KDUMP AND INTRODUCTION TO VMORE ANALYSIS

HOW TO GET STARTED WITH INSPECTING KERNEL FAILURES

PATRICK LADD
TECHNICAL ACCOUNT MANAGER, RED HAT
pladd@redhat.com

slides available at https://people.redhat.com/pladd
SUMMARY

TOPICS TO BE COVERED

- WHAT IS A VMCORE, AND WHEN IS IT CAPTURED?

- CONFIGURATION OF THE KDUMP UTILITY AND TESTING

- SETTING UP A VMCORE ANALYSIS SYSTEM

- USING THE "CRASH" UTILITY FOR INITIAL ANALYSIS OF VMCORE CONTENTS
WHAT IS A VMCORE?

It is the contents of system RAM. Ordinarily, captured via:

- makedumpfile
- VMWare suspend files
- QEMU suspend-to-disk images

```
# hexdump -C vmcore -s 0x00011d8 -n 200
000011d8 56 4d 43 4f 52 45 49 46 00 00 00 4f 53 52 45 |VMCOREINFO..OSRE|
000011e8 00 00 00 4f 53 52 45 |LEASE=2.6.32-573|
000011f8 00 00 00 2e 64 65 6f 64 5f 60 00 00 00 60 00 0a |.pages=4096.S|
00001208 00 00 00 2e 67 64 65 5f 70 67 5f 64 |BOL(swapper_pg_d|
00001218 00 00 00 2d 35 37 33 |SYMBOL(node_|
00001228 00 00 00 38 31 63 31 61 36 63 30 |online_map)=ffff|
00001238 00 00 00 66 66 66 66 38 31 61 38 |ir)=ffffffff81a8|
00001248 00 00 00 42 4f 4c 28 73 77 61 70 70 65 72 |BOL(swapper_pg_d|
00001258 00 00 00 29 3d 66 66 66 66 66 66 66 66 66 66 |ir)=ffffffff81a8|
00001268 00 00 00 42 4f 4c 28 73 77 61 70 70 65 72 |BOL(swapper_pg_d|
```
WHEN IS ONE WRITTEN?

By default, when the kernel encounters a state in which it cannot gracefully continue execution. Automatic captures can be expected on a configured system following:

- Kernel code level issues including
  - Memory corruption
  - Use-after-free conditions
  - Invalid memory accesses
- Machine Check exceptions

The kernel can be configured to also initiate a panic condition under a specific event condition. These can be found in the sysctl tunables on a system with "panic" in the title.
# sysctl -A | grep panic
kernel.panic = 0
kernel.panic_on_oops = 1
kernel.softlockup_panic = 0
kernel.unknown_nmi_panic = 0
kernel.panic_on_unrecovered_nmi = 0
kernel.panic_on_io_nmi = 0
kernel.hung_task_panic = 0
kernel.panic_on_warn = 0
vm.panic_on_oom = 0
KDUMP CONFIGURATION

The kdump service is the primary mechanism configuration component that allows systems to capture a vmcore during a failure.

- Provided by the "kexec-tools" package
- Configuration located in /etc/kdump.conf and most options are documented within the configuration file.
#raw /dev/sda5
#ext4 /dev/sda3
#ext4 LABEL=/boot
#ext4 UUID=03138356-5e61-4ab3-b58e-27507ac41937
#net my.server.com:/export/tmp
#net user@my.server.com
path /var/crash
core_collector makedumpfile -c --message-level 1 -d 31
#core_collector scp
#core_collector cp --sparse=always
#extra_bins /bin/cp
#link_delay 60
#kdump_post /var/crash/scripts/kdump-post.sh
#extra_bins /usr/bin/lftp
#disk_timeout 30
#extra_modules gfs2
#options modulename options
#default shell
#debug_mem_level 0
#force_rebuild 1
#sshkey /root/.ssh/kdump_id_rsa
#fence_kdump_args -p 7410 -f auto -c 0 -i 10
#fence_kdump_nodes node1 node2
Required additional configuration:

- The "crashkernel" option.
  - This is applied within the bootloader configuration file and determines how much RAM is used for the secondary kernel which is responsible for writing the contents of RAM for the primary failing kernel.
  - Should be sized appropriately to the amount of RAM installed to the system, with an eye to how many paths and modules are loaded on the system to establish a base level of functionality.
For example:

```bash
# awk '/^title/,/EOF/' /boot/grub/grub.conf

title Red Hat Enterprise Linux Server (2.6.32-573.22.1.el6.x86_64)
root (hd0,0)
kerneld /vmlinuz-2.6.32-573.22.1.el6.x86_64 crashkernel=128M ro
root=/dev/mapper/vg_unused-lv_root
initrd /initramfs-2.6.32-573.22.1.el6.x86_64.img
```
FURTHER ASSISTANCE:

