Linux Crash Dump Analysis

Crash Dump Analysis 2014/2015

CHARLES UNIVERSITY IN PRAGUE

faculty of mathematics and physics

Department of Distributed and Dependable Systems

ORACLE

SUSE

redhat

Agenda

- Userspace debugging
- Understanding kernel oops/panic output
- Creating kernel crash dumps
- Tools for crash dump analysis
- Debugging live systems

User space debugging

- Crashing application
  - Through an (unhadled) signal
    - `SIGABRT, SIGFPE, SIGSEGV, SIGBUS…`
  - Typically produces a line in kernel log (`dmesg`)
    - `modprobe[833]: segfault at 7fff76200038 ip 00007f0de8422fc2 sp 00007fff761b6cb0 error 4 in ld-2.11.1.so[7f0de8420000+20000]`
  - Will produce a `core(5)` file if
    - Limits allow it (`ulimit -c; /etc/security/limits.conf`)
    - Binary is readable and not suid
    - …
- Application stuck in syscall
  - `cat /proc/$PID/stack`

User space debugging

- Executing a program under gdb
  - Relies on the `ptrace(2)` syscall
  - `gdb /path/to/binary`
  - `(gdb) run $param1 $param2 ...`
- Attaching to a running program
  - `gdb -p $PID`
  - We can also create core file (without crashing)
    - `(gdb) generate-core-file`
- Inspecting the core file
  - `gdb /path/to/binary /path/to/core`

User space debugging

- `strace` – tool for tracing system calls and signals
  - Prints system call parameters and return values with symbolic translation
    - `open("/foo/bar", O_RDONLY) = -1 ENOENT (No such file or directory)`
    - Tries to keep ordering of enter/return between threads
    - Dereferences structure members
    - Can attach to a PID
  - "In some cases, strace output has proven to be more readable than the source."
- `valgrind` – for finding memory access bugs

Kernel oops/panic output

- Printed in console typically on fatal CPU exceptions
  - Lots of architecture-specific information
  - May contain enough information to figure out the problem without a full crash dump
- Oops leaves the system running
  - Kills just the current process (including kernel threads!)
  - System can still be left inconsistent (locks remain locked …)
- Panic kills the system completely
  - Oops in interrupt, with `panic_on_oops` enabled, manual `panic()` calls
    - HW failure, critical memory allocation fail, init/idle task killed, int. handler killed
  - May trigger crash dump if configured, or reboot after delay

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff8800916fbb68  EFLAGS: 00010286
[  266.492285] RAX: ffff88008f6b3ba0 RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[  266.492331] RDX: ffff8800914b3c98 RSI: 0000000000000001 RDI: ffff8800914b3c98
[  266.492376] RBP: ffff8800916fbba8 R08: 0000000000000002 R09: 0000000000000000
[  266.492421] R10: 0000000000000008 R11: 0000000000000001 R12: ffff88008f686068
[  266.492465] R13: ffff8800914b3c98 R14: ffff88008f6b3b90 R15: ffff88008f686000
[  266.492513] FS:  00007fb8966f6700(0000) GS:ffff88011ed80000(0000)
knlGS:0000000000000000
[  266.492566] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  266.492601] CR2: 00007f50fa190770 CR3: 0000000001b31000 CR4: 00000000000407e0
[  266.492652] Stack:
[  266.492665]  0000000000000000 ffff88008f686078 ffff8800916fbba8 ffff88008f686000
[  266.492714]  ffff8800916fbc08 0000000000000000 0000000000000000 ffff88008f686000
[  266.492764]  ffff8800916fbbf8 ffffffff8111ba5d 00007fb885918000 ffff88008edf3000
...
```

Example kernel oops

```
...
[  266.492815] Call Trace:
[  266.492834]  [<ffffffff8111ba5d>] free_pgtables+0x8e/0xcc
[  266.492873]  [<ffffffff8112253e>] exit_mmap+0x84/0x116
[  266.492907]  [<ffffffff8103f789>] mmput+0x52/0xe9
[  266.492940]  [<ffffffff81043918>] do_exit+0x3cd/0x9c9
[  266.492975]  [<ffffffff8170c1ec>] ? _raw_spin_unlock_irq+0x2d/0x32
[  266.493016]  [<ffffffff81044d7f>] do_group_exit+0x4c/0xc9
[  266.493051]  [<ffffffff8104eb87>] get_signal+0x58f/0x5bc
[  266.493090]  [<ffffffff810022c4>] do_signal+0x28/0x5b1
[  266.493123]  [<ffffffff8170ca0c>] ? sysret_signal+0x5/0x43
[  266.493162]  [<ffffffff81002882>] do_notify_resume+0x35/0x68
[  266.493200]  [<ffffffff8170cc7f>] int_signal+0x12/0x17
[  266.493235] Code: e8 03 b7 f4 ff 49 8b 47 78 4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43 10 48 39 45 c8 74 55 48 8b 7b 08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8b 53 10 48 89 df 48 89 42
[  266.493404] RIP  [<ffffffff81126630>] unlink_anon_vmas+0x102/0x159
[  266.493447]  RSP <ffff8800916fbb68>
[  266.508877] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.508880] Fixing recursive fault but reboot is needed!
```

(source: https://lkml.org/lkml/2015/1/11/14)

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdk  d amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm:  Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLi  allina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 t  f8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff8112  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff880091 fbb  00010286
[  266.492285] RAX  3b90 RCX: ffff8800a3b3cf30
[  266.492331] RD  01 RDI: ffff8800914b3c98
[  266.492376] RE  02 R09: 0000000000000000
[  266.492421] R1  01 R12: ffff88008f686068
[  266.492465] R1  90 R15: ffff88008f686000
[  266.492513] FS  d80000(0000)
knlGS:0000000000
[  266.492566] CS  80050033
[  266.492601] CR  00 CR4: 00000000000407e0
[  266.492652] St
[  266.492665]  0  00916fbba8 ffff88008f686000
[  266.492714]  f  0000000000 ffff88008f686000
[  266.492764]  f  b885918000 ffff88008edf3000
...
```

