Kdump, A Kexec-based Kernel Crash Dumping Mechanism

Vivek Goyal
IBM
vgoyal@in.ibm.com

Eric W. Biederman
Linux NetworkX
ebiederman@lnxi.com

Hariprasad Nellitheertha
IBM
hari@in.ibm.com

Abstract

Kdump is a kexec based kernel crash dumping mechanism, which is being perceived as a reliable crash dumping solution for Linux®. This paper begins with brief description of what kexec is and what it can do in general case, and then details how kexec has been modified to boot a new kernel even in a system crash event.

Kexec enables booting into a new kernel while preserving the memory contents in a crash scenario, and kdump uses this feature to capture the kernel crash dump. Physical memory layout and processor state are encoded in ELF core format, and these headers are stored in a reserved section of memory. Upon a crash, new kernel boots up from reserved memory and provides a platform to retrieve stored ELF headers and capture the crash dump. Also detailed are ELF core header creation, dump capture mechanism, and how to configure and use the kdump feature.

1 Introduction

Various crash dumping solutions have been evolving over a period of time for Linux and other UNIX® like operating systems. All solutions have their pros and cons, but the most important consideration for the success of a solution has been the reliability and ease of use. Kdump is a crash dumping solution that provides a very reliable dump generation and capturing mechanism [01]. It is simple, easy to configure and provides a great deal of flexibility in terms of dump device selection, dump saving mechanism, and plugging-in filtering mechanism.

The idea of kdump has been around for quite some time now, and initial patches for kdump implementation were posted to the Linux kernel mailing list last year [03]. Since then, kdump has undergone significant design changes to ensure improved reliability, enhanced ease of use and cleaner interfaces. This paper starts with an overview of the kdump design and development history. Then the limitations of existing designs are highlighted and this paper goes on to detail the new design and enhancements.

Section 2 provides background of kexec and kdump development. Details regarding how kexec has been enhanced to boot-into a new kernel in panic events are covered in section 3. Section 4 details the new kdump design. Details about how to configure and use this mechanism are captured in Section 5. Briefly discussed are advantages and limitations of this approach in section 6. A concise description of current status of project and TODOs are in-

cluded in Section 7.

2 Background

This section provides an overview of the kexec and original kdump design philosophy and implementation approach. It also brings forward the design deficiencies of kdump approach so far, and highlights the requirements that justified kexec and kdump design enhancements.

2.1 Kexec

Kexec is a kernel-to-kernel boot-loader [07], which provides the functionality to boot into a new kernel, over a reboot, without going through the BIOS. Essentially, kexec pre-loads the new kernel and stores the kernel image in RAM. Memory required to store the new kernel image need not be contiguous and kexec keeps a track of pages where new kernel image has been stored. When a reboot is initiated, kexec copies the new kernel image to destination location from where the new kernel is supposed to run, and after executing some setup code, kexec transfers the control to the new kernel.

Kexec functionality is constituted of mainly two components; kernel space [08] and user space [02]. Kernel space component implements a new system call `kexec_load()` which facilitates pre-loading of new kernel. User space component, here onwards called kexec tools, parses the new kernel image, prepares the appropriate parameter segment, and setup code segment and passes the this data to the running kernel through newly implemented system call for further processing.

2.2 A Brief History of Kdump Development

The core design principle behind this approach is that dump is captured with the help of a custom built kernel that runs with a small amount of memory. This custom built kernel is called capture kernel and is booted into upon a system crash event without clearing crashed kernel's memory. Here onwards, for discussion purposes, crashing kernel is referred to as first kernel and the kernel which captures the dump after a system crash is called capture kernel.

While capture kernel boots, first kernel's memory is not overwritten except for the small amount of memory used by new kernel for its execution. Kdump used this feature of kexec and added hooks in kexec code to boot into a capture kernel in a panic event without stomping crashed kernel's memory.

Capture kernel used the first 16 MB of memory for booting and this region of memory needed to be preserved before booting into capture kernel. Kdump added the functionality to copy the contents of the first 16 MB to a reserved memory area called backup region. Memory for the backup region was reserved during the first kernel's boot time, and location and size of the backup region was specified using kernel config options. Kdump also copied over the CPU register states to an area immediately after the backup region during a crash event [03].

