System Crash Analysis

Imed Chihi <imed.chihi@gmail.com>
January 2009

Layout

- What's a crash? Why does it happen?
- Core dumps
- Analogy with userspace
- Collecting a process core
- Inspecting a process core
- Collecting a vmcore
- Inspecting a vmcore

What's a crash?

- **Crash**
  - A very generic term used usually to say that the system has come to halt and no progress is observed. The system seems unresponsive or has already rebooted.
- **Panic**
  - A *voluntary* halt to all system activity when an abnormal situation is detected by the kernel.
- **Oops**
  - Similar to panics, but the kernel deems that the situation is not hopeless, so it kills the offending process and continues.

What's a crash?

- **BUG()**
  - Similar to a panic, but is called by intentional code meant to check abnormal conditions.
- **Hang**
  - The system does not seem to be making any progress. System does not respond to normal user interaction. Hangs can be soft or hard.

The BUG() macro

- Called by kernel code when an abnormal situation is seen
- Typically indicates a programming error when triggered
- The calling code is intentional code written by the developer
- Calls look like:

```
BUG_ON(in_interrupt());
```

- Inserts an invalid operand (0x0000) to serve as a landmark by the trap handler
- Output looks like:

```
Kernel BUG at spinlock:118
invalid operand: 0000 [1] SMP
CPU 0
```

Bad pointer handling

- Typically indicates a programming error
- Typically appear as:

  ```
  NULL pointer dereference at
  0x1122334455667788 ..
  ```

  or maybe ..

  ```
  Unable to handle kernel paging request at
  virtual address 0x11223344
  ```

- Detection of those situations is hardware assisted (MMU)
- Typically due to:
  - NULL pointer dereference
  - Accessing a non-canonical address on AMD Opteron
  - Accessing an illegal address on this architecture

Machine Check Exceptions

- Component failures detected and reported by the hardware via an exception
- Typically looks like:

```
kernel: CPU 0: Machine Check Exception:
      4 Bank 0: b278c00000000175
kernel: TSC 4d9eab664a9a60
kernel: Kernel panic - not syncing: Machine
check
```

- To decode, pipe entire line through `mcelog --ascii`
- Always indicates a hardware problem

NMI watchdog

- A hardware mechanism available on modern hardware (APIC) which detects CPU lockups
- When active, the “NMI” count should keep increasing in `/proc/interrupts`
- When a CPU fails to acknowledge an NMI interrupt after some time, the hardware triggers an interrupt and the corresponding handler is executed. The handler would typically call panic()
- Typically indicates a deadlock situation: a running process attempts to a acquire a spinlock which is never granted

EDAC

- *Error Detection and Correction* (aka BlueSmoke) is a hardware mechanism to detect and report memory chip and PCI transfer errors
- Introduced in RHEL 4 Update 3 for intel chips and in RHEL 4.5 for AMD chips
- Reported in `/sys/devices/system/edac/{mc/,pci}` and logged by the kernel as:

```
EDAC MC0: CE page 0x283, offset 0xce0, grain 8,
syndrome 0x6ec3, row 0, channel 1 "DIMM_B1":
amd76x_edac
```

Other hardware reports

- Machine Check Architecture. I have never seen this on i386 and x86_64.
- Machine Specific Register
- NMI notifications about ECC and other hardware problems. Typically look like:

```
Uhhuh. NMI received. Dazed and confused, but
trying to continue
You probably have a hardware problem with your
RAM chips
Uhhuh. NMI received for unknown reason 32. Dazed
and confused, but trying to continue.
Do you have a strange power saving mode enabled?
```

Pseudo-hangs

- In certain situations, the system **appears** to be hang, but some progress is being made
- Those situations include:
  - Thrashing – continuous swapping with close to no useful processing done
  - Lower zone starvation – on i386 the low memory has a special significance and the system may “hang” even when there's plenty of free memory
  - Memory starvation in one node in a NUMA system
- Hangs which are not detected by the hardware are trickier to debug:
  - Use [sysrq + t] to collect process stack traces when possible
  - Enable the NMI watchdog which should detect those situations
  - Run hardware diagnostics when it's a hard hang: memtest86, hp diagnostics

The OOM killer

- In certain memory starvation cases, the OOM killer is triggered to force the release of some memory by killing a “suitable” process
- In severe starvation cases, the OOM killer may have to panic the system when no killable processes are found:

```
Kernel panic – not syncing: Out of memory and no killable processes...
```

Investigating “crashes”

- Take a memory image dump of the process (or whole RAM) when the error occurred
- Allows offline debugging
- Use gdb for userspace processes
- Use crash, a modified gdb, for kernel dumps

