DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

Kernel debugging

Tools to understand whatever it is that is happening in there

Dominique Martinet

CEA

January 9, 2020

What to expect

- Mostly to use as a cookbook
- No magic to be found here, you still need to:
  - Try to understand what you're analyzing
  - Read code
  - Play the detective game
- But this should at least help get some tools to work on that

Prerequisites

- Some understanding of C
  - For proper debugging, assembly also help, but we will skip that here
- Standard unix commands
- Good to know (will be brief reminders during hands on):
  - What is and how to read a stack trace
  - Know kernel basic types (most specificially lists)

VM

- These slides are supposed to come with a centos7 VM you can log into. Send a mail for image URL if required.
- Import disk image somewhere or run something similar to this:

```
$ qemu-system-x86_64 -hda kdebug.qcow2 -serial mon:stdio   \
                     -enable-kvm -nographic -smp 16 -m 20G \
                     -cdrom ~/cloud.iso
```

- If you do not have a cloud-init image, you should get a login prompt where root:toor will get you in.

Crash

- Tool to analyze kernel state
- Mostly a big wrapper around gdb

Live analysis

- Cannot get stack trace for active threads (think `/proc/<pid>/stack`)
- Depending on used driver, can modify current VM memory (not default)
- No lock taken when traversing structures: here be dragons.

Post-mortem analysis (`kexec-kdump`)

- found in `/var/crash/<ip>-<date>/vmcore`

Run crash

```
# cd /var/crash/client_lbug
# crash /usr/lib/debug/lib/modules/3.10.0*/vmlinux vmcore
...
      PANIC: "Kernel panic - not syncing: LBUG"
        PID: 12295
    COMMAND: "trinity-c30"
...
crash>
```

Crash notes

- Can print this again anytime with `sys`
- Process shown here is whatever was active on the CPU that caused the crash
- It will be used by default for various commands if not specified
- In case of emergency, type `help`.

Analysis - dmesg


```
crash> dmesg | tail -n 31 | head -n 13
[  319.474274] LustreError: 12295:0:(file.c:2200:ll_put_grouplock())
               ASSERTION( fd->fd_grouplock.lg_lock != ((void *)0) ) failed:
[  319.478957] LustreError: 12295:0:(file.c:2200:ll_put_grouplock()) LBUG
[  319.481942] Pid: 12295, comm: trinity-c30 3.10.0-957.21.3.el7.x86_64
               #1 SMP Tue Jun 18 16:35:19 UTC 2019
[  319.485441] Call Trace:
[  319.487361]  [<ffffffffc08227cc>] libcfs_call_trace+0x8c/0xc0 [libcfs]
[  319.490289]  [<ffffffffc082287c>] lbug_with_loc+0x4c/0xa0 [libcfs]
[  319.493359]  [<ffffffffc0fcb92c>] ll_put_grouplock.isra.46+0x37c/0x380 [lustre]
[  319.496729]  [<ffffffffc0fde8d6>] ll_file_ioctl+0xb76/0x4a20 [lustre]
[  319.500151]  [<ffffffffa4e569d0>] do_vfs_ioctl+0x3a0/0x5a0
[  319.502573]  [<ffffffffa4e56c71>] SyS_ioctl+0xa1/0xc0
[  319.505419]  [<ffffffffa5375ddb>] system_call_fastpath+0x22/0x27
[  319.508395]  [<ffffffffffffffff>] 0xffffffffffffffff
[  319.511226] Kernel panic - not syncing: LBUG
```

Crash notes

- Can use pipes like one would in shell
- Can also redirect in files!