After the crash event, the system is unstable and usual device shutdown methods can not be relied upon, hence, devices are not shutdown after a crash. This essentially means that any ongoing DMAs at the time of crash are not stopped. In the above approach, the capture kernel was booting from the same memory location as the first kernel (1 MB) and used first 16 MB to boot, hence, it was prone to corruption due to any on-going DMA in that re-

gion. An idea was proposed and a prototype patch was provided for booting the capture kernel from a reserved region of memory instead of a default location. This reduced the chances of corruption of the capture kernel due to ongoing DMA [04] [05]. Kdump's design was updated to accommodate this change and now the capture kernel booted from reserved location. This reserved region was still being determined by kernel config options [06].

Despite the fact that the capture kernel was booting from a reserved region of memory, it needed first 640 KB of memory to boot for SMP configurations. This memory was required to retrieve configuration data like the MP configuration table saved by BIOS while booting the first kernel. It was also required to place the trampoline code needed to kick-start application processors in the system. Kdump reserved 640 KB of memory (backup region) immediately after the reserved region, and preserved the first 640 KB of memory contents by copying it to a backup region just before transferring control to capture kernel. CPU register states were being stored immediately after the backup region [06].

After booting, capture kernel retrieved the saved register states and backup region contents, and made available the old kernel's dump image through two kernel interfaces. The first one was through the `/proc/vmcore` interface, which exported the dump image in ELF core format, and other one being the `/dev/oldmem`, which provided a linear raw view of memory.

2.3 Need for Design Enhancement

Following are some of the key limitations of the above approach that triggered the design enhancement of kdump.

1. In the design above, kexec pre-loads the capture kernel wherever it can manage to grab a page frame. At the time of crash, the capture kernel image is copied to the destination location and control is transferred to the new kernel. Given the fact that the capture kernel runs from a reserved area of memory, it can be loaded there directly and extra copying of kernel can be avoided. In general terms, kexec can be enhanced to provide a fast reboot path to handle booting into a new kernel in crash events also.

2. Capture kernel and the associated data is pre-loaded and stored in the kernel memory, but there is no way to detect any data corruption due to faulty kernel programming.

3. During the first kernel boot, kdump reserves a chunk of memory for booting the capture kernel. The location of this region is determined during kernel compilation time with the help of config options. Determining the location of reserved region through config options is a little cumbersome. It brings in hard-coding in many places, at the same time it is static in nature and a user has to compile the kernel again if he decides to change the location of reserved region.

4. Capture kernel has to boot into a limited amount of memory, and to achieve this, the capture kernel is booted with user defined memory map with the help of `memmap=exactmap` command line options. User has to provide this user defined memory map while pre-loading the capture kernel and need to be explicitly aware of memory region reserved for capture kernel. This process can be automated by kexec tools and these details can be made opaque to the user.

5. When the capture kernel boots up, it needs to determine the location of the backup region to access the crashed kernel's backed-up memory contents. Capture kernel receives this information through hard coded config options. It also retrieves the saved register states assuming these to be stored immediately after the backup region and this introduces another level of hard-coding.

   In this approach, the capture kernel is explicitly aware of the presence of the backup region, which can be done away with. In general, there is no standard format for the exchange of information between two kernels which essentially makes two kernel dependent on each other, and it might now allow kernel skew between the first kernel and the capture kernel as kernel development progresses.

6. The `/proc/vmcore` implementation does not support discontiguous memory systems and assumes memory is contiguous, hence exports only one ELF program header for the whole of the memory.

3 Kexec On Panic

Initially, kexec was designed to allow booting a new kernel from a sane kernel over a reboot. Emergence of kdump called for kexec to allow booting a new kernel even in a crash scenario. Kexec has now been modified to handle system crash events, and it provides a separate reboot path to a new kernel in panic situations.

Kexec as a boot-loader supports loading of various kinds of images for a particular platform. For i386, vmlinux, bzImage, and multiboot images can be loaded. Capture kernel is compiled to load and run from a reserved memory location which does not overlap with the first kernel's memory location (1 MB). However, currently only a vmlinux image can be used as a capture kernel. A bzImage can not be used as capture kernel because even if it is compiled to run from a reserved location, it always first loads at 1 MB and later it relocates itself to the memory location it was compiled to run from. This essentially means that loading bzImage shall overwrite the first kernel's memory contents at 1 MB location and that is not the desired behavior.

From here on out the discussion is limited to the loading of a vmlinux image for i386 platform. Details regarding loading of other kind of images is outside the scope of this paper.