Analogy with userspace

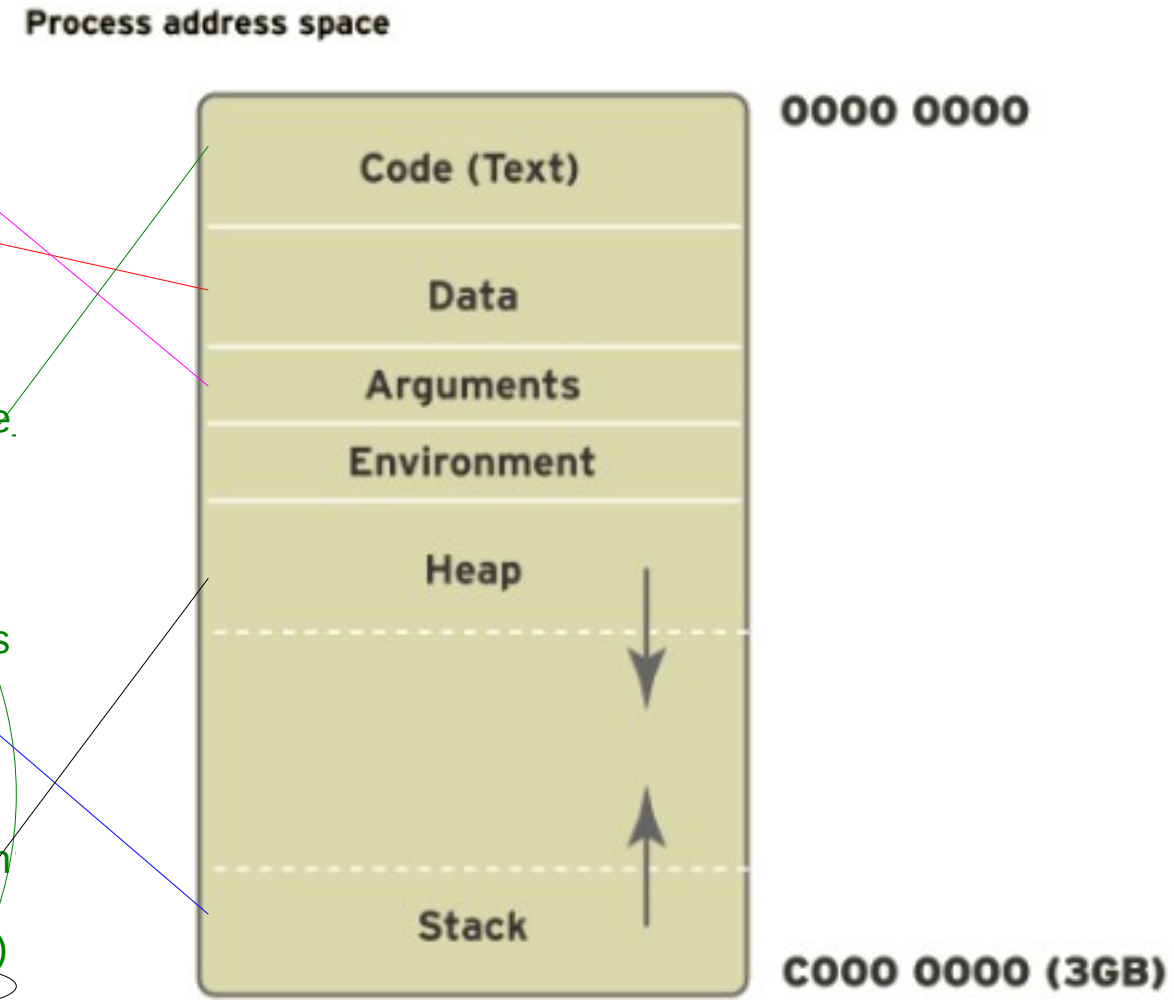
- signals ~ interrupts
- core ~ vmcore
- segfault ~ panic
- gdb ~ crash

User processes memory layout

```
#include <stdio.h>
   #include <stdlib.h>
   int main(int argc, char **argv) {
     int max_table_size = 200;
     int sum = 0;
     int i;
     int table_size;
     int *table;
     if (argc < 2) {
       fprintf(stderr, "Usage: %s <table_size>\n", argv[0]);
       return 1;
     }
     table_size = atoi(argv[1]);
     if (table_size > max_table_size) {
       fprintf(stderr, "table_size needs to be smaller than %d. \
       Aborting..\n", \
       max_table_size);
       return 1;
     }
     table = malloc(table_size*sizeof(int));
     for (i=0; i<table_size; i++) {
       fprintf(stdout, "%d  ", table[i]);
       sum += table[i];
     }
     fprintf(stdout, "sum is %d.\n", sum);
     return 0;
   }
```

User processes memory layout

```
#include <stdio.h>
   #include <stdlib.h>
   int main(int argc, char **argv) {
     int max_table_size = 200;
     int sum = 0;
     int i;
     int table_size;
     int *table;
     if (argc < 2) {
       fprintf(stderr, "Usage: %s <table
       return 1;
     }
     table_size = atoi(argv[1]);
     if (table_size > max_table_size) {
       fprintf(stderr, "table_size needs
       Aborting..\n", \
       max_table_size);
       return 1;
     }
     table = malloc(table_size*sizeof(in
     for (i=0; i<table_size; i++) {
       fprintf(stdout, "%d  ", table[i])
       sum += table[i];
     }
     fprintf(stdout, "sum is %d.\n", sum);
     return 0;
   }
```



Dumping a process core

- `ulimit -c unlimited`
- `kill -SIGSEGV` to force a core dump
- Inspect with:
  `gdb <path_to_binary> core.<pid>`
- Need to build programmes with `-g` for debug symbols
- Install `-debuginfo` packages on RHEL

Lab 1
Using gdb on cores

Collecting a vmcore -- netdump

- Server configuration
  - install package netdump-server
  - passwd netdump
  - enable the server
- Client configuration
  - install package netdump
  - set NETDUMPADDR in /etc/sysconfig/netdump
  - service netdump propagate
  - enable the service

Collecting a vmcore -- diskdump

- install package diskdumputils
- modprobe diskdump
- set DEVICE in /etc/sysconfig/diskdump
- service diskdump initialformat
- enable the service
- savecore -v <dump_device>

Collecting a vmcore -- kdump

- set destination in /etc/kdump.conf
- service kdump propagate
- boot with crashkernel=128M@16M
- service kdump restart

Inspecting a vmcore

- Install -debuginfo from ftp.redhat.com
- Use crash
- Need to know how the code works

Lab 2

Collecting a vmcore

Analysing a vmcore