

The kexec Way to Lightweight Reliable System Crash Dumping

Hariprasad Nellitheertha

hari@in.ibm.com

Suparna Bhattacharya

suparna@in.ibm.com

Linux Technology Center

IBM India Software Labs

Adam Litke

litke@us.ibm.com

Martin J Bligh

mbligh@us.ibm.com

Linux Technology Center

IBM Corporation

Abstract

Established ways of generating reliable crash dumps in the event of operating system failures have been around for years. Most of these have been developed and perfected to suit specific platform environments. Linux[®], however, has been deployed on the most diverse set of platforms and devices, and hence requires a more universally applicable crash dumping solution. Also, it has become evident over the past few years that the successful adoption of a crash dumping solution in Linux requires that the solution to be simple and easy to use.

A promising approach employs kernel-to-kernel bootloaders to reboot to a new kernel without clearing memory and thereafter write out the preserved state to the target device. In this paper, we talk about a new approach to developing the next generation first failure data capture facility for Linux. This new approach makes use of the kexec feature to reboot to a new kernel upon a system failure. The second kernel boots with a very minimal amount of

memory making available the rest of memory to be written out as a dump.

Additionally, the new feature provides ELF format file interfaces to the previous kernel's memory, thereby reducing the dump write out operation to a mere file copy. Such an abstraction also makes it possible to use standard tools such as gdb to perform dump analysis. It has been possible to obtain these features while still retaining a simple and minimally invasive code base.

This takes us well on the way to a fairly powerful and reliable crash dumping support for Linux.

1 Introduction

In this paper, we talk in detail about the new kexec-based crash dumping feature being developed for the Linux 2.6 kernel. We discuss the design and implementation of this new technique, talk about its advantages over existing approaches and provide an update on the status of its implementation and future

work. In Section 2, we prepare the background by providing brief material about related topics. We talk about existing approaches to reliable crash dumping, both in Linux and in other operating systems, and a brief overview of the kexec feature. We also provide insights into a few other kernel-to-kernel bootloaders that are similar in approach to kexec. Next, we talk about existing implementations that have used kernel-to-kernel bootloaders and make a case for the new approach taken. Section 3 gives an overview of the design philosophy behind the kexec-based dumping solution. We then provide the implementation details of this feature in Section 4 explaining the techniques used to preserve the entire system memory across reboots, the interface provided to write out the dump, and the method to extract the dump. In section 5, through an illustration, we explain how to install and setup this tool and the mechanism to obtain a dump. Section 6 talks about the benefits of the particular approach taken here in comparison with other dumping solutions available. Details of known limitations of this approach are also provided. Section 7 describes the current status of the project and also the future enhancements that have been planned. The paper concludes with a summary in Section 8.

2 Background

This section provides some background to the concepts involved in the design of this feature. Section 2.1 provides an insight into the known approaches to reliable crash dumping, in Linux and other operating systems. Section 2.2 provides an overview of the kexec feature implemented for the 2.6 series of Linux kernels. Section 2.3 provides brief insights into some other projects which are similar to kexec, and their current status.

Section 2.4 provides an overview of some projects that have used kernel to kernel bootloaders to collect crash dumps. Section 2.5 brings out the need for a new approach towards a bootloader-based crash dumping tool.

2.1 Various approaches to reliable crash dumping

Several techniques have been employed in Linux and other operating systems in an attempt to overcome the fundamental problem of writing to disk from a system in a suspect state. Some of the commonly used techniques are discussed below.

1. Stand-alone Dumper - This technique involves the use of an operating system independent, self-reliant dumping module which takes over the system completely at the time of a system problem and performs the dump. The module uses firmware calls to perform the I/O and always has to dump the entire system memory contents. Such utilities have been built for the OS/2 and OS/390 operating systems. Though this scheme works very reliably, its main drawbacks are the inability to dump selective portions of memory and the need to maintain low-level firmware code for every type of hardware. Also, special cases where the operating system may have bypassed system firmware and altered certain settings on specific devices complicate the switchback. For example, on ppc64, the interfaces for performing some of the firmware operations are lost after operating system initialization and only a subset of the firmware capabilities are available for use. In such cases, implementing a stand-alone dumper may be a difficult proposition.
2. Dump-specific driver interfaces - Also known as dedicated dump drivers, these modules are kernel-independent chunks of code that perform I/O to the dump devices without too much dependence on the failing kernel. Such techniques have been implemented for the AIX and IRIX operating systems. The advantage of such techniques is that they do allow flexibility in controlling memory that needs to be dumped. On the flip side, dump support needs to be added for every target device and it is very difficult to make such modules independent of the system infrastructure. In the case of Linux, the diskdump [12] tool that is available can be placed in this category. These tools have drivers (or at least hooks)

for specific devices such as SCSI, PCI, and network cards.