File + line translation enabled by CONFIG_DEBUG_BUGVERBOSE (implemented by `__bug_table` section on x86 - ~70-100kB)

The line in question contains:
`BUG_ON(anon_vma->degree);`

This is essentially a hard assertion:
`if (<condition>) BUG()`

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware na      AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8      b3c840 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[      fff81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x1
[  266.492249] RSP: 0018:f      16fbb68  EFLAGS: 00010286
[  266.492285] RAX: ffff88      0 RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[                               0000000000000001 RDI: ffff8800914b3c98
[                               0000000000000002 R09: 0000000000000000
[                               0000000000000001 R12: ffff88008f686068
[                               ffff88008f6b3b90 R15: ffff88008f686000
[                                GS:ffff88011ed80000(0000)
kn
[                               CR0: 0000000080050033
[                               0000000001b31000 CR4: 00000000000407e0
[
[                               8f686078 ffff8800916fbba8 ffff88008f686000
[  266.492714]                  0000000000000000 0000000000000000 ffff88008f686000
[  266.492764]  ffff8800916fbbf8 ffffffff8111ba5d 00007fb885918000 ffff88008edf3000
...
```

On x86, BUG() emits a standardized invalid opcode UD2 (0F 0B) triggering an exception.
The exception handler checks for UD2 opcode and searches __bug_table for details.

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm f e
[  266.492043] CPU: 3 PID: 5155 mm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: A BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a 40 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ff 81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:fff 5fbb68  EFLAGS: 00010286
[  266.492285] RAX: ffff880 RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[  266.4 I: 0000000000000001 RDI: ffff8800914b3c98
[  266 0000000000000002 R09: 0000000000000000
[  26 0000000000000001 R12: ffff88008f686068
[  26 ffff88008f6b3b90 R15: ffff88008f686000
[  26  GS:ffff88011ed80000(0000)
knlGS
[  26  CR0: 0000000080050033
[  26 0000000001b31000 CR4: 00000000000407e0
[  26
[  26 f686078 ffff8800916fbba8 ffff88008f686000
[  26 0000000 0000000000000000 ffff88008f686000
[  26 111ba5d 00007fb885918000 ffff88008edf3000
...
```

x86- and exception-specific error code (32-bit hex number). Typically useful for page fault exceptions where it's a mask:

Bit 0 – Present
Bit 1 – Write
Bit 2 – User
Bit 3 – Reserved write
Bit 4 – Instruction fetch

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD F      INA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b      ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffff      126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff      bb68  EFLAGS: 00010286
[  266.492285] RAX: ffff880      RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[  266      0000000000000001 RDI: ffff8800914b3c98
[  2      0000000000000002 R09: 0000000000000000
[  1      0000000000000001 R12: ffff88008f686068
[  1      fff88008f6b3b90 R15: ffff88008f686000
[  2      S:ffff88011ed80000(0000)
knl
[  1      CR0: 0000000080050033
[  2      000000001b31000 CR4: 00000000000407e0
[  1
[  1      586078 ffff8800916fbba8 ffff88008f686000
[  2      000000 0000000000000000 ffff88008f686000
[  1      11ba5d 00007fb885918000 ffff88008edf3000
...
```

Oops counter, followed by state of selected important kernel config options:

PREEMPT
SMP
DEBUG_PAGEALLOC
KASAN

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD B      INA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b      ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<fff      126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492240] RSP: 0018:ff      b68  EFLAGS: 00010286
[  2      ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[        0000000000001 RDI: ffff8800914b3c98
[        0000000000002 R09: 0000000000000000
[        0000000000001 R12: ffff88008f686068
[        ff88008f6b3b90 R15: ffff88008f686000
[        S:ffff88011ed80000(0000)
kn
[        R0: 0000000080050033
[        00000001b31000 CR4: 00000000000407e0
[
[        36078 ffff8800916fbba8 ffff88008f686000
[        00000 0000000000000000 ffff88008f686000
[        1ba5d 00007fb885918000 ffff88008edf3000
..
```

Mostly useful when it is known which drivers are built as modules (e.g. with standard distro kernel configs).

May also contain module taint flags:
P – proprietary
O – out-of-tree
F – force-loaded
C – staging
E – unsigned
X – external
+/- – being loaded/unloaded