3.1 Capture Kernel Space Reservation

On i386, the default location a kernel runs from is 1 MB. The capture kernel is compiled and linked to run from a non default location like 16 MB. The first kernel needs to reserve a chunk of memory where the capture kernel and associated data can be pre-loaded. Capture kernel will directly run from this reserved memory location. This space reservation is done with the help of `crashkernel=X@Y` boot time parameter to first kernel, where `X` is the the amount of memory to be reserved and `Y` indicates the location where reserved memory section starts.

3.2 Pre-loading the Capture Kernel

Capture kernel and associated data are pre-loaded in the reserved region of memory. Kexec tools parses the capture kernel image and loads it in reserved region of memory using `kexec_load()` system call. Kexec tools manage a contiguous chunk of data belonging to the same group in the form of segment. For example, bzImage code is considered as one

segment, parameter block is treated as another segment and so on. Kexec tools parses the capture kernel image and prepares a list of segments and passes the list to kernel. This list basically conveys the information like location of various data blocks in user space and where these blocks have to be loaded in reserved region of memory. `kexec_load()` system call does the verification on destination location of a segments and copies the segment data from user space to kernel space. Capture kernel is directly loaded into the memory where it is supposed to run from and no extra copying of capture kernel is required.

`purgatory` is an ELF relocatable object that runs between the kernels. Apart from setup code, purgatory also implements a sha256 hash to verify that loaded kernel is not corrupt. In addition, `purgatory` also saves the contents to backup region after the crash (section 4.3).

Figure 1 depicts one of the possible arrangements of various segments after being loaded into a reserved region of memory. In this example, memory from 16 MB to 48 MB has been reserved for loading capture kernel.

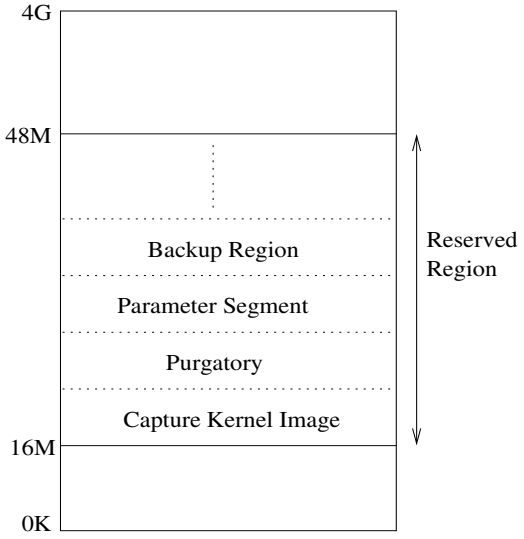

Figure 1: Various Data Segments in Reserved Region

3.3 Post Crash Processing

Upon a crash, kexec performs a minimum machine shutdown procedure and then jumps to the `purgatory` code. During machine shutdown, crashing CPU sends the NMI IPIs to other processors to halt them. Upon receiving NMI, the processor saves the register state, disables the local APIC and goes into halt state. After stopping the other CPUs, crashing CPU disables its local APIC, disables IOAPIC, and saves its register states.

CPU register states are saved in ELF note section format [09]. Currently the processor status is stored in note type NT_PRSTATUS at the time of crash. Framework provides enough flexibility to store more information down the line, if needed. One kilobyte of memory is reserved for every CPU for storing information in the form of notes. A final null note is appended at the end to mark the end of notes. Memory for the note section is allocated statically in the kernel and the memory address is exported to user space through `sysfs`. This address is in turn used by kexec tools while generating the ELF headers (Section 4.2).

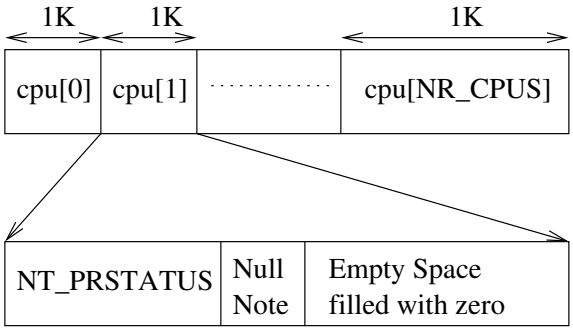

Figure 2: Saving CPU Register States

After saving register states, control is transferred to `purgatory`. `purgatory` runs sha256 hash to verify the integrity of the capture kernel and associated data. If no corruption is detected, purgatory goes on to copy the first

640 KB of memory to the backup region (Section 4.3). Once the backup is completed control flow jumps to start of the new kernel image and the new kernel starts execution.

