High-Level Design (HLD)

Revision 1.2
Last Updated: 10/10/2002 - 10:13 AM

Panic Handler Enhancements for Linux 2.4

Primary Author(s): Andrew Cress

Abstract

This design document describes the Panic Handler Enhancements project for Linux* 2.4 kernels. This is a key functionality gap in Linux for the carrier-grade and enterprise spaces, where being able to determine the cause of a panic is critical. In a standard Linux kernel without these enhancements, if a panic occurs, there may be no trail of evidence at all.

In carrier-grade and many enterprise environments, glass houses or racks of 'headless' servers are managed from a remote site with intervention and personnel only sent onsite when the remote operations team has detected that it is required. Typically, a software fault would cause fail-over to a redundant server while the faulting system is rebooted and put back online.

In this environment, it is critical that information that can lead to the cause of the fault be kept and made available to the remote diagnostic team to determine the reason for the fault so that corrective actions can be taken.

Implementing this functionality is high on the list of Reliability, Availability, and Serviceability (RAS) features that the Linux development teams are addressing as part of various open-source projects to harden Linux for the carrier-grade and enterprise environments. This feature needs to integrate smoothly with other open-source projects, such as the Kernel Debugger (KDB) and Linux Kernel Crash Dump (LKCD) projects, to provide the maximum level of serviceability in the event a panic occurs.

Table of Contents

1. Introduction

1.1 Purpose of this Document

This High-Level Design (HLD) document specifies the implementation, including inter-component dependencies, and provides sufficient design detail that any product based on this HLD will satisfy the product requirements.

1.2 Document Scope

Anyone interested in understanding the Panic Handler Enhancements internal design should read this document.

2. Assumptions and Dependencies

It is assumed that the system platforms that this software is installed on will support some form of Intelligent Platform Management Interface (IPMI) functionality. This is the most widely adopted standard interface for platform management across a variety of vendors. Platforms that have other types of interfaces would have to implement a platform-specific implementation, assuming that the firmware supports these management functions.

If the platform does not support IPMI, the bmc_panic kernel changes are inert, but the code is designed to be portable to another system management interface standard. The Service Availability Forum (saforum.org) is working on an umbrella API (named SA PI, see appendix A) that could be used to group IPMI and other system management interfaces under a meta-standard. When this becomes available, these Panic Handler Enhancements will conform to that API so that non-IPMI platforms can be integrated more easily.

3. High-level Design

3.1 Design Decomposition

When a kernel panic happens or the watchdog timers expire, it is not possible to send an incident escalation up to the management middleware layer; these errors have to be escalated by the kernel prior to rebooting the system. The aim would be to alert the cluster of the failure quicker than a missed heartbeat, save as much state information as possible for diagnosing the failure, and initiate a fast reboot.

The current kernel panic handler has the following characteristics:

- When a panic occurs, it calls each routine in the panic notifier list. Parameters passed to each routine are: an event code (always zero), and a panic string.
- After all routines are complete, the kernel panic handler waits for the number of seconds specified in the panic timeout (exposed via `/proc/sys/kernel/panic`).
- If the panic timeout is non-zero, then the kernel panic handler reboots the machine after the panic timeout expires.

To support some of the additional enhancements, the kernel needs a method in the kernel to communicate with the firmware-level platform management features. Currently, IPMI is the only open-source interface for this capability, so the current design is targeted for IPMI-based systems. However, if another platform management interface can be linked into the kernel, bmc_panic could be modified to use it.