Analysis - bt


```
crash> bt
PID: 12295  TASK: ffff9f955eae6180  CPU: 4   COMMAND: "trinity-c30"
 #0 [ffff9f94f45dfb80] machine_kexec at ffffffffa4c63934
 #1 [ffff9f94f45dfbe0] __crash_kexec at ffffffffa4d1d162
 #2 [ffff9f94f45dfcb0] panic at ffffffffa535c81b
 #3 [ffff9f94f45dfd30] lbug_with_loc at ffffffffc08228cb [libcfs]
 #4 [ffff9f94f45dfd50] ll_put_grouplock at ffffffffc0fcb92c [lustre]
 #5 [ffff9f94f45dfda0] ll_file_ioctl at ffffffffc0fde8d6 [lustre]
 #6 [ffff9f94f45dfe80] do_vfs_ioctl at ffffffffa4e569d0
 #7 [ffff9f94f45dff00] sys_ioctl at ffffffffa4e56c71
 #8 [ffff9f94f45dff50] system_call_fastpath at ffffffffa5375ddb
    RIP: 00007f7a30a931c9  RSP: 00007fff2314c668  RFLAGS: 00010216
    RAX: 0000000000000010  RBX: 0000000000000010  RCX: ffffffffa5375d21
    RDX: 000000007ffff000  RSI: 000000004008669f  RDI: 0000000000000037
    RBP: 00007f7a310ad000   R8: 00000064200a4f56   R9: ffffffffbbbbbbbc
    R10: 0000000110c10320  R11: 0000000000000246  R12: 00007f7a310ad058
    R13: 00007f7a311866b0  R14: 0000000000000000  R15: 00007f7a310ad000
    ORIG_RAX: 0000000000000010  CS: 0033  SS: 002b
```

Crash notes

- Can be much more verbose with switches:
  - `-l` for line numbers
  - `-FF` for (full?) frame
- Registers position matters: syscall entry (here) or interrupt.
  - calling convention for syscall on x86_64: RAX = syscall#, then RDI, RSI, RDX, R10, R8, R9
  - here, the first argument of ioctl is a fd, in RDI.

Analysis - files


```
crash> p/d 0x37
$1 = 55
crash> files
PID: 12295  TASK: ffff9f955eae6180  CPU: 4   COMMAND: "trinity-c30"
ROOT: /    CWD: /tmp/trinity/tmp
 FD       FILE            DENTRY           INODE       TYPE PATH
...
 55 ffff9f956da1ff00 ffff9f94e6411cc0 ffff9f920bdaad90 REG  /mnt/lus
```

Crash notes

- No trinity logs here but could help figure what happened from these or application code

Analysis - `bt -FF`


```
crash> bt -FF
...
 #6 [ffff9f94f45dfe80] do_vfs_ioctl at ffffffffa4e569d0
    ffff9f94f45dfe88: [ffff9f94e6411cc0:dentry] [ffff9f94f45d669f:mm_struct]
    ffff9f94f45dfe98: ffff9f958ed1590b ffff9f94f45dfe80
    ffff9f94f45dfea8: do_setitimer+476 0000000000000000
    ffff9f94f45dfeb8: 00000000ece1f068 [ffff9f956da1ff00:kmalloc-256]
    ffff9f94f45dfec8: 0000000000000037 00000000ece1f068
    ffff9f94f45dfed8: [ffff9f956da1ff00:kmalloc-256] 0000000000000037
    ffff9f94f45dfee8: 000000004008669f 000000007ffff000
    ffff9f94f45dfef8: ffff9f94f45dff48 sys_ioctl+161
```

Crash notes

- Can find here the dentry and file from `files` command
- The `[addr:slab]` syntax only means the address is within the slab: it could be anywhere within the struct!
- Slabs of same size will be shared, e.g. even if the code has a `kmem_cache_create()` you will not necessarily see it here.