3. Kernel reboot based dumper - This scheme depends on methods which allow the system to be rebooted to a second kernel without clearing system memory contents. Implementations of this technique can be found in LKCD [01] and MCLX. The approach taken in these implementations goes to great lengths to avoid having to set aside a large memory area just for dumping; however, in the worst case, memory pressure might limit how much of memory can be saved in the dump. For example, in the LKCD implementation which uses kexec, though pages which have been written are re-used as dump pages and compression of the dump pages is performed, if the compression ratio is not good, there is a possibility of not being able to capture complete system memory in the dump.

In addition to the above approaches, there also exists another technique that has been thought of. This technique, which can be called the mini kernel dumper, has, until this date, been a research or academic effort. The mini kernel dumper involves performing the dump operations from a separate mini kernel which runs at a separate privilege level. This approach is only possible if the operating system runs at a privilege level lower than the maximum allowed by the system.

2.2 Kexec

Kexec is a new feature being developed for the 2.6 series of the Linux kernel. Kexec allows a Linux kernel to load and boot another kernel image. The system does not have to go through the BIOS or the platform loader to boot to a new kernel. This greatly reduces the reboot time in systems, making it a very useful feature for enterprise-class systems looking for low downtime and quick turn-around time. Kexec can be used for very quick recovery at times of system panic. Kexec was designed to be flexible in that it can be used to load any ELF binary image, and not just the Linux kernel.

Kexec has two components to it; a user space component and a kernel component. The interaction between the two components is through a system call (`sys_kexec`). The user space component does the job of copying over the contents of the image to be loaded into memory. The system call can be used for loading the kernel image, through the kernel space component, into dynamic kernel memory. Subsequently, when a quick reboot is desired, the new image is overlaid onto the existing kernel image and is booted. The entire process of setting up the operating environment, going through the BIOS and bootloader, is skipped which makes reboots very fast.

While fast booting is one of the primary issues that kexec attempts to address, it has also been found very useful in crash dumping solutions. Further discussion of kexec and its design is beyond the scope of this paper. [02], [03] and [04] provide more details on kexec.

2.3 kernel-to-kernel bootloaders

There are at least three other known projects that have attempted kexec-type approaches to booting kernels. They are LOBOS [09], Two Kernel Monte [06] and bootimg [07]. None of the three projects have upgraded their implementation for the 2.6 series of Linux kernels, as yet.

LOBOS or Linux Os Boots OS is essentially a system call that allows a running Linux kernel to boot a new kernel without using the BIOS in any way. LOBOS is essentially targeted at clustering machines and has been ported for a certain set of mainboards only.

Two Kernel Monte, similarly, provides a way of loading another kernel image and restarting the system from that kernel. Two Kernel Monte has been implemented as a kernel module and hence does not require either patching or rebuilding of the kernel.

Bootimg is again similar to the other two projects mentioned above. It allows multi-stage boot procedures wherein a simple boot loader can load the first (possibly light-weight) Linux kernel from which the eventual Linux kernel can be loaded, using the full functionality of the Linux system.

2.4 Boot loader based crash dumping tools

Prior to the current implementation, there existed at least two other projects that have used the concept of kernel-to-kernel bootloaders to develop a crash dumping solution. The mcore dump implementation [08] by Mission Critical Linux and the memdev driver in LKCD both use these mechanisms. Both these projects are quite similar in nature and in fact the LKCD implementation is influenced by mcore and attempts to address the shortcomings of the mcore design.

The mcore project used booting as the bootloader. System dump is triggered either due to a kernel panic or due to user initiation. The crash dump is initially stored in memory itself. A destination map is created up front in memory to hold the dump pages. This map is created out of the available free and user pages. At the time of dumping, kernel pages are compressed and stored onto these destination pages. Then the system reboots to a new kernel using booting without clearing the memory used by the dump pages. Once the reboot is complete, the dump pages are written out, from user space, to a file and memory is cleared. Thus, mcore dump can dump only kernel pages. The mcore dump tool is not available for the Linux 2.6 kernel.

The LKCD implementation makes use of the kexec feature. Similar to mcore, it makes use of a destination map to map the dump pages. But, it has the ability to dynamically grow the map depending upon the need. It makes use of memory pages, which have already been written out as dump pages, to extend the memory map. This way, complete system dumps can be obtained under most conditions. This implementation also has the technique of marking the dump pages by their criticality which helps in salvaging at least the most important parts of the dump in case the dump process not completing. The LKCD implementation is available for the Linux 2.6 kernel.