4 Kdump

Previous kdump design had certain drawbacks which have been overcome in the new design. Following section captures the details of the new kdump design.

4.1 Design Overview

Most of the older crash dumping solutions have had the drawback of capturing/writing out the dump in the context of crashing kernel, which is inherently unreliable. This led to the idea of first booting into a sane kernel after the crash and then capturing the dump. Kexec enables kdump to boot into the already loaded capture kernel without clearing the memory contents and this sets the stage for a reliable dump capture.

The dump image can be represented in many ways. It can be a raw snapshot of memory read from a device interface similar to `/dev/mem`, or it can be exported in ELF core format. Exporting a dump image in ELF core format carries the advantage of being a standard approach for representing core dumps and provides the compatibility with existing analysis tools like `gdb, crash`, and so on . Kdump provides ELF core view of a dump through `/proc/vmcore` interface and at the same time it also provides `/dev/oldmem` interface presenting linear raw view of memory.

ELF core headers encapsulate the information like processor registers, valid RAM locations, and backup region, if any. ELF headers are prepared by kexec tools and stored in a reserved memory location along with other segments as shown in the Figure 3.

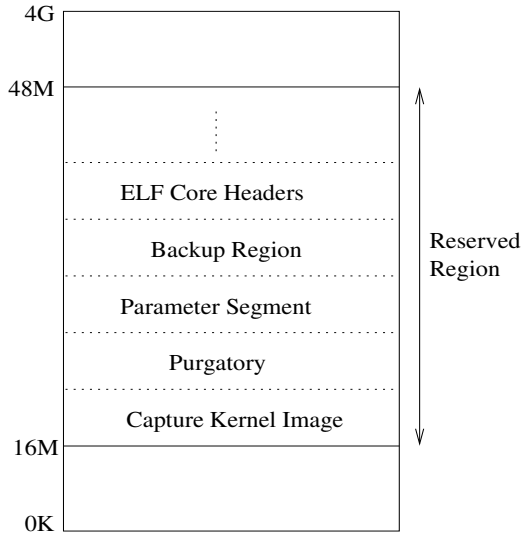

Figure 3: ELF Core Headers in Reserved Region

Memory for ELF core headers is reserved by bootmem allocator during first kernel boot using the `reserve_bootmem()` function call. Upon crash, system boots into new kernel and stored ELF headers are retrieved and exported through `/proc/vmcore` interface.

This provides a platform for capturing the dump image and storing it for later analysis. Implementation details are discussed in following sections of this paper.

4.2 ELF Core Header Generation

Kdump uses the ELF core format to exchange the information about dump image, between two kernels. ELF core format provides a generic and flexible framework for exchange of dump information. The address of the start of these headers is passed to the new kernel through a command line option. This provides

a cleaner interface between the two kernels, and at the same time ensures that the two kernels are independent of each other. It also allows kernel skew between the crashing kernel and the capture kernel, which essentially means that version of the crashing kernel and the capture kernel do not need to be the same. Also, an older capture kernel should be able to capture the dump for a relatively newer first kernel.

Kexec tools are responsible for ELF core header generation. ELF64 headers are sufficient to encode all the required information, but `gdb` can not open a ELF64 core file for 32 bit systems. Hence, kexec also provides a command line option to force preparation of ELF32 headers. This is useful for the users with non PAE systems.

One PT_LOAD type program header is created for every contiguous memory chunk present in the system. Information regarding valid RAM locations is obtained from `/proc/iomem`. Considering system RAM as a file, physical address represents the offset in the file. Hence the `p_offset` field of program header is set to actual physical address of the memory chunk. `p_paddr` is the same as `p_offset` except in case of a backup region (Section 4.3). Virtual address (`p_vaddr`) is set to zero except for the linearly mapped region as virtual addresses for this region can be determined easily at the time of header creation. This allows a restricted debugging with `gdb` directly, without assistance from any other utility used to fill in virtual addresses during post crash processing.