Analysis - `bt -l`


```
crash> mod -s libcfs
crash> mod -s lustre
crash> bt -l
```

```
...
 #4 [ffff9f94f45dfd50] ll_put_grouplock at ffffffffc0fcb92c [lustre]
    /usr/src/debug/lustre-2.12.3/lustre/llite/file.c: 2200
 #5 [ffff9f94f45dfda0] ll_file_ioctl at ffffffffc0fde8d6 [lustre]
    /usr/src/debug/lustre-2.12.3/lustre/llite/file.c: 3388
 #6 [ffff9f94f45dfe80] do_vfs_ioctl at ffffffffa4e569d0
    /usr/src/debug/kernel-3.10.0-957.21.3.el7/linux-3.10.0-957.21.3.el7.x86_64/fs/ioctl.c:
...
```

Crash notes

- `mod -s <module>` or `mod -S`
- ... now to read some code and undersatnd what happened!
- cscope helps

Closure on client crash

- Crash isn't magical, you still need to read code and understand how we got there.

Closure on client crash

- Crash isn't magical, you still need to read code and understand how we got there.

```
@@ static int ll_put_grouplock(struct inode *inode, ...
        if (!(fd->fd_flags & LL_FILE_GROUP_LOCKED)) {
                spin_unlock(&lli->lli_lock);
                CWARN("no group lock held\n");
                RETURN(-EINVAL);
        }

        LASSERT(fd->fd_grouplock.lg_lock != NULL);
```

Closure on client crash

- Crash isn't magical, you still need to read code and understand how we got there.

```
@@ static int ll_put_grouplock(struct inode *inode, ...
        if (!(fd->fd_flags & LL_FILE_GROUP_LOCKED)) {
                spin_unlock(&lli->lli_lock);
                CWARN("no group lock held\n");
                RETURN(-EINVAL);
        }

        LASSERT(fd->fd_grouplock.lg_lock != NULL);
```

- Did something reset `fd_grouplock` to NULL without touching the flag?
- Or did something modify `fd_flags` without setting the grouplock?

Closure on client crash

- Crash isn't magical, you still need to read code and understand how we got there.

```
@@ static int ll_put_grouplock(struct inode *inode, ...
        if (!(fd->fd_flags & LL_FILE_GROUP_LOCKED)) {
                spin_unlock(&lli->lli_lock);
                CWARN("no group lock held\n");
                RETURN(-EINVAL);
        }

        LASSERT(fd->fd_grouplock.lg_lock != NULL);
```

- Did something reset `fd_grouplock` to NULL without touching the flag?
- Or did something modify `fd_flags` without setting the grouplock?
- . . . There's another ioctl that lets users set any value in `fd->fd_flags`, without any check
- Opened LU-13257 for this problem, found just for this talk!

Another problem, context.

- Going to server side
- There is no hard crash here, we pretend server became unresponsive and an admin or HA triggered a crash.
- What can we look at ?

Lancer crash


```
$ cd /var/crash/server_load
$ crash /usr/lib/debug/lib/modules/3.10.0*/vmlinux vmcore
...
      PANIC: "SysRq : Trigger a crash"
        PID: 4715
    COMMAND: "bash"
...
crash>
```

Crash notes

- No “BUG” crash: SysRq crash
- Let’s see what was running...

Analysis - ps


```
crash> ps
   PID    PPID  CPU       TASK        ST  %MEM     VSZ     RSS  COMM
      0      0   0  ffffffff9da18480  RU   0.0       0       0  [swapper/0]
>     0      0   1  ffff9177f60a4100  RU   0.0       0       0  [swapper/1]
...
   5387      2   4  ffff917cbba30000  IN   0.0       0       0  [ll_ost02_006]
   5388      2   5  ffff917cbba36180  IN   0.0       0       0  [mdt02_003]
>  5389      2   0  ffff917cbba330c0  RU   0.0       0       0  [ll_ost00_003]
...
   5532      2   3  ffff917c6a9b1040  RU   0.0       0       0  [ll_ost_io01_006]
   5533      2   6  ffff917c6a9b2080  IN   0.0       0       0  [mdt03_011]
   5534      2   0  ffff917c6a9b30c0  IN   0.0       0       0  [mdt00_015]
   5535      2   7  ffff917c6a9b4100  IN   0.0       0       0  [mdt03_012]
   5536      2   5  ffff917c6a9b5140  UN   0.0       0       0  [ll_ost_io02_006]
...
```