- How to troubleshoot kernel crashes, hangs, or reboots with kdump on Red Hat Enterprise Linux
- https://access.redhat.com/solutions/6038

How to troubleshoot kernel crashes, hangs, or reboots with kdump on Red Hat Enterprise Linux

Environment

- Red Hat Enterprise Linux 5 [Below steps are not applicable for s390 or z/VM RHEL 5 instances]
- Red Hat Enterprise Linux 6 [RHEL 6 update 3 is required for s390 or z/VM RHEL instances]
- Red Hat Enterprise Linux 7

Issue

- How do I configure kexec/kdump on RHEL?
- How much disk space is required for kdump to dump the vmcore?
- Need RCA of kernel crash/panic
- How do I troubleshoot a server crash/reboot?
- Want root cause of a system reboot
- How do I capture a vmcore on my server?
- My server hung or became unresponsive, how to troubleshoot?
- Problem collecting a core file with kdump on a host
- How much time is required to capture vmcore?
- System freezes unexpectedly, how to troubleshoot?
FURTHER ASSISTANCE:

- In addition, the following lab app guides administrators through the process of configuration dynamically based on the individual end system.
  - Kdump Helper
  - [https://access.redhat.com/labs/kdumphelper/](https://access.redhat.com/labs/kdumphelper/)
KDUMP TESTING

Though the kdump service and configuration is fairly straightforward, the possible sources of failure are numerous. It is paramount to test the configuration. By initiating a kernel panic and verifying that a vmcore is present afterwards, confidence is raised that the service will function at a later date.

root@example ~]# echo c > /proc/sysrq-trigger
KDUMP TESTING

Which is closely followed by a backtrace and what looks like a system boot iteration.

```
Saving vmcore-dmesg.txt
Saved vmcore-dmesg.txt
Copying data : [100.0 %]
Excluding unnecessary pages : [100.0 %]
Copying data : [ 100.0 %]
Saving core complete
Restarting system.
machine restart
```

Failures to write a vmcore can be addressed here as opposed to following an outage event.
ANALYSIS SYSTEM

For any system that will review vmcore contents, the following must be installed:

1️⃣ The crash utility, available within the rhel-6-server-rpms base channel

2️⃣ Debugging symbols contained within the "vmlinux" file from the debuginfo package associated with the vmcore needing to be analyzed.
[root@example ~]# yum install crash --disablerepo=\* --enablerepo=rhel-6-server-rpms
Loaded plugins: product-id, security, subscription-manager
Setting up Install Process
Resolving Dependencies
<snip>
Dependencies Resolved

=================================================================================================================================================================================================================================================================================================================================
<table>
<thead>
<tr>
<th>Package</th>
<th>Arch</th>
<th>Version</th>
<th>Repository</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Installing:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>crash</td>
<td>x86_64</td>
<td>7.1.0-3.el6_7.1</td>
<td>rhel-6-server-rpms</td>
<td>2.5 M</td>
</tr>
</tbody>
</table>

Transaction Summary
=================================================================================================================================================================================================================================================================================================================================
| Install  | 1 Package(s) |