2.5 The need for a new bootloader based crash dumping tool

There are several reasons to choose the kexec based crash dumping approach. There are quite a few shortcomings in the alternate approaches that have been discussed above. The stand-alone dumper-based approach does not provide the flexibility and control in terms of the sections of memory that can be dumped. Complete memory snapshots may not be desirable at all times. Further, the architectural trend towards very large memory systems means that being able to save only the more relevant parts of memory state becomes a practical necessity. The independent device driver-based dumpers have their own limitations. Maintaining such drivers for the increasingly growing device base in Linux is a huge proposition. Our implementation is a fairly device driver-independent dumping technique.

Despite the existence of other boot loader-based crash dumping facilities, a new tool is needed. One of the primary requirements a crash dumping tool has to satisfy is that it needs to do very minimal processing in the context of the failing kernel. Most of the existing implementations do not fulfill this criteria satisfactorily. Writing out the dump in the context of the failed kernel is never completely reliable. Because crash dumping is the primary “first failure data capture” tool, it has to be highly reliable. The previous boot loader-based implementations have dependencies on the state of the memory, at the time of failure, in order to succeed. Further, invasiveness of the dump code into the core kernel code is also an issue. The crash dump code has to be as simple and non-invasive as possible. Another equally important requirement that most of the current implementations fail to address is ease of use. From a user point of view, setting up a system to take crash dumps must be a very simple task. Unfortunately, in most cases, setting up a system for dumping involves several pieces to be installed and configured right, often quietly unforgiving of user errors. This could affect the chances of dumps being saved when a failure actually occurs.

Therefore, these key requirements were considered when designing the new solution:

1. The design needs to be simple and not overly invasive into the core kernel code.
2. The design needs to be highly reliable. It should be able to dump under any situation.
3. The design needs to cover most architectures. Porting the tool to different architectures and maintaining it must be easy.
4. The design needs to provide the flexibility to either dump the entire memory contents or perform selective dumping.
5. The design must provide flexibility with regard to dump targets. It must be possible to dump locally or across the network without too many changes to the setup.
6. The whole operation of setting up the system for dumping, including installation, configuration and dump write out must be very easy to perform.

3 Design Overview

The core design principles behind the kexec-based crash dumping facility are discussed in this section.

The primary consideration behind the design was to ensure that very little processing was performed in the context of the failed kernel. This requirement comes from the fact that the kernel which has “failed” is inherently unreliable and so any operation performed in that context decreases the reliability of the dump collected.

The above requirement meant that the challenge was to quickly get to a safer, more reliable context while still preserving memory from the failed context. Fortunately, kexec makes it possible to overcome this challenge through a small trick. Because rebooting through kexec skips the entire BIOS (or firmware) stage, memory from the previous kernel is not cleared while booting to the new kernel. In the case of a reboot due to a crash, the second kernel is made to boot with a very small amount of memory. For example, the second kernel can be made to boot with

just 16 MB of memory. The rest of the system memory is unused by the second kernel and is available for write out.

Even with the above technique, there are still a couple of problems. The first one concerns the memory used by the second kernel. Because the second kernel overwrites this chunk of memory, it is not available for write out to the dump file. This problem is overcome by using a reserved area of memory, equal in size to the memory used by the second kernel, and copying over the contents of the memory chunk before rebooting. More details on how this is achieved are described in the next section. The second issue is with regard to the possible memory constraint that the second kernel may face. The resolution of this problem lies in realizing the fact that the sole purpose of the second kernel is to write out the dump to a destination device. For this purpose, it can be a custom built kernel which has minimal driver support and only necessary features. This kernel is also started with very minimal services, such as init level 1 or 3. Once the second kernel is up and the dump has been written out, a regular reboot is performed and the system is restored for normal operation.

The next stage of the dumping process starts with the system booting into the second kernel, and deals with providing the interfaces required to write out the dump into a destination device. The objective here is to provide a simple and easy to use interface for dump write out and also to ensure that the dump file adheres to a standard format so that existing tools could be used for analyzing the dumps and the need for a new tool does not arise.

There are two components to the process of accessing the dump contents in the second kernel. The first part deals with the way the memory is treated and the second part deals with how the dump file is abstracted out to user space. The first part is achieved by treating the unused (previous kernel’s) memory as a high memory device. Essentially, in the second boot, the system is equivalent to a machine with a huge amount of high memory. Pages from this region are mapped in with a pte during the write out.