Crash notes

>, IN, UN, RU	Active, Interruptible, Uninterruptible, Runnable
ZO, ST, TR, DE	Zombie, Stopped, Tracing, Dead
SW, WA, PA, NE, EX...	Swapping, Waking, Parked, New, Exclusive

Analysis - `ps -l`


```
crash> ps -l
[1834107714279] [IN] PID: 4831 TASK: ffff9..b230c0 CPU: 7 COMMAND: "socknal_sd03_01"
[1834107636683] [IN] PID: 4828 TASK: ffff9..b25140 CPU: 5 COMMAND: "socknal_sd02_00"
[1834107619697] [RU] PID: 6121 TASK: ffff9..8f6180 CPU: 2 COMMAND: "ll_ost_io01_042"
[1834107601107] [IN] PID: 4829 TASK: ffff9..b22080 CPU: 4 COMMAND: "socknal_sd02_01"
...
[1833607166340] [RU] PID: 6032 TASK: ffff9..f65140 CPU: 3 COMMAND: "ll_ost_io01_023"
[1833600488498] [UN] PID: 6225 TASK: ffff9..b44100 CPU: 2 COMMAND: "ll_ost_io01_068"
[1833568488187] [RU] PID: 5532 TASK: ffff9..9b1040 CPU: 3 COMMAND: "ll_ost_io01_006"
[1833531745921] [RU] PID: 6129 TASK: ffff9..208000 CPU: 3 COMMAND: "ll_ost_io01_044"
[1833522652478] [RU] PID: 6239 TASK: ffff9..aec100 CPU: 3 COMMAND: "ll_ost_io01_074"
[1833501019679] [RU] PID: 6094 TASK: ffff9..1a0000 CPU: 3 COMMAND: "ll_ost_io01_035"
[1833456118992] [RU] PID: 6214 TASK: ffff9..2b8000 CPU: 3 COMMAND: "ll_ost_io01_066"
```

Crash notes

- First column (brackets) contains the last scheduled timestamp of each task (in nanosecond from boot)
- Processes are sorted on this column: helps find group of stuck threads
- $1834107714279 - 1833456118992 = 652ms$
  - dmesg will show traces from 120s in UN or RU state there, this case isn't realistic
- `dmesg | tail` -> `[ 1834.107266] ...`

Analysis - foreach bt


```
crash> foreach bt > foreach_bt
crash> ! awk '/PID:/ {PID=$2;PNAME=$8}
             /^ ?#/ {print PNAME": "$1" "$3" "$5}' foreach_bt |
         clubak -c
---------------
"ll_ost00_003"
---------------
...
 #7 queued_spin_lock_slowpath ffffffff9d55d28b
 #8 _raw_spin_lock ffffffff9d56b760
 #9 lock_res_and_lock ffffffffc0fcc02c
 #10 ldlm_work_bl_ast_lock ffffffffc0fcf0d9
 #11 ptlrpc_check_set ffffffffc1014030
 #12 ptlrpc_check_set ffffffffc101587b
 #13 ptlrpc_set_wait ffffffffc1015c24
 #14 ldlm_run_ast_work ffffffffc0fd3055
 #15 ldlm_handle_conflict_lock ffffffffc0fd3780
 #16 ldlm_lock_enqueue ffffffffc0fd3cc3
 #17 ldlm_handle_enqueue0 ffffffffc0ffc336
```

Crash notes

- tons of possible arguments: regex (`'mdt0.*'`), state (`active`, `RU`, `UN`)...
- `foreach bt -R lock_res_and_lock`, `foreach bt -FF`, `foreach files`...