The Panic Handler Enhancements feature will add a module to the panic notifier list which will do the following actions:

- *Log the error condition in the Basic Mode Controller (BMC) System Event Log (SEL)*
  The IPMI Event Record will be logged in the SEL with a type code of 20h ('OS Critical Stop' type code). As much information as possible will be saved about the panic within the format of the SEL event message. Note that the SEL persists in firmware, whereas the Linux* system log (`syslog`) resides in the file system, either locally or remotely (via a network)..
- *Send a Simple Network Management Protocol (SNMP) trap through IPMI*
  Have the BMC send an alert (SNMP trap) message to a pre-configured IP address.
- *Raise an alarm to the alarms panel*
  Use IPMI and the BMC to raise a critical alarm to the alarms panel. This turns on one of the alarm LEDs (CRT, MAJ, MIN) on the front of the chassis, and sets a relay.
- *Initiate a system memory dump*
  Coordinate panic handling with the Linux Kernel Crash Dump (LKCD) project to properly initiate the memory dump. This theoretically should not require any additional handling outside the memory dump module.
- *Optionally, bring up the kernel debugger to permit remote debugging of the problem*
  Coordinate panic handling with the Kernel Debugger (KDB) project. This theoretically should not require any additional handling outside the KDB module.
- *Initiate a reboot*
  Force a reset of the system at the conclusion of a memory dump to restart the system. This could be handled simply by ensuring that the existing panic timeout is set to a non-zero value. Or, if that is insufficient, the panic handler could send a Cold or Warm Reset Command to the BMC. However it is done, the reset should be invoked from the panic handler after all of the notifier list tasks have been completed (including the memory dump, if configured).
- *Utilities*
  Add whatever user-space Linux tools are necessary to configure and use the panic handler enhancements.

Due to the nature of the condition(s) that lead to a fatal kernel error, some of these steps may not be possible, and the module containing these enhancements should take steps to continue on without further interruption if one of the aforementioned steps cannot be performed. For example, if the alarms panel LEDs are not present on a given system, this step should be skipped, but the other steps should still be performed.

Other requirements that were considered but not included in this project are:

- Handling the IPMI Watchdog timer on behalf of the KDB and the Kernel Memory Dump. The debugger and memory dump interaction with the watchdog timer does not need to involve the panic handler.
- Retrieving and logging from a kernel memory dump image any kernel messages that were not logged due to the panic. It is unresolved at this time where to send/store this additional information, and this information is recorded by the memory dump.

3.2 Internal Components

Table 1: Panic handler components, showing interactions between firmware, kernel, and user-space

User-space	***utilities*** `pefconfig, showsel, hwreset, tmconfig,` etc.				
Kernel	ipmidrvr (Intel, valinux*, or other)		***bmc_panic***	Kernel panic handler (notifier_list, panic_timeout)	
Firmware	BMC				
	Sensors	SEL	Alarms Panel	BMC LAN	Serial EMP

The shaded areas in Table 1 are those covered by the Panic Handler Enhancements feature.

Since it is only called if the kernel has called panic, the **`bmc_panic`** kernel module must be assure that it does not depend on a loadable module for its functions. The `bmc_panic` module could theoretically be a loadable module, but due to the possibility of a panic occurring without the `bmc_panic` module loaded in physical memory, and the fact that the `bmc_panic` module is not very big, the `bmc_panic` module should be directly linked into the kernel. It would be simplest for `bmc_panic` to utilize an IPMI driver (or some future open-source platform management driver) that was already linked into the kernel, if it exposed an interface that had the ability not to introduce additional timer waits in processing the commands that `bmc_panic` needs. There are several potential Linux IPMI kernel drivers, and they may or may not be suitable for `bmc_panic`, so in the current implementation, `bmc_panic` contains a subset of the valinux* IPMI driver to perform its tasks, but does not expose any IPMI interfaces outside the

module, so that it will not conflict with any normal IPMI modules that may also be loaded. The `bmc_panic` does not allocate any new memory after the kernel is loaded, and is assumed to persist in memory.

The user-space ***utilities*** shall be able to use one of the common IPMI drivers that are likely to be installed on a Linux system, such as the Intel® IPMI (`/dev/imb`) or the valinux IPMI (`/dev/ipmikcs`) drivers. These utilities shall be used to configure `bmc_panic` functions, as well as other related administrative functions. Of course, the utilities are run as needed and have no persistent resources.