Total download size: 2.5 M
Installed size: 7.0 M
Is this ok [y/N]: y
```
[root@example ~]# yum install kernel-debuginfo-`uname -r` --disablerepo=* \
--enablerepo=rhel-6-server-debug-rpms
Loaded plugins: product-id, security, subscription-manager
Setting up Install Process
```
```
rhel-6-server-debug-rpms                          |  2.9 kB  00:00
rhel-6-server-debug-rpms/primary_db              | 1.8 MB  00:00
```
Resolving Dependencies
<snip>
Dependencies Resolved

```
Transaction Summary
```
```
Install  2 Package(s)
```
```
Total size: 309 M
Total download size: 43 M
Installed size: 1.7 G
Is this ok [y/N]: y
```
Though the above example shows the simplest method for achieving a working vmcore analysis system, the full kernel-debuginfo is not necessary. The core symbols are contained within the "vmlinux" file within the package.

```
# rpm -qlp kernel-debuginfo-2.6.32-573.8.1.el6.x86_64.rpm | grep 'vmlinux' -C 5
/usr/lib/debug/lib/modules/2.6.32-573.8.1.el6.x86_64/vdso
/usr/lib/debug/lib/modules/2.6.32-573.8.1.el6.x86_64/vdso/vdso.so.debug
/usr/lib/debug/lib/modules/2.6.32-573.8.1.el6.x86_64/vdso/vdso32-int80.so.debug
/usr/lib/debug/lib/modules/2.6.32-573.8.1.el6.x86_64/vdso/vdso32-syscall.so.debug
/usr/lib/debug/lib/modules/2.6.32-573.8.1.el6.x86_64/vdso/vdso32-sysenter.so.debug
/usr/lib/debug/lib/modules/2.6.32-573.8.1.el6.x86_64/vmlinux
/usr/lib/debug/usr
```
Extracting this file for direct use can be achieved via the "rpm2cpio" and "cpio" command combination below:

```bash
# TEMPDIR=$(mktemp -d)
# cd $TEMPDIR
# rpm2cpio /tmp/kernel-debuginfo-2.6.32-573.8.1.el6.x86_64.rpm | \
cpio -idmv ./usr/lib/debug/lib/modules/2.6.32-573.8.1.el6.x86_64/vmlinux
./usr/lib/debug/lib/modules/2.6.32-573.8.1.el6.x86_64/vmlinux
3252733 blocks
```
ANALYSIS SYSTEM

OTHER RESOURCES AVAILABLE:

Both of the projects below allow the process of downloading separate debuginfo packages and maintaining separate directory structures for each individual vmcore to be entirely automated. Both provide a web interface that can be used to queue vmcores for analysis on local infrastructure.

- **Retrace Server**
  - https://fedoraproject.org/wiki/Features/RetraceServer
  - Well maintained with updates on regular interval

- **Core Analysis System**
  - https://fedorahosted.org/cas/
  - Not maintained at this time, the last commits to the upstream project were in 2010

- **redhat-support-tool**
  - https://access.redhat.com/articles/445443#btextract
  - This simple tool is available for retrieving necessary debuginfo packages.
  - It requires a similar release system such as Red Hat Enterprise Linux 6 for vmcores captured on a Red Hat Enterprise Linux 6 system.
INITIAL ANALYSIS

NOW THAT A VMCORE HAS BEEN CAPTURED, WHAT NEXT?

The initial steps within a vmcore can be distilled down to the following few commands:

1. log
2. bt
3. kmem -i

Each of these commands help narrow the focus of what could have caused a particular outage and will be discussed in the following slides.
INITIAL ANALYSIS

Crash itself is a debugger utility that acts as an interactive shell, accessing the contents of a given vmcore in a human readable format using the debugging symbols provided.

Opening the utility simply requires the vmcore and vmlinux file.
INITIAL ANALYSIS - LOG

The log command returns the contents of the failed kernels log_buf array. When a message is emitted within the kernel via the printk() function call, the string is put into the this array in the correct entry.

For any kernel panic event, the cause of the condition will generally be noted at the end of the output.

BUG: unable to handle kernel NULL pointer dereference at (null)
IP: [fffffffff81350f46] sysrq_handle_crash+0x16/0x20
PGD 21d711067 PUD 21d520067 PMD 0
Oops: 0002 [#1] SMP
last sysfs file: /sys/devices/pci0000:00/0000:00:00:05.7/usb1/1-1/speed
CPU 0
Modules linked in: onload(U) sfc_char(U) sfc_resource(U) sfc_affinity(U) <snip>

Pid: 2932, comm: bash Not tainted 2.6.32-573.8.1.el6.x86_64 #1 Bochs Bochs
RIP: 0010:[<ffffffff81350f46>] [<ffffffff81350f46>] sysrq_handle_crash+0x16/0x20
RSP: 0018:fffffff88021da87e18 EFLAGS: 00010096
RAX: 0000000000000010 RBX: 0000000000000063 RCX: 0000000000000000
RDX: 0000000000000000 RSI: 0000000000000000 RDI: 0000000000000063
RBP: ffffffff88021da87e18 R08: 0000000000000000 R09: 0000000000000077
R10: 00000000ffffffff R11: 00000000000000246 R12: 0000000000000000
R13: ffffffff81b109a0 R14: 00000000000000286 R15: 0000000000000004
FS: 00007fed7d8f5700(0000) GS:fffffff88002820000(0000) knlGS:0000000000000000
CS: 0010 DS: 0000 ES: 0000 CR0: 0000000000050003
CR2: 0000000000000000 CR3: 0000000000000021db04000 CR4: 0000000000000406f0
DR0: 0000000000000000 DR1: 0000000000000000 DR2: 0000000000000000
DR3: 0000000000000000 DR6: 00000000fff0ff0 DR7: 0000000000000400
<snip>
INITIAL ANALYSIS - LOG

Continued.