Analysis - `bt -FF`


```
crash> mod -S > /dev/null
crash> bt -FF 5389
PID: 5389   TASK: ffff917cbba330c0  CPU: 0   COMMAND: "ll_ost00_003"
...
#14 [ffff917c9eb33b40] ldlm_run_ast_work at ffffffffc0fd3055 [ptlrpc]
    ffff917c9eb33b48: [ffff917c62522f40:kioctx] [ffff917c981c3b40:kmalloc-192]
    ffff917c9eb33b58: [ffff917c981c3b5c:kmalloc-192] ffff917c9eb33c68
    ffff917c9eb33b68: ffff917c9eb33ba0 ldlm_handle_conflict_lock+112
#15 [ffff917c9eb33b70] ldlm_handle_conflict_lock at ffffffffc0fd3780 [ptlrpc]
    ffff917c9eb33b78: ffff917c9eb33c68 0000000000000000
    ffff917c9eb33b88: ffff917c9eb33bd8 [ffff917c981c3b40:kmalloc-192]
    ffff917c9eb33b98: [ffff917c62522f40:kioctx] ffff917c9eb33c18
    ffff917c9eb33ba8: ldlm_lock_enqueue+723
#16 [ffff917c9eb33ba8] ldlm_lock_enqueue at ffffffffc0fd3cc3 [ptlrpc]
    ffff917c9eb33bb0: 000000008058614a [ffff917cb168a430:kmalloc-2048]
    ffff917c9eb33bc0: ffff91772a3780e0 ldlm_process_extent_lock
    ffff917c9eb33bd0: 00000000c111e5a0 [ffff917c59d6be88:kioctx]
    ffff917c9eb33be0: [ffff917c6059e808:kioctx] 000000008058614a
    ffff917c9eb33bf0: [ffff917cb168a050:kmalloc-2048] ffff91772a3780e0
    ffff917c9eb33c00: 0000000000000000 [ffff917cdd6a2000:kmalloc-1024]
    ffff917c9eb33c10: [ffff917cf510fc40:kmalloc-64] ffff917c9eb33ca8
    ffff917c9eb33c20: ldlm_handle_enqueue0+2646
#17 [ffff917c9eb33c20] ldlm_handle_enqueue0 at ffffffffc0ffc336 [ptlrpc]
    ffff917c9eb33c28: ffff917700000000 ffff917700000000
    ffff917c9eb33c38: lustre_swab_ldlm_request [ffff917cb168a430:kmalloc-2048]
    ffff917c9eb33c48: 0000006800000001 [ffff917c62522f40:kioctx]
    ffff917c9eb33c58: 0000000000000038 [ffff917cb168a430:kmalloc-2048]
    ffff917c9eb33c68: 0000000000020000 [ffff917c62522f40:kioctx]
    ffff917c9eb33c78: 000000008058614a [ffff917c5ead9960:sigqueue]
    ffff917c9eb33c88: [ffff917cb168a050:kmalloc-2048] [ffff917ca0e6a110:kmalloc-2048]
    ffff917c9eb33c98: [ffff917cb168a050:kmalloc-2048] tgt_dlm_handlers
...
```

Analysis - `struct -o`


```
crash> set radix 16
crash> struct -o ldlm_resource
struct ldlm_resource {
   [0x0] struct ldlm_ns_bucket *lr_ns_bucket;
   [0x8] struct hlist_node lr_hash;
  [0x18] atomic_t lr_refcount;
  [0x1c] spinlock_t lr_lock;
  [0x20] struct list_head lr_granted;
  [0x30] struct list_head lr_waiting;
  [0x40] struct ldlm_res_id lr_name;
...
  [0x70] enum ldlm_type lr_type;
  [0x74] int lr_lvb_len;
  [0x78] struct mutex lr_lvb_mutex;
  [0xa0] void *lr_lvb_data;
  [0xa8] _Bool lr_lvb_initialized;
  [0xa9] struct lu_ref lr_reference;
}
SIZE: 0xb0
```

Crash notes

- 0*xb*0 = 176: look for kmalloc-192
- Only one “round” value matching in earlier bt -FF. . .