With such a mechanism to read the dump contents in place, the dump file is abstracted as a file in /proc. For example, the file corresponding to the complete

dump is available as `/proc/vmcore` in the second boot. The user just needs to copy out the dump just like an ordinary file. Similarly, `/proc/vmcore-kern` will act as the interface for a kernel pages-only dump. The other advantage of this scheme is that the dump files are available in standard ELF format. Tools like `gdb` and `crash` (with modifications) can be used to analyze the dump files. In fact, providing the file interface means that the dump need not even be written out. The analysis can be performed right on the files in `/proc`.¹

The user is also provided with a linear view of the crash dump. This is made possible through the creation of a `/dev/oldmem` device. This is analogous to the `/dev/mem` interface in Linux and is possibly suitable for various custom tools that might want to look at or dump out specific parts of memory.

Details on how each of these components have been implemented are provided in the next section.

4 Implementation details

This sections details the techniques used to implement the `kexec`-based crash dumping tool. We discuss the implementation in two sections. In section 4.1, we discuss how the memory contents corresponding to the failing kernel are made available in the second `kexec`-booted kernel. Section 4.2 talks about the interfaces provided to access the dump and also the mechanisms available to write out the dump file.

For discussion purposes, all subsequent references to “first kernel” refer to the kernel which fails (due to panic or hang) and all references to “second kernel” imply the `kexec`-booted kernel from the where the dump is retrieved.

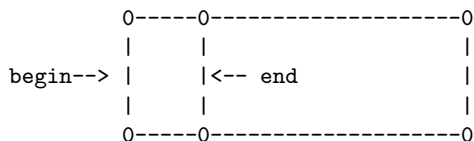
4.1 Memory preserving reboot

As mentioned above, by booting the second kernel with very little memory and by leaving the memory from the previous kernel untouched, the objective of accessing the failed kernel’s memory in the second

kernel is achieved. In the current implementation, the second kernel is booted with only 16MB of memory. The following diagram shows the memory layout of the system corresponding to the first and the second kernels.



Figure 1: Memory layout of the first kernel



Note that the size of the window depends upon factors such as available RAM and the platform.

Figure 2: Memory layout of the second kernel

As seen from figure 2, the second kernel, while leaving most of the first kernel’s memory untouched, does overwrite the first few megabytes of memory. Obviously, without the original contents of this region of memory, the dump is incomplete. The first task therefore is to ensure that this chunk of memory which the second kernel overwrites is preserved across reboots. To make this possible, some groundwork needs to be done in the first kernel, very early during boot up. A backup region, equal in size to the memory used by the second kernel, is created during the boot up of the first kernel. This involves setting aside a chunk of memory as not-available for use by the first kernel. The `reserve_bootmem()` call is used to make this reservation. The first kernel’s usable memory, post this reservation, looks as shown in figure 3

The location and size of this backup region is configurable during kernel compilation. The default value for the location of the backup region is 128MB and the default value for the size is 16MB.

¹Due to memory constraints in this boot, a detailed analysis may not be feasible.

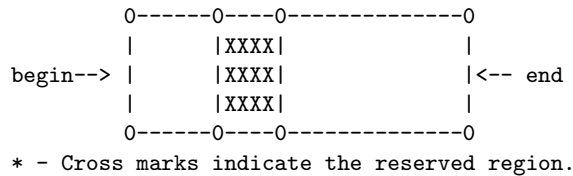


Figure 3: Memory layout of the first kernel - post reservation

Kexec requires that the kernel that needs to be re-booted into be pre-loaded into memory. This is done through a user-space utility called kexec-tools [02]. In addition to the normal command-line parameters that need to be passed to kexec while loading, one additional parameter needs to be passed in the case of loading a kernel for crash dumping. This parameter is a string called “dump” and indicates to the second kernel that a crash has occurred in the previous kernel leading to the reboot. This parameter is also useful while exporting the dump contents as a file. [05] has details on how to pre-load a kexec kernel and information on the commands and syntax.

The entire dumping operation, involving the reboot and subsequent write out of the dump file, has to happen as quickly as possible. With this in mind, it is a good idea to ensure that the second kernel used is a small, stripped down kernel having minimal drivers and services. Passing command-line parameters to the second kernel to make it boot to a runlevel of “1” is also a good way of speeding up the entire process.

The dumping operation starts when a panic occurs and control is transferred to the kernel’s panic handler. The panic handler sets a flag to indicate that the system has crashed and transfers control to kexec.

Among the many things that the kexec code does, two of them are of interest to us. In fact, these two operations result from hooks placed by the crash dump code in kexec. The first of these operations is to snapshot the register values of all processors and halt the processors. This is achieved when the processor running the kexec code sends an NMI (Non-Maskeable Interrupt) to all the other processors that are currently on-line in the system. Before sending

the NMI, though, the NMI handler is changed from the default function to a crash dump-specific routine. Every processor, upon receiving the NMI, runs the crash dump-specific NMI handler. In the NMI handler, every processor saves the contents of the registers on to a reserved location. This reserved location is essentially one additional page of memory at the end of the backup region that has been previously allocated. Once the registers have been snapshotted, the processor “halts” itself.