Some of these configuration functions could be performed by rebooting to the service partition and running DOS tools, such as System Setup Utility (SSU). However, there are several issues with this approach. First, even though they could be performed remotely, the DOS configuration functions would have to be performed manually, and this is impractical for a large number of servers. Second, rebooting to DOS takes the server out of service and reduces the availability of the system. Therefore, implementing Linux command-line utilities allows these functions to be performed while the Linux OS is running, and also allows them to be automated via a script to facilitate large numbers of servers.

The following utilities shall be implemented:

- **`pefconfig`**
  This utility is intended to automatically configure the Platform Event Filter (PEF) table and the BMC LAN Parameters so that the BMC is ready to send an SNMP Alert over the LAN (eth0) when events are logged into the SEL. It can also display the values of parameters. In addition, since the Linux LAN parameters are usually already configured, this utility tries to retrieve those parameters and use them to automatically set the same values for the BMC LAN. If the user specifies a specific command-line option, the `pefconfig` should override the default values automatically obtained from Linux. Since some of the BMC IPMI v1.5 systems have 12 pre-defined entries (0-11), the default PEF table entry used to insert the OS Critical Stop filter entry is offset 12, which is the next available entry. In a system with greater than or fewer than 12 pre-defined entries, this offset can be changed from the command-line options.
- **`showsel`**
  This utility should show the SEL records from the BMC, and should display them with textual descriptions for each event. Also, it is periodically necessary to clear the event log to prevent it from filling up and rejecting new events; therefore, the utility should have an option to clear the log. Many customers may want to consolidate the SEL log with the standard Linux log. To accomplish this, the utility should have the capability to read any SEL records not already saved, and write them to the `syslog`.
  This utility decodes the various types of records similar to the way that the Direct Platform Control (DPC) SEL feature (part of ISC Remote Console) does. In addition, `showsel` decodes the new OS Critical Stop messages and displays the available panic string characters in a readable format.
- **`hwreset`**
  This utility should be able to perform a hardware (chassis) reset of the system, or a power

down. It should also have an option to allow the user to initiate a reboot into the service partition.

? **`tmconfig`**
This utility should configure the firmware to support several types of serial parameter configurations. First, it should set up the serial parameters to support shared Basic-Mode IPMI and BIOS Console Redirection. Next, it should set up the serial parameters to support shared Terminal Mode IPMI and BIOS Console Redirection. Terminal Mode is defined in the IPMI v1.5 specification, and will be supported in a future release of firmware.

3.3 Internal Data Structure Map

The IPMI v1.5 specification, Table 36-3, defines the sensor types for SEL records, as used by `showsel` and the `bmc_panic` kernel module.

The IPMI v1.5 specification, Table 15-2, defines the PEF table entries, as used by `pefconfig`.

The IPMI v1.5 specification, Table 19-4, defines the LAN Configuration Parameters, as used by `pefconfig`.
The IPMI v1.5 specification, Table 20-4, defines the Serial Configuration Parameters, as used by `tmconfig`.

The IPMI v1.5 specification, Table 18-13, describes the Master Write-Read Command used to access the Alarms Panel from the `bmc_panic` kernel module.

The IPMI v1.5 specification, Table 22-4, describes the Chassis Control Command used to reset the system by `hwreset` and `bmc_panic`.

3.4 Internal Methods

All IPMI calls in `bmc_panic` pass through a common IPMI transfer function for timed requests and responses. All IPMI methods within `bmc_panic` must be static (not public/exposed), unless uniquely defined for necessary communication with other critical kernel modules (like `panic.c`).

For the utilities, all IPMI requests and responses shall pass through a common `ipmi_cmd()` routine so that this routine can format them appropriately and direct them to whichever IPMI driver is detected when the utility was loaded. A simple IPMI command will be issued at startup to check `/dev/imb` (and if unsuccessful, `/dev/ipmikcs`) to determine which driver is present. After detection, a flag is set to use the selected driver. Support for both devices will be present in the code for each utility.