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff88[illegible]3b3c840 ti: [illegible]16f8000
[  266.492191] RIP: 0010:[<ff[illegible]fff81126[illegible]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff88[illegible]
[  266.492285] RAX: ffff88008f6b[illegible]
[  266.492331] RDX: ffff8800914b3c[illegible]
[  266.492376] RBP: ffff8800916fbba[illegible]
[  266.492421] R10: 0000000000000008
[  266.492465] R13: ffff8800914b3c98 R[illegible]
[  266.492513] FS:  00007fb8966f6700(00[illegible]
knlGS:0000000000000000
[  266.492566] CS:  0010 DS: 0000 ES: 0[illegible]
[  266.492601] CR2: 00007f50fa190770 CR[illegible]
[  266.492652] Stack:
[  266.492665]  0000000000000000 ffff88[illegible]000
[  266.492714]  ffff8800916fbc08 000000[illegible]000
[  266.492764]  ffff8800916fbbf8 ffffff[illegible]000
...
```

Information about CPU, process, kernel version, hardware.

Taint flags:
POFCEX – same as per-module
R – module was force-unloaded
M – system has reported a MCE
B – bad page was encountered
U – userspace-defined
D – there was an oops before
W – there was a warning before
A – ACPI table was overriden
I – firmware bug workaround
L – soft-lockup has occurred before
K – kernel has been live patched
S – SMP kernel on UP machine

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff8800916fbb68  EFLA        10286
[  266.492285] RAX: ffff88008f6b3ba0 RBX: ffff88        RCX: ffff8800a3b3cf30
[  266.492331] RDX: ffff8800914b3c98 RSI:
[  266.492376] RBP: ffff8800916fbba8 R08
[  266.492421] R10: 0000000000000008 R11
[  266.492465] R13: ffff8800914b3c98 R14
[  266.492513] FS:  00007fb8966f6700(0000
knlGS:0000000000000000
[  266.492566] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  266.492601] CR2: 00007f50fa190770 CR3: 0000000001b31000 CR4: 00000000000407e0
[  266.492652] Stack:
[  266.492665]  0000000000000000 ffff88008f686078 ffff8800916fbba8 ffff88008f686000
[  266.492714]  ffff8800916fbc08 0000000000000000 0000000000000000 ffff88008f686000
[  266.492764]  ffff8800916fbbf8 ffffffff8111ba5d 00007fb885918000 ffff88008edf3000
...
```

Information about task that's supposed to be currently running, and whose stack we are actually running on.

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff      0916fbb68  EFLAGS: 00010286
[  266.492285] RAX: ffff8800       0 RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[  266.492331] RDX: ffff8800       RSI: 0000000000000001 RDI: ffff8800914b3c98
[  266.492376] RBP: ffff                                     0000000000000000
[  266.492421] R10:                                          ff88008f686068
[  266.492465] R13:                                          ff88008f686000
[  266.492513] FS:                                           0)
knlGS:0000000000000
[  266.492566] CS:
[  266.492601] CR2:                                          000000000407e0
[  266.492652] Stac
[  266.492665]  0000                                         ffff88008f686000
[  266.492714]  ffff                                         ffff88008f686000
[  266.492764]  ffff8800                                     ffff88008edf3000
...
```

Which instruction was executing, translated to function name + offset.

This may be different from where position where BUG_ON( ) was reported, if the Function containing BUG_ON was inlined.

Example kernel oop

Values for the rest of the registers at the trapping instruction. Some are clearly kernel addresses. Some may hold the bad value of anon_vma->degree. Maybe RSI, R08, R10 or R11?

```
[  266.491864] ------------[ cut here ]--
[  266.491904] kernel BUG at mm/rmap.c:39
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu                cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tain                .0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina, B              3L3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 ti: ffff880091600 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff81126630>]  [<ffff fff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff8800916fbb68  EFLAGS: 00010286
[  266.492285] RAX: ffff88008f6b3ba0 RBX: ffff88008f6b3b90 RCX: ffff8800a3b3cf30
[  266.492331] RDX: ffff8800914b3c98 RSI: 0000000000000001 RDI: ffff8800914b3c98
[  266.492376] RBP: ffff8800916fbba8 R08: 0000000000000002 R09: 0000000000000000
[  266.492421] R10: 0000000000000008 R11: 0000000000000001 R12: ffff88008f686068
[  266.492465] R13: ffff8800914b3c98 R14: ffff88008f6b3b90 R15: ffff88008f686000
[  266.492513] FS:  00007fb8966f6700(0000) GS:ffff88011ed80000(0000)
knlGS:0000000000000000
[  266.492566] CS:  0010 DS: 0000 ES: 0000 CR0: 0000000080050033
[  266.492601] CR2: 00007f50fa190770 CR3: 0000000001b31000 CR4: 00000000000407e0
[  266.492652] Stack:
[  266.492665]  0000000000000000 ffff88008f686078 ffff8800916fbba8 ffff88008f686000
[  266.492714]  ffff8800916fbc08 0000000000000000 0000000000000000 ffff88008f686000
[  266.492764]  ffff8800916fbbf8 ffffffff8111ba5d 00007fb885918000 ffff88008edf3000
...
```

Example kernel oops

```
[  266.491864] ------------[ cut here ]------------
[  266.491904] kernel BUG at mm/rmap.c:399!
[  266.491934] invalid opcode: 0000 [#1] SMP
[  266.491962] Modules linked in: amdkfd amd_iommu_v2 radeon cfbfillrect cfbimgblt
cfbcopyarea drm_kms_helper ttm fuse
[  266.492043] CPU: 3 PID: 5155 Comm: java Not tainted 3.19.0-rc3-kfd+ #24
[  266.492087] Hardware name: AMD BALLINA/Ballina, BIOS WBL3B20N_Weekly_13_11_2
11/20/2013
[  266.492141] task: ffff8800a3b3c840 ti: ffff8800916f8000 task.ti: ffff8800916f8000
[  266.492191] RIP: 0010:[<ffffffff81126630>]  [<ffffffff81126630>]
unlink_anon_vmas+0x102/0x159
[  266.492249] RSP: 0018:ffff8800916fb
[  266.492285] RAX: ffff88008f6b3ba0 RB                                  cf30
[  266.492331] RDX: ffff8800914b3c98 RS                                  3c98
[  266.492376] RBP: ffff8800916fbba8 R08: 000                0002 R09: 0000000000000000
[  266.492421] R10: 0000000000000008 R11: 0000               0001 R12: ffff88008f686068
[  266.492465] R13: ffff8800914b3c98 R14: ffff8              b3b90 R15: ffff88008f686000
[  266.492513] FS:  00007fb8966f6700(0000) GS:ff             11ed80000(0000)
knlGS:0000000000000000
[  266.492566] CS:  0010 DS: 0000 ES: 0000 CR0: 0            000080050033
[  266.492601] CR2: 00007f50fa190770 CR3: 00000000           b31000 CR4: 00000000000407e0
[  266.492652] Stack:
[  266.492665]  0000000000000000 ffff88008f686078 ffff8800916fbba8 ffff88008f686000
[  266.492714]  ffff8800916fbc08 0000000000000000 0000000000000000 ffff88008f686000
[  266.492764]  ffff8800916fbbf8 ffffffff8111ba5d 00007fb885918000 ffff88008edf3000
...
```

Raw contents of top of the stack starting at RSP

Example kernel oops

Backtrace reconstructed by unwinding the stack, showing the return addresses from individual call frames

"?" means a pointer to function is on stack but doesn't fit in the frame; could be leftover from previous execution or heuristics failure