The second important operation performed by the kexec code happens next. The processor running the kexec code waits for all other processors to halt and then proceeds to back up the region of memory that the second kernel will use. If, for example, the second kernel uses 16MB of memory, then the first 16MB of memory is copied over to the backup region prior to reboot. Figure 4 shows what the contents of memory, after the backup, look like.

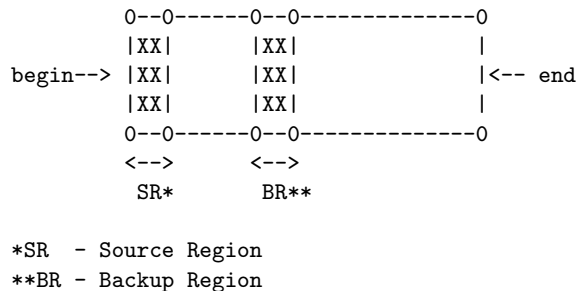


Figure 4: Memory layout of the first kernel just before reboot

Next, kexec performs its magic and the system boots into the second kernel.

4.2 Accessing the dump

This section talks about the mechanisms employed, and the interfaces provided, to write out the dump in the second kernel. The current implementation makes available two mechanisms for accessing the dump. The first one is through the /proc interface in the form of an ELF format core file. The second mechanism provides a linear (raw) view of the dump and can be accessed through /dev/oldmem.

Section 4.3 talks about the ELF file format interface to access the dump file. Section 4.4 details how the dump can be read using the `/dev/oldmem` interface.

4.3 The ELF file view

One of the main design goals of this project was to ensure that the users of the solution were provided a simple and easy-to-use method for writing out the dump. Several of the other dumping solutions discussed earlier suffer from the requirement that a separate user space utility needs to be installed, configured and run on the system to extract the dump file. Also, most of them require that a separate dump “device” be reserved for the purpose of collecting the dump. This device could be a disk partition on the system or a server machine on the network. Another restriction that existing solutions placed on the user was that the dump had to be written out every time, before any analysis could be performed on it. Many a time a developer or a service engineer may need to perform very minimal inspection of the dump contents before moving on to the next iteration of the problem identification cycle. In such circumstances, copying out the dump to a destination device is a burden.

These conditions are an irritant on the part of a user and it is obvious that removing these restrictions would be a welcome feature in the new solution. It was realized that abstracting the dump contents in the form of an ELF format core file addressed most of these concerns.

We will discuss the implementation techniques of this approach now and defer expanding on the advantages of the scheme to section 6.

The process of creating the ELF file starts during the parsing of the kernel command line, very early during the booting of the second kernel. If the string “dump” is found in the command line, a flag, called `dump_enabled` is set indicating that the system has rebooted due to a crash. Next, during the initialization of the `proc` filesystem, a new `proc` entry called “`vmcore`” is created if the `dump_enabled` flag has been set. This entry is a binary file and its size is set to the size of the memory corresponding to the first kernel. The size of the first kernel’s

memory, which is the entire memory that the system has, is obtained through low-level architecture-specific function calls. The file operations structure corresponding to the `vmcore` entry also provides a “read” function. This read function is the interface that applications can use to read the dump contents. Once the above initialization is done, the dump file appears as `/proc/vmcore` in the system.

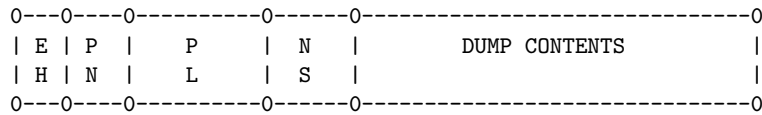
The actual reading of the dump file, and related processing, starts only when a user process requests a read. Such a request could be, for example, a `cp`, `sep` or `ftp` command doing a normal file read operation on the `vmcore` file. The `vmcore` file-specific read function, known as `read_vmcore`, lays out the file format as shown in figure 5.

The various sections of this file are:

1. The ELF Header - This is the main header for the file providing basic file information.
2. The `PT_NOTE` Program Header - This is again a basic structure pointing to the list of notes sections that the file contains.
3. The `PT_LOAD` Program Header - This is a series of program headers describing each range of memory that the file maps. In the current implementation, there exists just one program header describing the entire range of memory.
4. The notes sections - This area contains auxiliary information such as register contents and process information.
5. The data area - This area contains the actual memory contents from the previous kernel.