3.5 System Dependencies and File Structures

The bmc_panic module is linked into the kernel under /drivers/char as a "misc" character module. If the kernel configuration defines CONFIG_BMCPANIC as 'y', then the misc.c module must call an initialization routine from the bmc_panic module.

The IPMI driver, either Intel or valinux, is needed by the utilities, and will have a device node that resides in the /dev directory. The Intel driver device node is called /dev/imb, while the valinux device node is called /dev/ipmikcs. The utilities are executables and can reside in any suitable directory.

3.6 External Data Structures

In order for the system to reboot properly after a panic, the panic_timeout must be set. The default setting of zero (0) means an infinite timeout. This value is counted in seconds, and a normal value for bmc_panic and LKCD is five seconds. This can be set via the /proc interface as /proc/sys/kernel/panic, or it can be set via the lilo "append" command as "panic=5". Note that the panic_timeout is not preserved across reboots. When the Panic Handler Enhancements feature is installed, the default behavior should be to set this timeout via lilo.conf so that it is set even if a panic occurs while lilo is loading the OS.

The bmc_panic kernel module inserts itself into the panic_notifier_list when the module is loaded, so that it will be notified, along with other handlers in the list, when a panic occurs. The kernel panic handler core (panic.c) controls each of the handlers in the panic notifier list. It invokes each of the handlers: bmc_panic, LKCD, and/or KDB. When all of the active handlers in the list have completed their tasks, the panic timeout starts. By default, some Linux distributions (for example, Red Hat*) set the panic timeout to zero (infinite). When either panicsel or LKCD is installed, the panic timeout is set to five seconds. This means that once all the handlers are done, the system reboots in five seconds. When KDB is one of the handlers, the timer doesn't start until 'go' or 'reboot' is entered. The following code illustrates the format of the panic notifier list structure.

```
struct notifier_block {
     int (*notifier_call)(struct notifier_block *self,
                                  unsigned long, void);
     struct notifier_block next;
     int priority;
}
/* Few, if any, list members should set the priority to 0
(highest).  Bmc_panic sets this value to 200, so that it
comes after more critical list members, like LKCD. */
```

The bmc_panic module also needs to view the panic_string, so this variable should be exported by panic.c.

3.7 External APIs

The only external APIs exposed are those documented in the IPMI v1.5 specification.

3.8 Design Strategies

There was a proof-of-concept Panic Handler Enhancements kernel patch that used Intel IPMI driver code; it was configured as a loadable module. This worked, but had several design flaws. It had license conflicts with the GPL kernel source, and it exposed scenarios where the loadable module might not be loaded and could fail to load when a panic had occurred. Therefore, in the initial release, part of a GPL valinux IPMI driver was integrated into the Panic Handler Enhancements kernel module (`bmc_panic`). If enabled, the configuration parameter (CONFIG_BMCPANIC) links the module into the kernel. The kernel configuration should not present the user with the option to make `bmc_panic` a loadable module.

This design could be simplified to use interfaces exposed by a common IPMI driver if it were linked into the kernel and provided the ability not to introduce additional timer waits in processing the commands that `bmc_panic` needs. In a future kernel release, it is expected that this will be possible, allowing the bmc_panic module to remove the valinux IPMI code and use the common IPMI code. It is also possible in the future to modify the bmc_panic module to use another platform management interface, if the platform had an open-source driver which supported similar functions.

Also, due to the reasons outlined in Section 3.2, the scope of the utilities was enlarged so that this Panic Handler Enhancements feature could be configured and used without the need for DOS utilities such as SSU.

3.8.1 Product Installation Strategy

When the kernel is installed, either from a binary RPM or from kernel source, the `bmc_panic` module will be part of the kernel. The CONFIG_BMCPANIC kernel configuration parameter should be enabled by default. The BMC LAN features also need to be configured, either by a DOS SSU program, or by the `pefconfig` Linux utility provided. For more details, see the `panicsel` README included with the software.