```
...
[  266.492815] Call Trace:
[  266.492834]  [<ffffffff8111ba5d>] free_pgtables+0x8e/0xcc
[  266.492873]  [<ffffffff8112253e>] exit_mmap+0x84/0x116
[  266.492907]  [<ffffffff8103f789>] mmput+0x52/0xe9
[  266.492940]  [<ffffffff81043918>] do_exit+0x3cd/0x9c9
[  266.492975]  [<ffffffff8170c1ec>] ? _raw_spin_unlock_irq+0x2d/0x32
[  266.493016]  [<ffffffff81044d7f>] do_group_exit+0x4c/0xc9
[  266.493051]  [<ffffffff8104eb87>] get_signal+0x58f/0x5bc
[  266.493090]  [<ffffffff810022c4>] do_signal+0x28/0x5b1
[  266.493123]  [<ffffffff8170ca0c>] ? sysret_signal+0x5/0x43
[  266.493162]  [<ffffffff81002882>] do_notify_resume+0x35/0x68
[  266.493200]  [<ffffffff8170cc7f>] int_signal+0x12/0x17
[  266.493235] Code: e8 03 b7 f4 ff 49 8b 47 78 4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43 10 48 39 45 c8 74 55 48 8b 7b 08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8b 53 10 48 89 df 48 89 42
[  266.493404] RIP  [<ffffffff81126630>] unlink_anon_vmas+0x102/0x159
[  266.493447]  RSP <ffff8800916fbb68>
[  266.508877] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.508880] Fixing recursive fault but reboot is needed!
```

Example kernel oops

```
...
[  266.492815] Call Trace:
[  266.492834]  [<ffffffff8111
[  266.492873]  [<ffffffff8112
[  266.492907]  [<ffffffff810
[  266.492940]  [<ffffffff81
[  266.492975]  [<ffffffff
[  266.493016]  [<fffffff
[  266.493051]  [<ffffff
[  266.493090]  [<ffff
[  266.493123]  [<ff
[  266.493162]  [<f
[  266.493200]  [<ffffffff8170cc7f>] int_signal+0x12/0x17
[  266.493235] Code: e8 03 b7 f4 ff 49 8b 47 78 4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43 10 48 39 45 c8 74 55 48 8b 7b 08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8b 53 10 48 89 df 48 89 42
[  266.493404] RIP  [<ffffffff81126630>] unlink_anon_vmas+0x102/0x159
[  266.493447]  RSP <ffff8800916fbb68>
[  266.508877] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.508880] Fixing recursive fault but reboot is needed!
```

A bunch of instructions around the RIP.
RIP position denoted by < >

Recall that 0F 0B is opcode for UD2

We can disassemble the code listing by piping the oops into `./scripts/decodecode` in the kernel source tree.

Example decodecode output

```
~/linux.git> ./scripts/decodecode < oops-example.txt

[ 266.493235] Code: e8 03 b7 f4 ff 49 8b 47 78 4c 8b 20 48 8d 58 f0 49 83 ec 10 48 8d 43 10 48 39 45 c8 74 55
48 8b 7b 08 83 bf 8c 00 00 00 00 74 02 <0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8b 53 10 48 89 df 48 89 42
All code
========
   0:   e8 03 b7 f4 ff          callq  0xfffffffffff4b708
   5:   49 8b 47 78             mov    0x78(%r15),%rax
   9:   4c 8b 20                mov    (%rax),%r12
   c:   48 8d 58 f0             lea    -0x10(%rax),%rbx
  10:   49 83 ec 10             sub    $0x10,%r12
  14:   48 8d 43 10             lea    0x10(%rbx),%rax
  18:   48 39 45 c8             cmp    %rax,-0x38(%rbp)
  1c:   74 55                   je     0x73
  1e:   48 8b 7b 08             mov    0x8(%rbx),%rdi
  22:   83 bf 8c 00 00 00 00    cmpl   $0x0,0x8c(%rdi)
  29:   74 02                   je     0x2d
  2b:*  0f 0b                   ud2              <-- trapping instruction
  2d:   e8 a4 fd ff ff          callq  0xffffffffffffdd6
  32:   48 8b 43 18             mov    0x18(%rbx),%rax
  36:   48 8b 53 10             mov    0x10(%rbx),%rdx
  3a:   48 89 df                mov    %rbx,%rdi
  3d:   48                      rex.W
  3e:   89                      .byte 0x89
  3f:   42                      rex.X

