring
- mcelog website

  http://www.mcelog.org/
- Documentation for EINJ error injector

  https://www.kernel.org/doc/Documentation/acpi/apei/einj.txt

Author

Neo Cui is a Linux Engineer at the Lenovo Data Center Group in Beijing, China. He joined Lenovo OS team after graduating from Ocean University of China, where he researched Ultra Wide-Band Communications. His main focus is the security and RAS features of Linux kernel development in Lenovo.

Thanks to the following people for their contributions to this project:

- Ocean He, Advisory OS Engineer
- Jason Liu, Advisory Firmware Architect
- David Watts, Lenovo Press

Notices

Lenovo may not offer the products, services, or features discussed in this document in all countries. Consult your local Lenovo representative for information on the products and services currently available in your area. Any reference to a Lenovo product, program, or service is not intended to state or imply that only that Lenovo product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Lenovo intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any other product, program, or service.

Lenovo may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Lenovo (United States), Inc.
1009 Think Place - Building One
Morrisville, NC 27560
U.S.A.
Attention: Lenovo Director of Licensing

LENOVO PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Lenovo may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Lenovo product specifications or warranties. Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Lenovo or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary.

Lenovo may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Lenovo Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Lenovo product, and use of those Web sites is at your own risk.

Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

This document was created or updated on October 31, 2017.

Send us your comments via the **Rate & Provide Feedback** form found at
http://lenovopress.com/lp0778

Trademarks

Lenovo, the Lenovo logo, and For Those Who Do are trademarks or registered trademarks of Lenovo in the United States, other countries, or both. These and other Lenovo trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by Lenovo at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of Lenovo trademarks is available on the Web at http://www.lenovo.com/legal/copytrade.html.

The following terms are trademarks of Lenovo in the United States, other countries, or both:

Lenovo®	ThinkSystem™
Lenovo(logo)®	TruDDR4™

The following terms are trademarks of other companies:

Intel, Xeon, and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.