The read function essentially consists of two parts. The first one handles creating the ELF headers on the fly. The second part deals with reading the actual dump contents from the preserved memory. Depending upon the offset into the file that has been requested, relevant portions of the file, in other words, either the headers, or memory, or both, are read and returned to the user.

The most interesting part of this entire operation is that of reading the previous kernel’s memory. Because the second kernel has booted with very little



EH - ELF Header
 PN - PT_NOTE Program Header
 PL - PT_LOAD Program Header
 NS - Notes Sections

Figure 5: Format of the ELF dump file

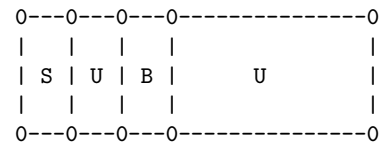
memory, the previous kernel's memory cannot be accessed directly. Access is obtained by treating this unavailable region of memory as a high memory device. Access to this region is handled on a page-by-page basis. Because the page in question does not have any "page struct" associated with it, some special techniques are used to obtain access. The steps associated with this operation are enumerated below.

1. Calculate the page-frame number corresponding to the physical address of the memory page requested.
2. Allocate a temporary page of memory to copy the dump contents.
3. Fix up a virtual address for this page.
4. Create a page table entry corresponding to this address.
5. Copy the contents of the dump page to the temporary page.
6. Unmap the dump page. Erase the page table entry that was created previously.
7. Copy the contents of the temporary page on to the user supplied buffer.
8. Free the temporary page.

The function implementing the steps is used to read in all the dump pages that have been requested by the user.

One special consideration to keep in mind while reading the dump pages is the re-ordering of the pages

that is needed to view the dump contents correctly. Because the first few megabytes of the previous kernel's memory are backed up into the reserved area, the layout of the dump is not in the correct order. Figure 6 shows the layout of the dump in terms of physical memory.



S - Memory corresponding to the second kernel.
 U - Untouched memory from the previous kernel.
 B - Backup of memory from the previous kernel, overwritten by region S.

Figure 6: Physical memory layout of the dump

Referring to the regions described in figure 6, region S corresponds to the second kernel and is not part of the dump. Region B contains the memory that was originally located in region S. Therefore, whenever a read request asks for dump contents corresponding to region S, the read function reads the corresponding page from region B. When a request asks for memory from region B itself, a zero-filled buffer is returned. No change in the logic is required for read operations corresponding to region U.

Another important piece of information that the dump read logic code provides is the register values of the various processors at the time of failure. The ELF

format allows auxiliary information to be provided in the form of “notes” sections within the header. The read function, while creating the dump file header, reads in the register values from the backup region and creates a notes section for this purpose. Tools which understand the ELF format interpret the contents of the notes section as register values, if the notes header indicates so, using a special flag known as NT_PRPSINFO. The same flag is used by the dump read logic while storing the register values in the header. Note that, currently, register values corresponding to processor 0 only are being read. Work is underway to enable reading of the register contents of all processors.

Thus the ELF format interface provides an easy mechanism for reading out the contents of the dump, in the correct order, while also providing register contents at the time of failure.

4.4 Linear view of the dump

An alternate interface provided by the tool enables a user to obtain a linear or raw view of the dump. This method provides a simple, unformatted view and allows selective portions of the dump to be read. This method provides an interface that is equivalent to the `/dev/mem` device interface that exists in the kernel for analyzing live memory.

The first step towards this is by registering the device number for this interface. Minor number 12 under the category of character devices has been reserved for this purpose. The device is called as “oldmem” and can be accessed by creating a char device with major number 1 and minor number 12 using the `mknod` command. The `file_operations` structure for this particular device is as shown in figure 7.

```
static struct file_operations oldmem_fops = {
    .read = read_oldmem,
    .open = open_oldmem,
};
```

Figure 7: File operations structure for the oldmem device

While the `open_oldmem` function just performs a permissions check on the user to ensure that only privileged users are allowed access to the dump, the `read_oldmem` routine is what does the job of reading out the dump pages. The actions performed by this routine are very similar to the corresponding dump page reading routine in the ELF format interface. Here too, the logic is to map a page of the previous kernel’s memory into a temporary page-table entry and then copy out the contents to user space. Again, the code contains logic to correctly re-order the contents of memory depending upon the offset requested, to ensure that the relocation of the initial chunk of memory into the backup region is taken care of.

5 Illustration

In this section, through a step-by-step approach, we explain how to install and set up the new feature and obtain a dump.