Code starting with the faulting instruction
===========================================
   0:   0f 0b                   ud2
   2:   e8 a4 fd ff ff          callq  0xfffffffffffffdab
   7:   48 8b 43 18             mov    0x18(%rbx),%rax
   b:   48 8b 53 10             mov    0x10(%rbx),%rdx
   f:   48 89 df                mov    %rbx,%rdi
  12:   48                      rex.W
  13:   89                      .byte 0x89
  14:   42                      rex.X
```

Example decodecode output

```
All code
========
   0:   e8 03 b7 f4 ff          callq  0xfffffffffff4b708
   5:   49 8b 47 78             mov    0x78(%r15),%rax
   9:   4c 8b 20                mov    (%rax),%r12
   c:   48 8d 58 f0             lea    -0x10(%rax),%rbx
  10:   49 83 ec 10             sub    $0x10,%r12
  14:   48 8d 43 10             lea    0x10(%rbx),%rax
  18:   48 39 45 c8             cmp    %rax,-0x38(%rbp)
  1c:   74 55                   je     0x73
  1e:   48 8b 7b 08             mov    0x8(%rbx),%rdi
  22:   83 bf 8c 00 00 00 00    cmpl   $0x0,0x8c(%rdi)
  29:   74 02                   je     0x2d
  2b:*  0f 0b                   ud2              <-- trapping instruction
  2d:   e8 a4 fd ff ff          callq  0xffffffffffffdd6
  32:   48 8b 43 18             mov    0x18(%rbx),%rax
  36:   48 8b 53 10             mov    0x10(%rbx),%rdx
  3a:   48 89 df                mov    %rbx,%rdi
  3d:   48                      rex.W
  3e:   89                      .byte 0x89
  3f:   42                      rex.X
```

Examp…ut

BUG_ON(anon_vma->degree);

We skip the UD2 instruction if 0x8c(%rdi) equals zero → we trap if the value is non-zero

This suggests that RDI holds the struct anon_vma pointer and degree is at offset 0x8c

However, we can't determine the value that had been compared to zero in this case...

```
All code
========
   0:    e…                                  …f4b708
   5:    4…                                  …rax
   9:    4…
   c:    4…                                  …%rbx
  10:    4…
  14:    48 …                                …,%rax
  18:    48 39 45 c8                         …,-0x38(%rbp)
  1c:    74 55                               je     …0x73
  1e:    48 8b 7b 08                         mo…    0x8(%rbx),%rdi
  22:    83 bf 8c 00 00 00 00                cmpl   $0x0,0x8c(%rdi)
  29:    74 02                               je     0x2d
  2b:*   0f 0b                               ud2            <-- trapping instruction
  2d:    e8 a4 fd ff ff                      callq  0xfffffffffffffdd6
  32:    48 8b 43 18                         mov    0x18(%rbx),%rax
  36:    48 8b 53 10                         mov    0x10(%rbx),%rdx
  3a:    48 89 df                            mov    %rbx,%rdi
  3d:    48                                  rex.W
  3e:    89                                  .byte 0x89
  3f:    42                                  rex.X
```

Example decodecode output

```
All code
========
   0:   e8 03 b7 f4 ff
   5:   49 8b 47 78
   9:   4c 8b 20
   c:   48 8d 58 f0
  10:   49 83 ec 10
  14:   48 8d 43 10
  18:   48 39 45 c8
  1c:   74 55
  1e:   48 8b 7b 08
  22:   83 bf 8c 00 00 00 00
  29:   74 02
  2b:*  0f 0b
  2d:   e8 a4 fd ff ff
  32:   48 8b 43 18
  36:   48 8b 53 10
  3a:   48 89 df
  3d:   48
  3e:   89
  3f:   42
```

```
su                12
lea             %rbx),%rax
cmp             x,-0x38(%rbp)
je              0x73
mov     0x8(%rbx),%rdi
cmpl    $0x0,0x8c(%rdi)
je      0x2d
ud2                     <-- trapping instruction
callq   0xffffffffffffdd6
mov     0x18(%rbx),%rax
mov     0x10(%rbx),%rdx
mov     %rbx,%rdi
rex.W
.byte 0x89
rex.X
```

```
struct anon_vma_chain *avc;
list_for_each_entry_safe(avc, ...)
    struct anon_vma *anon_vma = avc->anon_vma;
    BUG_ON(anon_vma->degree);
```

Suggests that RBX holds the `struct anon_vma_chain` pointer `avc` and `anon_vma` member is at offset `0x8`

Verifying structure offsets

- We can use pahole from dwarves package
  - May depend on GCC version, .config options
    - rwsem size depends on CONFIG_DEBUG_SPINLOCK, CONFIG_DEBUG_LOCK_ALLOC

```
> pahole --hex -C anon_vma mm/vmscan.o
struct anon_vma {
        struct anon_vma *          root;                 /*     0   0x8 */
        struct rw_semaphore        rwsem;                /*   0x8  0x80 */
        /* --- cacheline 2 boundary (128 bytes) was 8 bytes ago --- */
        atomic_t                   refcount;             /*  0x88   0x4 */
        unsigned int               degree;               /*  0x8c   0x4 */
        struct anon_vma *          parent;               /*  0x90   0x8 */
        struct rb_root             rb_root;              /*  0x98   0x8 */

        /* size: 160, cachelines: 3, members: 6 */
        /* last cacheline: 32 bytes */
};
```

Example kernel oops