Analysis - `struct ldlm_resource`


```
crash> struct ldlm_resource ffff917c981c3b40
  lr_ns_bucket = 0xffffb40a6ea1d018,
...
  lr_refcount = {
    counter = 0x145d7
  },
...
  lr_granted = {
    next = 0xffff917c6648f420,
    prev = 0xffff91772946b1e0
  },
  lr_waiting = {
    next = 0xffff917c62522fa0,
    prev = 0xffff917c867b0060
  },
  lr_name = {
    name = {0x28, 0x0, 0x0, 0x0}
  },
...
  lr_type = LDLM_EXTENT,
  lr_lvb_len = 0x38,
```

Crash notes

`lr_name` often a fid, but on OST here it is an oid
`lr_refcount` refcount (duh) associated with the resource
`lr_granted/waiting` lists of locks

Analysis - `struct ldlm_resource` 2


```
crash> struct ldlm_resource | grep lr_ns_bucket
    struct ldlm_ns_bucket *lr_ns_bucket;
crash> struct ldlm_ns_bucket 0xffffb40a6ea1d018
  nsb_namespace = 0xffff917cdd6a2000,
...
crash> struct ldlm_ns_bucket | grep nsb_namespace
    struct ldlm_namespace *nsb_namespace;
crash> struct ldlm_namespace 0xffff917cdd6a2000
...
  ns_name = 0xffff917ce5b5ae60 "filter-testfs0-OST0000_UUID",
```

Crash notes

- Going through multiple structures is a bit painful, but can be done

Reminders on ldiskfs


```
# mount -o loop -t ldiskfs /tmp/ost0 /media/
# ls /media/O/0/d$((40%32))/40
/media/O/0/d8/40
# umount /media
# debugfs -c /tmp/ost0
...
debugfs:  stat /O/0/d8/40
...
  fid: parent=[0x200000401:0x4d:0x0] stripe=0
       stripe_size=1048576 stripe_count=1
...
^D
# lfs fid2path /mnt/lustre 0x200000401:0x4d:0x0
/mnt/lustre/test
```

Reminders on kernel linked list


```
crash> help list
crash> struct list_head
struct list_head {
    struct list_head *next;
    struct list_head *prev;
}
SIZE: 16
```

Notes

- double-linked list
- Often used with an independant: `list -H`
  - i.e. `lustre/ldlm/ldlm_internal.h:38: extern struct list_head ldlm_srv_namespace_list;`
  - `/!\` same type = sometimes hard to tell appart

Parenthèse linked list kernel


```
crash> list -H ldlm_srv_namespace_list
ffff917cdd6a2000
...
```

Usage

```
        list_for_each_entry(ns, ldlm_namespace_list(LDLM_NAMESPACE_SERVER),
                            ns_list_chain) {
```

```
crash> list -s ldlm_namespace ldlm_namespace.ns_list_chain
            -H ldlm_srv_namespace_list
ffff917cdd6a2000
struct ldlm_namespace {
  ns_obd = 0xffff917ca0d620f0,
  ns_client = LDLM_NAMESPACE_SERVER,
  ns_name = 0xffff917ce5b5ae60 "filter-testfs0-OST0000_UUID",
  ns_rs_hash = 0xffff917ce0680000,
...
```

Analysis - `struct ldlm_resource` 3


```
crash> struct -o ldlm_resource | grep -E 'lr_granted|lr_waiting'
  [0x20] struct list_head lr_granted;
  [0x30] struct list_head lr_waiting;
crash> struct -d ldlm_resource.lr_refcount ffff917c981c3b40
  lr_refcount = {
    counter = 83415
  }
crash> list -H ffff917c981c3b60 | wc -l
82718
crash> list -H ffff917c981c3b70 | wc -l
3
```