1. Obtain the latest kexec patches [02] and the kexec-based crash dump patches [11]. Apply the kexec patch followed by the crash dump patches.
2. Choose the “kernel crash dumps” option under the “Processor type and features” section and the “kexec system call” option under “Kernel Hacking”. Build and boot to the new kernel.
3. Install the kexec-tools user-space utility available for download from the same location as kexec.
4. The second kernel that is booted into needs to be pre-loaded. The following command will do the loading.

```
kexec -l <kernel-image> -
append="root=<root-dev> mem=16M dump
init 1"
```

Note that `<kernel-image>` has to be replaced with the name of the kernel image that is to be booted into and `<root-dev>` is to be replaced with the correct root device.

5. When a panic occurs, the system reboots into the second kernel.

6. Write out the dump file using

```
cp /proc/vmcore <dump-file>
```
7. In order to access the dump as a device for a linear/raw view, the device needs to be created first. This can be done with

```
mknod /dev/oldmem c 1 12
```
8. To write out specific portions of the dump, use the “dd” command with suitable options for the count, bs and skip command-line arguments.
9. To perform analysis on the dump file using gdb, a “vmlinux” image built with the -g option is required. Once this is in place, type the following command to open the dump:

```
gdb vmlinux <dump-file>
```

Replace <dump-file> with the actual name of the dump file.

6 Advantages and Limitations

A detailed discussion of the advantages and limitations of this approach are discussed here. This will help evaluate how much the advantages outweigh the limitations as this comparison is the primary determinant of the effectiveness of the solution. In section 6.1, we discuss some of the benefits that the kexec based crash dumping solution offers. The limitations of this solution are discussed in section 6.2.

6.1 Advantages

One of the primary advantages that this approach brings is that the dependency on the failed kernel is significantly reduced. Most dumping solutions perform a considerable amount of processing in the context of the failed kernel. This includes writing out the dump to a device in most cases. Such operations are completely unreliable. The kexec-based crash dump implementation defers most of the processing to the second kernel and hence may deliver more reliability. The only significant operation performed in the first kernel is that of copying a chunk of memory into

the backup region. No device or hardware-dependent operation is performed.

The design is not bound by resource restrictions to perform dumping. Because the task of obtaining a dump is reduced to a reboot, complete dumps are available in all cases. Unlike other approaches, this design does not depend upon things like the memory available in the system, presence of a dump partition or dump server, to perform the dump. This greatly increases the chances of obtaining a reliable dump.

Because the main operation involved here is a reboot, the crash dump code is fairly simple in nature. It is also very minimally invasive into the existing kernel code and does not add too much overhead. The entire code for this tool comes up to a few hundred lines of code. This makes it extremely lightweight compared to other approaches.

This approach does not require the presence of a dedicated dump “device” to collect the dump. Therefore, users need not reserve a separate disk partition or a network server in order to collect the dump. The dump is written out through the normal file system route in the second kernel.

The technique of abstracting the dump as an ELF format file provides significant benefits. Writing out the dump is a very easy operation. Ordinary file copy commands such as cp, scp or ftp can be used to write out the dump file. The user is spared from having to install, configure and run a separate user-space tool to write out the dump. The tool provides the user with the option of writing out the dump to either a disk or across the network without additional setup (by choosing the appropriate file copy utility).

This approach has an added advantage when it comes to transferring dumps across the network. Since copying the dump is a simple file copy operation, standard protocols can be used for transferring dumps on to another machine. Other solutions require a special catcher program to be running on the server where the dump is collected. For example, the Red Hat netdump [13] tool requires that a “netdump-server” be running on the target machine. The kexec-based approach obviates the need for a dedicated machine. The ability to use standard file transfer protocols has an added advantage in that it becomes possible to configure support for dump col-

lection even across a firewall.

Writing out the dump in the form of an ELF file means that existing dump analysis tools such as gdb or crash can be used to perform the investigation. Unlike other tools, this does not come with its own dump analysis package. Service engineers are spared from having to learn (yet) another dump analysis tool.

The /proc interface to the dump file has a major use for kernel developers. Often a kernel developer needs to make only a few observations from the dump file in order to proceed to the next stage of problem determination. This may include things like stack analysis of the running process or examining a certain region of memory. Having to write out the entire dump (which may run into many GBs) is a complete waste. With the current approach, analysis can be performed straightaway on the dump, through the /proc interface², without having to write out the dump. Developers can run gdb on the /proc dump file, perform the analysis, reboot the machine and move on.

6.2 Limitations

As with any dumping solution, the current approach also has a few limitations. A few of them are discussed here.

Booting the second kernel with very little memory means that an additional reboot, once dump write out is complete, is required to resume normal system operation. If a small, stripped down kernel is not used for the kexec boot, there is a chance of increasing the downtime of the system.