```
...
[  266.492815] Call Trace:
[  266.492834]  [<ffffffff8111ba5d>] free_pgtables+0x8e/0xcc
[  266.492873]  [<ffffffff8112253e>] exit_mmap+0x84/0x116
[  266.492907]  [<ffffffff8103f789>] mmput+0x52/0xe9
[  266.492940]  [<ffffffff81043918>] do_exit+0x3cd/0x9c9
[  266.492975]  [<ffffffff8170c1
[  266.493016]  [<ffffffff81044
[  266.493051]  [<ffffffff8104e
[  266.493090]  [<ffffffff81002
[  266.493123]  [<ffffffff8170c
[  266.493162]  [<ffffffff8100288
[  266.493200]  [<ffffffff8170cc7f>] i                1+0x12/0x17
[  266.493235] Code: e8 03 b7 f4 ff 49                78 4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43 10 48 39 45 c8 74 55 48                7b 08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8                53 10 48 89 df 48 89 42
[  266.493404] RIP  [<ffffffff81126630>] unlink_anon_vmas+0x102/0x159
[  266.493447]  RSP <ffff8800916fbb68>
[  266.508877] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.508880] Fixing recursive fault but reboot is needed!
```

The most important registers again, with higher `printk` level, or in case the details had scrolled away.

Example kernel oops

```
...
[  266.492815] Call Trace:
[  266.492834]  [<ffffffff8111ba5d>] free_pgtables+0x8e/0xcc
[  266.492873]  [<ffffffff8112253e>] exit_mmap+0x84/0x116
[  266.492907]  [<ffffffff8103f789>] mmput+0x52/0xe9
[  266.492940]  [<ffffffff81043918>] do_exit+0x3cd/0x9c9
[  266.492975]  [<ffffffff8170c1
[  266.493016]  [<ffffffff81044
[  266.493051]  [<ffffffff8104e
[  266.493090]  [<ffffffff81002
[  266.493123]  [<ffffffff8170c
[  266.493162]  [<ffffffff81002882>]
[  266.493200]  [<ffffffff8170cc7f>] ir                    0x12/0x17
[  266.493235] Code: e8 03 b7 f4 ff 49                  4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43 10 48 39 45 c8 74 55 48                  08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fd ff ff 48 8b 43 18 48 8                 10 48 89 df 48 89 42
[  266.493404] RIP  [<ffffffff81126630>] unlink_anon_vmas+0x102/0x159
[  266.493447]  RSP <ffff8800916fbb68>
[  266.508877] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.508880] Fixing recursive fault but reboot is needed!
```

Randomization to distinguish reports of same bug instance from separate instances.

Example kernel oops

The task was already exiting when it oopsed. In this case it's clearly graceful exit (from the backtrace), but it could be exiting due to previous oops. It's safer to leave task as zombie than to risk infinite loops in the exit path.