Process bash (pid: 2932, threadinfo ffff88021da84000, task ffff88021d450ab0)
Stack:
  ffff88021da87e68 ffffffff81351202 ffff88021ad58728 0000000000000000
  <d> 0000000000000022 0000000000000022 ffff88011af9c00 00007fed7d8fc000
  <d> 0000000000000002 ffffffffffffffff ffff88021da87e98 ffffffff813512be
Call Trace:
  [<ffffffff81351202>] __handle_sysrq+0x132/0x1a0
  [<ffffffff813512be>] write_sysrq_trigger+0x4e/0x50
  [<ffffffff811fd29e>] proc_reg_write+0x7e/0xc0
  [<ffffffff81191ab8>] vfs_write+0xb8/0x1a0
  [<ffffffff81192fa6>] ? fget_light_pos+0x16/0x50
  [<ffffffff811925f1>] sys_write+0x51/0xb0
  [<ffffffff8100b0d2>] system_call_fastpath+0x16/0x1b
Code: d0 88 81 a3 6e ff 81 c9 c3 66 66 66 2e 0f 1f 84 00 00 00 00 00 00 56 48 <snip>
RIP [ <ffffffff81350f46>] sysrq_handle_crash+0x16/0x20
RSP <fff888021da87e18>
CR2: 0000000000000000

In the event that a vmcore was captured not as a result of a kernel panic, careful inspection of the full ring buffer contents will be necessary. A system may enter a degraded state, but continue operation for some time without encountering a panic state. This, if continued long enough can result in the ring buffer wrapping and thereby masking the original initiating failure condition.
INITIAL ANALYSIS - LOG

In the above example, the following message is reported:

Followed by a register dump, and backtrace. Register contents have been omitted from the following for legibility:

BUG: unable to handle kernel NULL pointer dereference at (null)

Pid: 2932, comm: bash Not tainted 2.6.32-573.8.1.el6.x86_64 #1 Bochs Bochs
RIP: 0010:[<ffffffff81350f46>] [<ffffffff81350f46>] sysrq_handle_crash+0x16/0x20
<snip>
Call Trace:
[<ffffffff81351202>] __handle_sysrq+0x132/0x1a0
[<ffffffff813512be>] write_sysrq_trigger+0x4e/0x50
[<ffffffff811fd29e>] proc_reg_write+0x7e/0xc0
[<ffffffff81191ab8>] vfs_write+0x51/0xb0
[<ffffffff81192fa6>] ? fget_light_pos+0x16/0x50
[<ffffffff811925f1>] sys_write+0x51/0xb0
[<ffffffff8100b0d2>] system_call_fastpath+0x16/0x1b
Code: d0 88 81 a3 6e ff 81 c9 c3 66 66 66 2e 0f 1f 84 00 00 00 00 00 00 55 48 89 e5 0f 1f 44 00
RIP [<ffffffff81350f46>] sysrq_handle_crash+0x16/0x20
RSP <ffffff88021da87e18>
CR2: 0000000000000000

RIP: Is the instruction that was underway at the time of the failure.

Call Trace: Is the breadcrumb trail of functions that led to this end RIP. Functions with a "?" next to them are a guess by the stack unwinder and should not be assumed to be correct without further verification.
INITIAL ANALYSIS - LOG

Using the above backtrace, the RIP yields the exact operation that was underway at the time of the failure.

Kernel crash in sysrq_handle_crash

Resolution

- Determine what issued the sysrq+c and if desired, work to stop the source of the sysrq+c command.

Root Cause

- The box has been panicked with a sysrq+c instruction. This would have to be manually initiated.
- Please see this documentation for more information on the SysRq facility.