Crash notes

- Here (read code!) `lr_granted`/`lr_waiting` are similar list “heads”
- Links to `struct ldlm_lock` on field `l_res_link`

Analysis - `struct ldlm_lock`


```
crash> struct -o ldlm_resource
struct ldlm_lock {
...
   [0x48] struct ldlm_resource *l_resource;
   [0x60] struct list_head l_res_link;
   [0x98] enum ldlm_mode l_req_mode;
   [0x9c] enum ldlm_mode l_granted_mode;
   [0xb8] struct obd_export *l_export;
  [0x100] __u64 l_flags;
          union {
  [0x160]     time64_t l_activity;
  [0x160]     time64_t l_blast_sent;
          };
  [0x1c0] __u32 l_pid;
  [0x1f8] struct ldlm_lock *l_blocking_lock;
...
}
SIZE: 0x230
```

Analysis - `struct ldlm_lock` 2


```
crash> list -H ffff917c981c3b70 ldlm_lock.l_res_link
           -s ldlm_lock.l_req_mode
ffff917c62522f40
  l_req_mode = LCK_PW
ffff917c641e5f80
  l_req_mode = LCK_PR
crash> list -H ffff917c981c3b60 ldlm_lock.l_res_link
           -S ldlm_lock.l_req_mode | grep -B 1 'l_req_mode = 0x2'
ffff917c6648f3c0
  l_req_mode = 0x2
ffff917c96fdfcc0
  l_req_mode = 0x2
...
```

Crash notes

- `-S` is faster than `-s`, but no symbol resolution
- multiple `LCK_PW` granted but most are `PR`, should be incompatible but `l_req_extent` are different: work on independant ranges.

Analysis - `struct ldlm_lock` 3 - find a nid

Analysis - `struct ldlm_lock` 3 - find a nid


```
crash> list -H ffff917c981c3b70 ldlm_lock.l_res_link
            -S ldlm_lock.l_export
ffff917c62522f40
  l_export = 0xffff917c96f59000
```

Analysis - `struct ldlm_lock` 3 - find a nid


```
crash> list -H ffff917c981c3b70 ldlm_lock.l_res_link
            -S ldlm_lock.l_export
ffff917c62522f40
  l_export = 0xffff917c96f59000
```

```
crash> struct obd_export.exp_connection 0xffff917c96f59000
  exp_connection = 0xffff917cdee0ba80
```

Analysis - `struct ldlm_lock` 3 - find a nid


```
crash> list -H ffff917c981c3b70 ldlm_lock.l_res_link
            -S ldlm_lock.l_export
ffff917c62522f40
  l_export = 0xffff917c96f59000
```

```
crash> struct obd_export.exp_connection 0xffff917c96f59000
  exp_connection = 0xffff917cdee0ba80
```

```
crash> struct ptlrpc_connection 0xffff917cdee0ba80
struct ptlrpc_connection {
  c_peer = {
    nid = 0x200000ac80002,
    pid = 0x3039
  },
```

Crash notes

- nid = lnd type, lnd number, ip/id
  - Here: tcp, 0, `0a.c8.00.02` = `10.200.0.2`
  - o2ib: 0005001a0a800002
  - ptlf: 000f001500000014
- `net -N 0x200000ac80002` → `2.0.200.10`

Analysis - mass processing


```
crash> list -H ffff917c981c3b60 ldlm_lock.l_res_link
       -S ldlm_lock.l_export | awk '/l_export/ { print $3 }'
           > export
crash> list -H ffff917c981c3b70 ldlm_lock.l_res_link
       -S ldlm_lock.l_export | awk '/l_export/ { print $3 }'
           >> export
crash> !head -n 1 export
0xffff917c96f59000
crash> struct obd_export.exp_connection 0xffff917c96f59000
  exp_connection = 0xffff917cdee0ba80
crash> struct -o obd_export.exp_connection
struct obd_export {
  [0x118] struct ptlrpc_connection *exp_connection;
}
crash> rd -o 0x118 0xffff917c96f59000
ffff917c96f59118:  ffff917cdee0ba80                    ....|...
crash> rd -o 0x118 < export | awk '{ print $2 }' > conn
```