One PT_NOTE type program header is created per CPU for representing note information associated with that CPU. Actual notes information is saved at the time of crash (Section 3.3), but PT_NOTE type program header is created in advance at the time of loading the capture kernel. The only information required at this point is the address of location where actual notes section reside. This address is exported to user space through `sysfs` by kexec. Kexec user space tools read in the `/sys/kernel/crash_notes` file and prepare the PT_NOTE headers accordingly.

In the event of memory hotplug, the capture kernel needs to be reloaded so that the ELF headers are generated again reflecting the changes.

4.3 Backup Region

Capture kernel boots from the reserved area of memory after a crash event. Depending on the architecture, it may still need to use some fixed memory locations that were used by the first kernel. For example, on i386, it needs to use the first 640 KB of memory for trampoline code for booting SMP kernel. Some architectures like ppc64 need fixed memory locations for storing exception vectors and other data structures. Contents of these memory locations are copied to a reserved memory area (backup region) just after crash to avoid any stomping by the capture kernel. `purgatory` takes care of copying the contents to backup region (Section 3.2).

Capture kernel/capture tool need to be aware of the presence of a backup region because effectively some portion of the physical memory has been relocated. ELF format comes in handy here as it allows to envelop this information without creating any dependencies. A separate PT_LOAD type program header is generated for the backup region. The `p_paddr` field is filled with the original physical address and the `p_offset` field is populated with the relocated physical address as shown in the Figure 4.

Currently, kexec user space tools provide the backup region handling for i386, and the first 640 KB of memory is backed-up. This code is

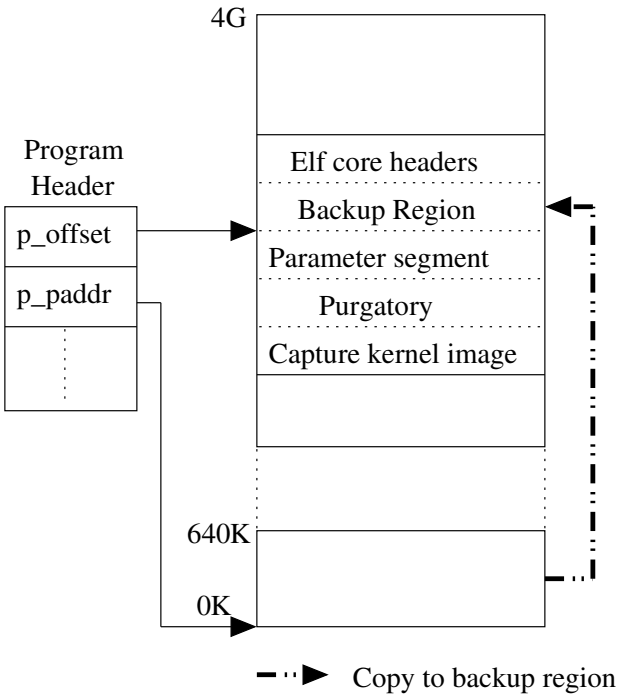

Figure 4: Saving Contents To Backup Region

more or less architecture dependent. Other architectures can define their own backup regions and plug-in the implementations into existing kexec user space code.

4.4 Booting into Capture Kernel

The capture kernel is compiled to boot from a non-default memory location. It should not stomp over crashed kernel's memory contents to be able to retrieve a sane dump. Hence, capture kernel is booted with an user defined memory map instead of the one provided by BIOS or one passed in parameter segment by kexec. The command line option `memmap=exactmap` along with `memmap=X@Y` is used to override BIOS provided memory map and define user memory map.

These boot time parameters are automatically added to command line by kexec tools while loading the capture kernel and it's details are opaque to user. Internally, kexec prepares a list of memory regions that the capture kernel can safely use to boot into, and appropriate `memmap` options are appended to the command line accordingly. The backup region and ELF header segments are excluded from this list to avoid stomping of these memory areas by new kernel.

Address of start of ELF header segment is passed to the capture kernel through the `elfcorehdr=` command line option. This option is also added automatically to command line by kexec tools.

4.5 Dump Capture Mechanism

Once the capture kernel has booted there are multiple design options for dump capturing mechanism. Few of them are as following.

- **Kernel Space**

  Export ELF core image through the `/proc/vmcore` interface which can be directly used by ELF core format aware analysis tools such as `gdb`. Also export raw linear view of memory through device interface `/dev/oldmem`. Other crash analysis tools can undergo required modifications to adapt to these formats.