The target systems will normally have Intel® Server Control (ISC) software available on the Resource CD and this software (or at least an IPMI driver) should be installed before the Panic Handler utilities (also an RPM) are installed. When these utilities are installed, the `pefconfig` utility and any other configuration settings will be changed to allow the Panic Handler Enhancements to perform as many of the functions as are supported by the system.

3.8.2 Initialization and Shutdown Strategies

Upon initialization of the kernel, the miscellaneous char devices are initialized (`drivers/char/misc.c`), and the `bmc_panic` init routine is registered in the `panic_notifier_list`.

Shutdown is irrelevant to the `bmc_panic` module, since it is part of the kernel. During a panic, however, it is assumed that other modules (such as software RAID, SCSI, and LKCD) are more important than `bmc_panic`, so its notifier priority is purposely set to 200 to ensure that it follows all of the critical modules.

3.8.3 Interoperability and Compatibility Support

Because the `bmc_panic` kernel module is self-contained and resides in the kernel source tree, it does not have compatibility issues in the kernel. However, it requires IPMI support in the firmware of the target system. If `bmc_panic` is enabled in the kernel, and the target system does not support IPMI, an error message will be sent to the log; the `bmc_panic` will not harm other kernel operations.

3.8.3.1 Target hardware/software environment

The target platform for this software is a Linux 2.4 OS on a server that supports IPMI. This software makes use of the IPMI interface and requires IPMI support in the platform firmware (i.e. BMC).

It is assumed that the IPMI v1.5 compliant firmware (BMC) supports the following features: sensors, SEL, BMC LAN, and serial Emergency Management Port (EMP). Also, some platforms will have alarm panel LEDs (Critical, Major, Minor, and Power). For a given system, the version of BMC firmware may be important. For instance, with the Intel® TSRLT2/TSRMT2 systems, the version of BMC must be version 54 or greater, and the SSU, if used, must correspondingly be version 1.R.3 or greater. See the `panicsel` README for more details. If one or more of the features is not supported by the IPMI platform, that step will be skipped when a panic occurs.

3.8.3.2 Imported Interfaces

The Panic Handler utilities require that a version of an IPMI driver be installed. Intel's IPMI driver (`ipmidrvr` or `/dev/imb`) comes as part of the ISC package for Intel® servers. It is planned to be released as an open-source driver, and can be obtained from http://www.intel.com/design/servers/ipmi/tools.htm. The valinux IPMI driver (`/dev/ipmikcs` or `/dev/ipmi/kcs`), written by San Mehat, is available as an open-source kernel module. If an IPMI driver on a given Linux system supports the /dev/imb or /dev/ipmikcs interfaces, the Panic Handler utilities can use it.

3.8.3.3 Exported and Defined Software Interfaces

The interfaces to this feature are through the command-line utilities. Corresponding 'man' pages will be delivered with the utilities, and will describe in detail how to use each utility.

3.8.3.4 Product Standards

Platforms must support the IPMI specification with the current design. It is possible, if a common API can be defined to include other system management interfaces, and their access methods can be included in the GPL Linux kernel, that non-IPMI platforms could then be included. Currently, if a vendor wished to use this feature on a non-IPMI platform, new code would have to be written to substitute for the logging and alerting function calls in `bmc_panic`.

3.8.4 Addressing Performance Issues

The `bmc_panic` does not impact the overall performance of the system during normal operations. If a panic occurs, system performance is secondary to saving the information and quickly notifying an administrator. Two performance considerations need to be addressed, however:

1) How much latency is there between the occurrence of a kernel panic and notification to the network administrator?
2) How much time does it take for the server to become operational after a panic occurs, assuming a reboot is sufficient to accomplish this?

3.8.5 Locking and Synchronization Strategy

There are two concerns related to synchronization:

1) Ensuring that critical modules get priority when a system panic happens. The `bmc_panic` should not prevent LKCD or software RAID from getting enough priority to function.
2) Ensuring that the internal IPMI driver within `bmc_panic` does not cause conflicts during normal operations with the regular IPMI driver. The BMC serializes input requests and can handle multiple sessions, but the `bmc_panic` module only uses the IPMI interface to the BMC in two instances, initialization and panic, so it should not conflict with normal operations. This issue would be simplified if a suitable IPMI driver were linked into the kernel so that `bmc_panic` could use it instead.