Duplicate(s) Info

- What is the source of these Call Traces in /var/log/messages
INITIAL ANALYSIS - BT

Similar to the log command, the bt returns (by default) the backtrace of the task that is currently under inspection. As the active task on opening is the process that is the panicking process, this yields the following on the same vmcore.

```
crash> bt
<snip>
[exception RIP: sysrq_handle_crash+22]
RIP: fffffff81350f46  RSP: fffffff88021da87e18  RFLAGS: 00010096
RAX: 0000000000000000  RBX: 0000000000000063  RCX: 0000000000000000
RDX: 0000000000000000  RSI: 0000000000000000  RDI: 0000000000000063
RBP: fffffff88021da87e18  R8: 0000000000000000  R9: 00007fed7d8f5700
R10: 00000000ffffff  R11: 0000000000000246  R12: 0000000000000000
R13: fffffff81b109a0  R14: 0000000000000286  R15: 0000000000000004
ORIG_RAX: fffffff81b109a0  CS: 0010  SS: 0018
#9 [ffffff88021da87e20] __handle_sysrq at fffffff81351202
#10 [ffffff88021da87e70] write_sysrq_trigger at fffffff813512be
#11 [ffffff88021da87ea0] proc_reg_write at fffffff811fd29e
#12 [ffffff88021da87ef0] vfs_write at fffffff81191ab8
#13 [ffffff88021da87f30] sys_write at fffffff811925f1
#14 [ffffff88021da87f80] system_call_fastpath at fffffff8100b0d2
RIP: 0000000374aedb520  RSP: 00007fffc06b2c410  RFLAGS: 00010206
RAX: 0000000000000000  RBX: fffffff8100b0d2  RCX: 0000000000000400
RDX: 0000000000000000  RSI: 00007fed7d8fc000  RDI: 0000000000000001
RBP: 00007fed7d8fc000  R8: 0000000000000000a  R9: 00007fed7d8f5700
R10: 00000000ffffff  R11: 0000000000000246  R12: 0000000000000002
R13: 0000000374b18e7a0  R14: 0000000000000002  R15: 0000000374b18e7a0
ORIG RAX: 0000000000000000  CS: 0033  SS: 002b
```
INITIAL ANALYSIS - BT

For initial analysis purposes, the bt command also allows backtraces to be returned for all active tasks in the vmcore via the "-a" flag:

```
crash> bt -a | grep PID -A 1
PID: 2932 TASK: ffff88021d450ab0 CPU: 0 COMMAND: "bash"
  #0 [fff88021da879e0] machine_kexec at ffffffff8103d1ab
--
PID: 0 TASK: ffff88011e679520 CPU: 1 COMMAND: "swapper"
  #0 [fff880028246e90] crash_nmi_callback at ffffffff81033cf6
--
PID: 0 TASK: ffff88011e688040 CPU: 2 COMMAND: "swapper"
  #0 [fff880028286e90] crash_nmi_callback at ffffffff81033cf6
--
PID: 0 TASK: ffff88011e6d0ab0 CPU: 3 COMMAND: "swapper"
  #0 [fff8800282c6e90] crash_nmi_callback at ffffffff81033cf6
--
PID: 0 TASK: ffff88011e6f5520 CPU: 4 COMMAND: "swapper"
  #0 [fff880028306e90] crash_nmi_callback at ffffffff81033cf6
--
PID: 0 TASK: ffff88011e704040 CPU: 5 COMMAND: "swapper"
  #0 [fff880028346e90] crash_nmi_callback at ffffffff81033cf6
--
PID: 0 TASK: ffff88011e74aab0 CPU: 6 COMMAND: "swapper"
  #0 [fff880123c06e90] crash_nmi_callback at ffffffff81033cf6
--
PID: 0 TASK: ffff88011e767520 CPU: 7 COMMAND: "swapper"
  #0 [fff880123c46e90] crash_nmi_callback at ffffffff81033cf6
```
INITIAL ANALYSIS - BT

Based on the RIP of the process at the time of the failure, mapping the failure to code is possible.

```sh
crash> bt | awk '/^#8/,/^#9/'
#8 [ffff88021da87d60] page_fault at ffffffff8153bf05
    [exception RIP: sysrq_handle_crash+22]
RIP: ffffffff81350f46  RSP: ffff88021da87e18  RFLAGS: 00010096
   RAX: 0000000000000010  RBX: 0000000000000063  RCX: 0000000000000000
   RDX: 0000000000000000  RSI: 0000000000000000  RDI: 0000000000000000
   RBP: ffff88021da87e18   R8: 0000000000000000  R9: 00007fed7d8f57
   R10: 00000000fffffff  R11: 0000000000000246  R12: 0000000000000000
   R13: ffffffff81b109a0  R14: 0000000000000286  R15: 0000000000000004

The "dis" function can be used to inspect the RIP using either the hexadecimal representation, or symbol+offset value.