  This is an easy to use solution which offers a wide variety of choices. Standard tools like `cp, scp, ftp` can be used to copy the image to the disk either locally or over the network. `gdb` can be used directly for limited debugging. The flip side is that the `/proc/vmcore` code is in the kernel and debugging the kernel code is relatively harder.

- **User Space**

  User space utilities which read the raw physical memory through suitable interfaces like `/dev/oldmem` and write out the dump image.

- **Early User Space**

  Utilities that run from initial ramdisk and perform a raw dump to pre-configured disk. This approach is especially useful in a scenario when root file system happens to be corrupted after the crash.

For now, we stick to kernel space implementation and other solutions (user space or early user space) can evolve slowly to cater to wide variety of requirements. The following sections cover implementation details.

4.5.1 Accessing Dump Image in ELF Core Format

ELF core headers, as stored by crashed kernel, are parsed and the dump image is exported to user space through `/proc/vmcore`. Backup region details are abstracted in ELF headers, and `/proc/vmcore` implementation is not even aware of the presence of the backup region. The physical address of the start of the ELF header is passed to the capture kernel through the `elfcorehdr=` command line option. Stored ELF headers undergo a sanity check during the `/proc/vmcore` initialization and if valid headers are found then initialization process continues otherwise `/proc/vmcore` initializaiton is aborted and the `vmcore` file size is set to zero.

CPU register states are saved in note sections by crashing kernel and one PT_NOTE type program header is created for every CPU. To be fully compatible with ELF core format, all the PT_NOTE program headers are merged into one during the `/proc/vmcore` initialization. Figure 5 depicts what a `/proc/vmcore` exported ELF core images looks like.

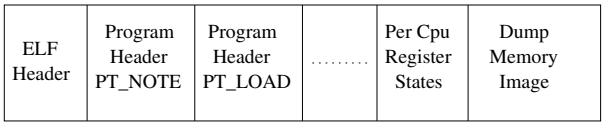

Figure 5: ELF Core Format Dump Image

Physical memory can be discontiguous and this means that offset in the core file can not directly map to a physical address unless memory holes are filled with zeros in the core file. On some architectures like IA64, holes can be big enough to deter one from taking this approach.

This new approach does not fill memory holes with zeros, instead it prepares one program header for every contiguous memory chunk. It maintains a linked list in which each element represents one contiguous memory region. This list is prepared during init time and also contains the data to map a given offset to respective physical address. This enables `/proc/vmcore` to determine where to get the contents from associated with a given offset in ELF core file when a read is performed.

`gdb` can be directly used with `/proc/vmcore` for limited debugging. This includes processor status at the time of crash as well as analyzing linearly mapped region memory contents. Non-linearly mapped areas like vmalloced memory regions can not be directly analyzed because kernel virtual addresses for these regions have not been filled in ELF headers. Probably a user space utility can be written to read in the dump image, determine the virtual to physical address mapping for vmalloced regions and export the modified ELF headers accordingly.

Alternatively, the `/proc/vmcore` interface can be enhanced to fill in the virtual addresses in exported ELF headers. Extra care needs to be taken while handling it in kernel space because determining the virtual to physical mapping shall involve accessing VM data structures of the crashed kernel, which are inherently unreliable.

4.5.2 Accessing Dump Image in linear raw Format

The dump image can also be accessed in linear raw format through the `/dev/oldmem` interface. This can be especially useful for the users who want to selectively read out portions of the dump image without having to write out the entire dump. This implementation of `/dev/oldmem` does not possess any knowledge of the backup region. It's a raw dummy interface that treats the old kernel's memory as high memory and accesses its contents by stitching up a temporary page table entry for the requested page frame. User space application needs to be intelligent enough to read in the stored ELF headers first, and based on these headers retrieve rest of the contents.

5 How to Configure and Use

Following is the detailed procedure to configure and use the kdump feature.

1. Obtain a kernel source tree containing kexec and kdump patches.
2. Obtain appropriate version of kexec-tools.
3. Two kernels need to be built in order to get this feature working. The first kernel is the production kernel and the second kernel is the capture kernel. Build the first kernel as follows.
   - Enable `kexec system call` feature.
   - Enable `sysfs file system support` feature.
4. Build the capture kernel as follows.
   - Enable `kernel crash dumps` feature.
   - The capture kernel needs to boot from the memory area reserved by the first kernel. Specify a suitable value for `Physical address where kernel is loaded`.
   - Enable `/proc/vmcore` support. (Optional)
5. Boot into the first kernel with the commandline `crashkernel=Y@X`. Pass appropriate values for X and Y. Y denotes how much memory to reserve for the second kernel and X denotes at what physical address the reserved memory section starts. For example, `crashkernel=32M@16M`
6. Preload the capture kernel using following commandline.