3.8.6 Buffer Strategy

The `bmc_panic` module uses one buffer for requests and one for responses. Since the IPMI commands are sequential, this does not pose a problem.

The Panic Handler utilities also use one buffer for requests and one for responses. Since the IPMI commands are sequential, this does not pose a problem. The utilities select which IPMI driver to use at run-time, so when separate command buffers are needed for different drivers, these buffers are allocated from the subroutine's stack.

3.9 Additional Features Required

Table 2 shows some additional requirements for the Panic Handler Enhancements feature that were added after its initial release.

Any proposed future enhancements will be included in the project TODO list.

Table 2: Correlation of requirements to HLD specifications

Requirements	Design Implementation
Linux* Panic Handler Enhancements shall provide additional configurable features to the default panic handler.	The `bmc_panic` kernel module is configurable in the kernel configuration parameters (CONFIG_BMCPANIC). It should be enabled by default on all Linux platforms that support IPMI.
One of the following panic actions shall be initiated: ? Reboot (soft or hard) ? Power down ? Power cycle	The default for the `bmc_panic` kernel module is to perform a reset of the system (soft reboot). Options to take other actions are configurable via kernel configuration parameters that are linked to the CONFIG_BMCPANIC parameter governing this feature. Each type of reboot (soft reboot, hard reboot, power cycle) clears an additional level of firmware, BIOS and adapter buffers, but then requires some additional time to re-initialize them on the way back up. It would be possible to implement a `/proc` interface to dynamically modify this action; however, this does not seem necessary, since the `hwreset` utility already allows the user to perform these actions under Linux.
Documentation - Panic Handler shall provide a detailed specification that documents all configuration options and the behavior of each feature.	The User Guide for the Panic Handler Enhancements feature shall describe all configuration options and their behavior.

Appendix A: References

IPMI Specification, Version 1.5: http://www.intel.com/design/servers/ipmi/index.htm

The Panic Handler Enhancements open-source project: http://panicsel.sourceforge.net

The Service Availabilty Forum Platform Interface specification: http://www.saforum.org

Appendix B: Abbreviations, Acronyms and Definitions

Abbreviation	Description
`bmc_panic`	The name of the Panic Handler Enhancements kernel module, which communicates to the BMC via IPMI commands.
BMC	Baseboard Management Controller – The BMC processes firmware-level functions, as well as many server management functions.
DPC	Direct Platform Control – A server remote management utility that interacts directly with the BMC firmware from a remote system. It is packaged with ISC.
EMP	Emergency Management Port – This serial port can be configured for access to server management functions via the SSU or Panic Handler utilities, such as `tmconfig`.
GPL	GNU Public License – The default license for Linux software, required for most Linux kernel modules.
HLD	High Level Design
IP	Internet Protocol
IPMI	Intelligent Platform Management Interface
ipmidrvr	Name of the Intel® IPMI driver RPM.
ISC	Intel Server Control – An Intel® server management daemon that runs on the system being managed to provide sensors and control to remote management applications. This software is provided with Intel® servers on the accompanying Resource CD.
KDB	Kernel Debugger. See http://systemras.sourceforge.net/
LED	Light-Emitting Diode
LKCD	Linux Kernel Crash Dump. See http://systemras.sourceforge.net/

Abbreviation	Description
OS	Operating System
PEF	Platform Event Filter
RAID	Redundant Array of Inexpensive Disks
RAS	Reliability, Availability, and Serviceability
RPM	RPM Package Manager
SCSI	Small Computer System Interface
SEL	System Event Log
SNMP	Simple Network Management Protocol
SSU	System Setup Utility – A DOS program used to configure firmware and BIOS parameters. It is provided with Intel® servers on the accompanying Resource CD.
valinux	The name of a company that developed an open-source IPMI driver, currently maintained by San Mehat.