```sh
crash> dis -l sysrq_handle_crash+22
<snip>/drivers/char/sysrq.c: 130
0xffffffff81350f46 <sysrq_handle_crash+22>: movb $0x1,0x0
```

Though the above path is truncated, the above dis operation yields the source in which the system was executing at the time of the failure.
INITIAL ANALYSIS - BT

With source file and line number, the Red Hat Code browser can be used to determine what operation was underway at the time.

Red Hat Code Browser

Within the code browser, navigating to the drivers/char/sysrq.c source file at line number 130 yields:

```c
static void sysrq_handle_crash(int key, struct tty_struct *tty)
{
    char *killer = NULL;

    panic_on_oops = 1; /* force panic */
    wmb();
    *killer = 1;
}
```

```c
static struct sysrq_key_op sysrq_crash_op = {
    .handler    = sysrq_handle_crash,
    .help msg   = "Crash",
```
INITIAL ANALYSIS - BT

In the event that the log output shows that there is a failure present in a task that is not in the end default bt command, using the bt command with the specific PID will allow a review of the current state of that task.

For hung_task_panic events, a careful review of the current state of the task is necessary as the panic task will be the "khungtaskd" process and not the actual task that has been within Uninterruptible sleep state for greater than 120 seconds.
The kmem command can be used to determine a wealth of information regarding the VMM subsystem. However, an overview of memory use can be reviewed via the "-i" flag:

```
crash> kmem -i

<table>
<thead>
<tr>
<th>PAGES</th>
<th>TOTAL</th>
<th>PERCENTAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOTAL MEM</td>
<td>2012387</td>
<td>7.7 GB</td>
</tr>
<tr>
<td>FREE</td>
<td>1911140</td>
<td>7.3 GB</td>
</tr>
<tr>
<td>USED</td>
<td>101247</td>
<td>395.5 MB</td>
</tr>
<tr>
<td>SHARED</td>
<td>12196</td>
<td>47.6 MB</td>
</tr>
<tr>
<td>BUFFERS</td>
<td>4964</td>
<td>19.4 MB</td>
</tr>
<tr>
<td>CACHED</td>
<td>38010</td>
<td>148.5 MB</td>
</tr>
<tr>
<td>SLAB</td>
<td>20287</td>
<td>79.2 MB</td>
</tr>
<tr>
<td>TOTAL SWAP</td>
<td>2047999</td>
<td>7.8 GB</td>
</tr>
<tr>
<td>SWAP USED</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>SWAP FREE</td>
<td>2047999</td>
<td>7.8 GB</td>
</tr>
<tr>
<td>COMMIT LIMIT</td>
<td>3054192</td>
<td>11.7 GB</td>
</tr>
</tbody>
</table>
```

Specific types of memory starvation can be evaluated further using the kmem utility in full.
INITIAL ANALYSIS - KMEM

"kmem -s" - Slab usage

```
crash> kmem -s | head
CACHE            NAME                 OBJSIZE  ALLOCATED  TOTAL  SLABS  SSIZE
ffff8011c1d1fc0 onloadfs_inode_cache 616 1 6 1 4k
ffff8011c3e1f80 nfs_direct_cache 200 0 0 0 4k
ffff8011c3d1f40 nfs_commit_data 704 0 0 0 8k
ffff8011c3c1f00 nfs_write_data 960 36 36 9 4k
ffff8011c3b1ec0 nfs_read_data 896 0 0 0 4k
ffff8011c3a1e80 nfs_inode_cache 1048 1 3 1 4k
ffff8011c391e40 nfs_page 128 0 0 0 4k
ffff8011cc51e00 fscache_cookie_jar 80 3 48 1 4k
```

"kmem -z" - Per zone statistics

```
crash> kmem -z | head

NODE: 0  ZONE: 0  ADDR: ffff80000015440  NAME: "DMA"
SIZE: 4095  PRESENT: 3830  MIN/LOW/HIGH: 41/51/61
VM_STAT:
    NR_FREE_PAGES: 3926
    NR_INACTIVE_ANON: 0
    NR_ACTIVE_ANON: 0
    NR_INACTIVE_FILE: 0
    NR_ACTIVE_FILE: 0
    NR_UNEVICTABLE: 0
    NR_MLOCK: 0
```
QUESTIONS?
THANK YOU!

slides available at https://people.redhat.com/pladd

linkedin.com/company/red-hat
facebook.com/redhatinc
youtube.com/redhat

twitter.com/RedHatNews
plus.google.com/+RedHat