   ```
   kexec -p <capture kernel>
   --crash-dump --args-linux
   --append="root=<root-dev>
   maxcpus=1 init 1"
   ```

7. Either force a panic or press `Alt SysRq c` to force execution of kexec on panic. System reboots into the capture kernel.
8. Access and save the dump file either through the `/proc/vmcore` interface or the `/dev/oldmem` interface.
9. Use appropriate analysis tool for debugging. Currently `gdb` can be used with the `/proc/vmcore` for limited debugging.

6 Advantages and Limitations

Every solution has its advantages and limitations and kdump is no exception. Section 6.1 highlights the advantages of this approach and limitations have been captured in Section 6.2.

6.1 Advantages

- More reliable as it allows capturing the dump from a freshly booted kernel as opposed to some of other methods like `LKCD` , where dump is saved from the context of crashing kernel, which is inherently unreliable.
- Offers much more flexibility in terms of choosing the dump device. As dump is captured from a newly booted kernel, virtually it can be saved to any storage media supported by kernel.
- Framework is flexible enough to accommodate filtering mechanism. User space or kernel space based filtering solutions can be plugged in, unlike firmware based solutions. For example, a kernel pages only filter can be implemented on top of the existing infrastructure.

6.2 Limitations

- Devices are not shutdown/reset after a crash, which might result in driver initialization failure in capture kernel.
- Non-disruptive dumping is not possible.

7 Status and TODOS

Kdump has been implemented for i386 and initial set of patches are in -mm tree. Following are some of the TODO items.

- Harden the device drivers to initialize properly in the capture kernel after a crash event.
- Modify `crash` to be able to analyze kdump generated crash dumps.
- Port kdump to other platforms like x86_64 and ppc64.
- Implement a kernel pages only filtering mechanism.

8 Conclusions

Kdump has made significant progress in terms of overcoming some of the past limitations, and is on its way to become a mature crash dumping solution. Reliability of the approach is further bolstered with the capture kernel now booting from a reserved area of memory, making it safe from any DMA going on at the time of crash. Dump information between the two kernels is being exchanged via ELF headers, providing more flexibility and allowing kernel skew. Usability of the solution has been further enhanced by enabling the kdump to support PAE systems and discontiguous memory.

Capture kernel provides `/proc/vmcore` and `/dev/oldmem` interfaces for retrieving the dump image, and more dump capturing mechanisms can evolve based on wide variety of requirements.

There are still issues with driver initialization in the capture kernel, which need to be looked into.

References

[01] Hariprasad Nellitheertha, *The kexec way to lightweight reliable system crash dumping*, Linux Kongress, 2004.

[02] The latest kexec tools patches, *http://www.xmission.com/~ebiederm/files/kexec/*

[03] Initial Kdump patches, *http://marc.theaimsgroup.com/?l=linux-kernel&m=109274443023485&w=2*

[04] Booting kernel from non default location patch (bzImage), *http://www.xmission.com/~ebiederm/files /kexec/2.6.8.1-kexec3/broken-out /highbz-Image.i386.patch*

[05] Booting kernel from non default location patch (vmlinux), *http://www.xmission.com/~ebiederm/files /kexec/2.6.8.1-kexec3/broken-out /vmlinux-lds.i386.patch*

[06] Improved Kdump patches *http://marc.theaimsgroup.com/?l=linux-kernel&m=109525293618694&w=2*

[07] Andy Pfiffer, Reducing System Reboot Time with kexec, *http://developer.osdl.org/rddunlap/kexec/whitepaper/kexec.pdf*

[08] Hariprasad Nellitheertha, Reboot Linux Faster using kexec, *http://www-106.ibm.com/developerworks /linux/library/l-kexec.html*

[09] Tool Interface Standard (TIS), Executable and Linking Format (ELF) Specification (version 1.2)

Acknowledgments

The authors wish to express their sincere thanks to Suparna Bhattacharya who had been continuously providing ideas and support. Thanks to Maneesh Soni for numerous suggestions, reviews and feedback. Thanks to all the others who have helped us in our efforts.

9 Legal Statement