```
...
[
[                                                                    e/0xcc
[                                                                    x116
[  266.49
[  266.49294                 ff81043918>] do_exit+0x3cd/0x9c9
[  266.49297                 fff8170c1ec>] ? _raw_spin_unlock_irq+0x2d/0x32
[  266.49301                 fff81044d7f>] do_group_exit+0x4c/0xc9
[  266.49305                 ffff8104eb87>] get_signal+0x58f/0x5bc
[  266.49309                 fffff810022c4>] do_signal+0x28/0x5b1
[  266.49312                 fffff8170ca0c>] ? sysret_signal+0x5/0x43
[  266.49316                 ffffff81002882>] do_notify_resume+0x35/0x68
[  266.49320                 ffffff8170cc7f>] int_signal+0x12/0x17
[  266.49323]      : e8 03 b7 f4 ff 49 8b 47 78 4c 8b 20 48 8d 58 f0 49 83
ec 10 48 8d 43    48 39 45 c8 74 55 48 8b 7b 08 83 bf 8c 00 00 00 00 74 02
<0f> 0b e8 a4 fc    f ff 48 8b 43 18 48 8b 53 10 48 89 df 48 89 42
[  266.49340] R    [<ffffffff81126630>] unlink_anon_vmas+0x102/0x159
[  266.49344]  SP <ffff8800916fbb68>
[  266.50887] ---[ end trace 02d28fe9b3de2e1a ]---
[  266.50888] Fixing recursive fault but reboot is needed!
```

How is stack unwinding implemented?

- Start at value of RSP and increment in a loop
  - Check if stack contains kernel text address
  - Print with translation to function name+offset
  - When RSP matches RBP `+ sizeof(long)`, consider address *reliable* (i.e. without "?") and update RBP from the address it points to
- Not fully reliable, even with frame pointers
  - Cannot be relied upon functionally (live patching?)
  - Assembler functions now audited for missing frame pointers
    - Planned: runtime checks + DWARF validations

How is stack unwinding implemented?

- For perf callgraph sampling, this would be slow
  - Therefore, fully rely on frame pointer walk there
- Alternative approach: use DWARF2 exception handler (EH) frame info
  - Patch in SUSE kernels, rejected upstream
  - Also not fully reliable, and more complex

What else can produce oops/panic?

- BUG_ON seen in the example – hard assertion
- Memory paging related faults
  - "`BUG: unable to handle kernel paging request`"
    - "`... handle NULL pointer dereference`" (when bad_addr < PAGE_SIZE)
  - Corrupted page table
  - Kernel trying to execute NX-protected page
  - Kernel trying to execute userspace page (Intel SMEP)
  - Failed bounds check in kernel mode (Intel MPX feature)
  - General protection fault, unhandled double fault

What else can produce oops/panic?

- Soft lockup
  - CPU spent 20s in kernel without reaching a schedule point
  - A warning, unless config/bootparam `softlockup_panic` enabled
    - Soft lockup can be harmless, so not good idea in production
- Hard lockup
  - CPU spent 10s with disabled interrupts
- Detection of both combines several generic mechanisms
  - High priority kernel watchdog thread updates soft lockup timestamp
  - hrtimer set to deliver periodic interrupts, increments hard lockup counter and wakes up the watchdog thread
  - NMI perf event checks if hrtimers interrupts were processed and if watchdog thread was scheduled

What else can produce oops/panic?

- Hung task check
  - "`INFO: task … blocked for more than 120 seconds`"
  - `khungtaskd` periodically processes tasks in uninterruptible sleep and checks if their switch count changed
- RCU stall detector
  - Detects when RCU grace period is too long (21s)
    - CPU looping in RCU critical section or disabled interrupts, preemption or bottom halves, no scheduling points in non-preempt kernels
    - RT task preempting non-RT task in RCU critical section
- Several other debugging config options (later)

Obtaining crash dumps

- Several historical methods
  - diskdump, netdump, LKCD
  - Not very reliable (some parts of crashed kernel must still work) or universal, needs dedicated server on same network etc.
  - Out of tree patches, included in old enterprise distros
- Current solution: kexec-based `kdump`
  - Crash kernel loaded into a boot-reserved memory area
  - On panic, kexec switches to the crash kernel without reboot
  - Memory of crashed kernel available as `/proc/vmcore`
  - Kdump utility can save to disk, network, filter pages...
    - `kexec(8), kdump(5), makedumpfile(8)`

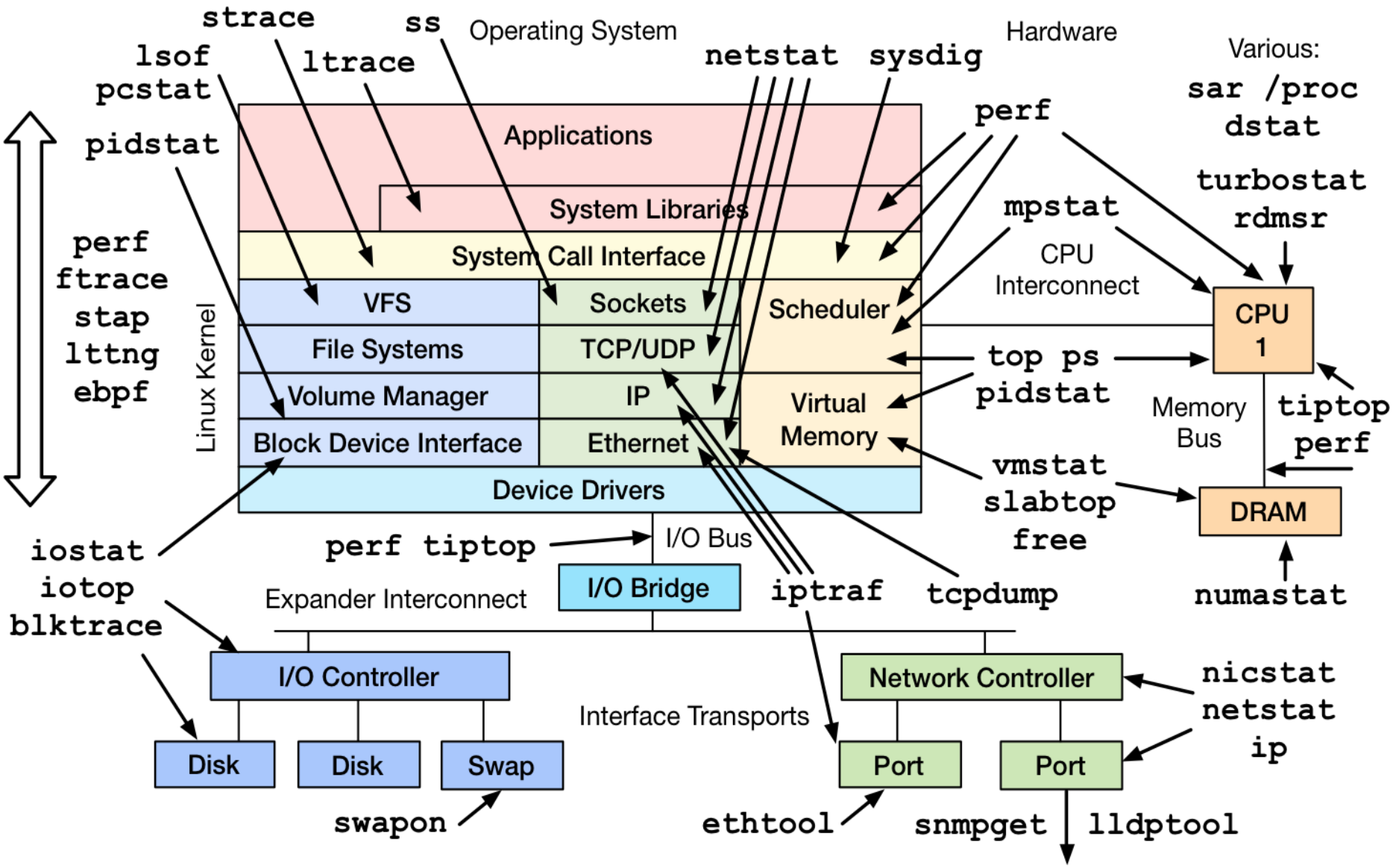
Analyzing crash dumps with gdb

- gdb can be used to open ELF based dumps
  - But those are not easily compressed and filtered
- gdb has no understanding of kernel internals or virtual/physical mapping
  - There are some recently added Python scripts under `scripts/gdb` in the Linux source
    - Can obtain per-cpu variables, dmesg, modules, tasks
- A better tool for Linux kernel crash dumps - `crash`

Debugging live systems

- Observability tools – separate lecture?
- Sysrq magic host keys
- Live debugger support
- Ftrace, kprobes, SystemTap, perf – separate lecture?
- Debugging config options

Linux Performance Observability Tools



http://www.brendangregg.com/linuxperf.html 2015