Booting the system with very little memory may create its own problems. The solution needs to be tested on different kinds of hardware to establish the appropriate limit on this size.

Because a reboot is involved every time, it is not possible to obtain non-disruptive dumps with this approach. If a process or thread-level dump needs to be collected, this approach is not suitable.

The requirement of a reserved backup area in memory may be a problem in embedded environments

²Memory constraints in the second kernel may limit the nature and extent of analysis that can be performed

that may be heavily memory constrained.

7 Status and TODOS

The kexec-based crash dump solution is in the proof of concept stage at the time of writing this paper. An initial set of patches, demonstrating the feasibility of this approach, have been released to the Linux kernel mailing list. Feedback and review comments that have been received have been positive.

The current implementation is limited to the Intel 32-bit platforms, primarily because kexec itself is available only for this platform. With the current patches, basic dump analysis can be performed on the dump using gdb. Stack tracing of the task running on processor 0 is possible. Examining memory contents is working fine. Register contents of processor 0 is possible.

In terms of TODOS, there are several enhancements that are in the pipeline. These improvements will make the kexec based crash dumping tool a complete solution. Some of the items on the todo list are as follows

1. The ability to write out selective portions of the dump. Practically, this implies the ability to selectively write out only the kernel pages from the system.
2. Support for handling NUMA machines and discontinuous memory. The dump reading logic needs to become aware of memory holes and handle them appropriately.
3. Shutting down all devices safely before rebooting the system into the dumping kernel. This requirement is a must for reliable dumps. Issues like on-going DMA rolling over to the second kernel, and corrupting the dump, need to be addressed. Discussions are on currently, in the relevant mailing lists, about the best way to handle these issues.

One of the approaches being considered is to make the second kernel boot from a non-default location. For example, on 1386 machines, the

second kernel can be made to boot from an offset of 16MB instead of the default 1MB location. This alternate location is reserved in the first kernel which immunizes the region from any memory or DMA use. This way, the second kernel is safe from any side effects of DMA rolling over. The design will be improved based on the outcome of these discussions to make the solution robust and reliable.

4. Changes to the crash [10] dump analyzing utility to recognize this file format and perform analysis on it.
5. Porting of this tools to other architectures like x86_64 and ppc64.

8 Conclusions

Kexec-based crash dumping is a promising new approach to reliable first failure data capture. It aims to provide reliable yet lightweight crash dumping capability to the Linux kernel. It also takes into consideration the need for a simple and easy-to-use mechanism for setup, configuration and dump write out.

The first set of patches has been received positively by the Linux community. Work is underway to make this tool more comprehensive and available on all architectures.

9 Acknowledgements

The authors wish to express their sincere thanks to Andrew Morton, Dave Hansen, Greg Kroah-Hartman and Eric Biederman for the numerous suggestions, review comments and feedback. Thanks to all the others who have helped us in our efforts.

References

[01] Linux Kernel Crash Dumps, <http://lkcd.sourceforge.net/>

[02] The latest kexec patches, <http://www.xmission.com/ebierm/files/kexec/>

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

[04] The latest kexec-tools patches, <http://developer.osdl.org/rddunlap/kexec/kexec-tools/>

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

[06] Two Kernel Monte (Linux loading Linux on x86), <http://www.scyld.com/software/monte.html>

[07] bootimg (Boot kernel Image), <http://bootimg.sourceforge.net/>

[08] Mission Critical Linux - In Memory Core Dump, <http://oss.missioncriticallinux.com/projects/mcore/>

[09] Ron Minnich, *LOBOS:(Linux OS Boots OS) Booting a kernel in 32-bit mode*, Proceedings of the 4th Annual Linux Showcase & Conference, Atlanta, Georgia, USA, October 10-14, 2000.

[10] The crash dump analyzing utility, <http://people.redhat.com/anderson/>

[11] The kexec based crash dump patches, <http://www.ussg.iu.edu/hypermil/linux/kernel/0408.2/0451.html>

[12] Diskdump, <http://sourceforge.net/projects/lkdump>

[13] Red Hat netdump <http://www.redhat.com/support/pers/redhat/netdump/>

10 Legal Statement and Trademarks

This work represents the views of the authors and does not necessarily represent the view of IBM.

IBM, AIX, OS/2 and OS/390 are registered trademarks of International Business Machines in the United States, other countries or both.

Intel is a trademark of Intel Corporation in the United States, other countries, or both.

Red Hat is a trademark of Red Hat, Inc. in the United States, other countries, or both.

Linux is a registered trademark of Linus Torvalds.

Two Kernel Monte is a trademark of Scyld Computing Corporation.

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