Standard debugging means

- Magic SysRq hot keys
- For dealing with hangs and security issues
  - Operator's intervention to the running system
  - Can be enabled/disabled by `/proc/sys/kernel/sysrq`
    - Alt + SysRq + 0 .. 9 set console logging level
    - Alt + SysRq + H show help
    - Alt + SysRq + C crash by a NULL pointer dereference
    - Alt + SysRq + B immediate reboot
    - Alt + SysRq + O immediate shutdown
    - Alt + SysRq + S sync all mounted filesystems
    - Alt + SysRq + U remount all filesystems read-only
    - Alt + SysRq + J freeze filesystems by `FIFREEZE` ioctl

Standard debugging means (2)

- Alt + SysRq + P — dump registers to console
- Alt + SysRq + T — dump process information to console
- Alt + SysRq + L — dump stack traces of running threads
- Alt + SysRq + M — dump memory statistics to console
- Alt + SysRq + D — dump locked locks to console
- Alt + SysRq + K — kill all processes on the current console
- Alt + SysRq + E — terminate all processes except `init`
- Alt + SysRq + I — kill all processes except `init`
- Alt + SysRq + F — execute the OOM killer
- Alt + SysRq + N — reset nice level of all real-time processes
- Alt + SysRq + R — switch off raw keyboard mode
- Alt + SysRq + Q — dump armed hritmers, clockevent devices
- Alt + SysRq + V — forcefully restore framebuffer console
- Alt + SysRq + W — dump tasks in uninterruptible sleep
- Alt + SysRq + Z — dump the ftrace buffer

Standard debugging means (3)

- Activate from command line by writing into `/proc/sysrq-trigger`
- Activate over network by a special `sysrqd` server

- **R**aising **E**lephants **I**s **S**o **U**tterly **B**oring **R**eboot **E**ven **I**f **S**ystem **U**tterly **B**roken
  - Raw keyboard
  - Send `SIGTERM` to all processes
  - Send `SIGKILL` to all processes
  - Sync data to disk
  - Remount all filesystems read-only
  - Reboot

Live kernel debugging - /proc/kcore

- `/proc/kcore` enabled by `CONFIG_PROC_KCORE`
  - Provides virtual ELF “core dump” file
  - Usable by gdb and crash for read-only inspection

Live kernel debugging - kgdb

- kgdb was merged in 2.6.26 (2008)
- Provides a server for remote gdb client
  - Over serial port – `CONFIG_KGDB_SERIAL_CONSOLE`
  - Over network using `NETPOLL` – not mainline (KDBoE)
- Enable on server
  - Boot with `kgdboc=ttyS0,115200`
  - `echo g > /proc/sysrq-trigger` or `kgdbwait` boot param
- Use from a client
  - `% gdb ./vmlinux`
  - `(gdb) set remotebaud 115200`
  - `(gdb) target remote /dev/ttyS0`
  - Allows limited gdb debugging similar to a userspace program

Live kernel debugging - kdb

- kdb is a frontend for kgdb that runs in the debugged kernel (no need for other client) – since 2.6.35 (2010)
- Provides a shell accessed via serial terminal, with optional PS/2 keyboard support
  - Enabled same way as the kgdb server
  - Switch between kdb/kgdb by `$3#33` and `kgdb`
- Provides some kernel-specific commands not available in pure gdb
  - `lsmod, ps, ps A, summary, bt, dmesg, go, help`
  - Some can be executed from gdb – `monitor help`
- Out of tree discontinued version seemed to be more capable
- KMS console support was proposed, but dropped

Live debugging - User-Mode Linux

- UML
  - Special pseudo-hardware architecture
    - Otherwise compatible with the target architecture
    - Running Linux kernel as a user space process
      - Originally a virtualization effort
    - Great for debugging and kernel development
      - A plain standard gdb can be used to attach to the running kernel
      - Guest threads are threads of the UML process
        - Slightly more complicated to follow processes

Kernel debugging config options

- Kernel can be built with additional debugging options enabled
  - Extra checks that can catch errors sooner, or provide extra information, at the cost of CPU and/or memory overhead
  - Can also hide errors such as race conditions...
- Many of them under “Kernel hacking” in make menuconfig
  - Others placed in the given subsystem/driver

Kernel debugging config options (VM)

- DEBUG_VM – enable VM_BUG_ON(cond) checks
- PAGE_OWNER – track who allocated which pages in order to find a memory leak
- DEBUG_PAGEALLOC – unmap (or poison) pages after they are freed
- DEBUG_SLAB – detect some cases of double free, or use-after-free (by poisoning)
  - SLUB_DEBUG variant can enable/disable debugging in runtime
- DEBUG_KMEMLEAK – detect leaks with a conservative garbage collection based algorithm
- KASAN – Find out of bounds accesses and use-after-free bugs at the cost of 1/8 memory and 3x slower performance

Kernel debugging config options

- DEBUG_STACKOVERFLOW – check if random corruption involving struct thread_info is caused by too deep call chains
- DEBUG_SPINLOCK and others for different locks – catch missing init, freeing of live locks, some deadlocks
- LOCK_STAT – for lock contention, perf lock
- PROVE_LOCKING - “lockdep” mechanism for online proving that deadlocks cannot happen and report that deadlock can occur before it actually does