Analysis - mass processing 2


```
crash> struct -o ptlrpc_connection.c_peer
struct ptlrpc_connection {
  [0x18] struct lnet_process_id c_peer;
}
crash> !head conn
ffff917cdee0ba80
crash> struct ptlrpc_connection.c_peer.nid ffff917cdee0ba80
  c_peer.nid = 0x200000ac80002,
crash> rd -o 0x18 ffff917cdee0ba80
ffff917cdee0ba98:  000200000ac80002                        ........
crash> rd -o 0x18 < conn | awk '{ print $2 }' |
           sort | uniq -c | sort -h
    998 000200000ac8002c
   1019 000200000ac80041
   1024 000200000ac80024
   1029 000200000ac8002f
...
   1435 000200000ac80006
   1436 000200000ac8001e
   1445 000200000ac80016
```

Analyse - mass processing 3

Notes crash

- Using rd instead of `struct foo.bar` to save a lot of time as there is very little cache in crash (will do many gdb requests ; there is a patch in progress to implement something similar to `list`'s `-S` vs. `-s` option)
- Abuse external files and external commands
- Some extensions like `crash-extscript` can help if needed, but you get used to this...

Analysis - `struct ptlrpc_request`


```
crash> struct -o ptlrpc_request
struct ptlrpc_request {
    [0x0] int rq_type;
    [0x4] int rq_status;
...
   [0x48] struct lustre_msg_v2 *rq_reqmsg;
   [0x50] struct lustre_msg_v2 *rq_repmsg;
   [0x58] __u64 rq_transno;
   [0x60] __u64 rq_xid;
   [0x68] __u64 rq_mbits;
...
  [0x390] struct obd_export *rq_export;
  [0x398] struct obd_import *rq_import;
  [0x3a0] lnet_nid_t rq_self;
  [0x3a8] struct lnet_process_id rq_peer;
  [0x3b8] struct lnet_process_id rq_source;
  [0x3c8] time_t rq_timeout;
  [0x3d0] time64_t rq_sent;
  [0x3d8] time64_t rq_deadline;
  [0x3e0] struct req_capsule rq_pill;
}
SIZE: 0x450
```

- 0*x*450 = 1104 : kmalloc-2048 or `ptlrpc_cache`
- Not much easily exploitable except the `rq_peer` field

Analysis - `struct ptlrpc_request` 2


```
crash> struct ptlrpc_request ffff917cb168a050
```

```
...
  rq_export = 0xffff917c96f59000,
  rq_import = 0x0,
  rq_self = 0x200000ac80001,
  rq_peer = {
    nid = 0x200000ac80002,
    pid = 0x3039
  },
  rq_source = {
    nid = 0x200000ac80002,
    pid = 0x3039
  },
  rq_timeout = 0x0,
  rq_sent = 0x0,
...
```

Description

- `ls` stuck on a user dir while I was writing these slides. . . .
- Quick glance: You should be able to make similar observations now!

foreach bt


```
---------------                                  ---------------
"mdt00_069"/18640,"mdt00_100"/18689,             "mdt01_033"/25786
"mdt00_117"/18714,"mdt01_041"/25805,             ---------------
"mdt01_058"/18667,"mdt01_088"/18734,              #0 __schedule ffffffff97768a32
"mdt01_109"/7506 (7)                              #1 schedule ffffffff97768ed9
---------------                                   #2 ldlm_completion_ast ffffffffc0ea9aac
 #0 __schedule ffffffff97768a32                   #3 ldlm_cli_enqueue_local ffffffffc0eaab63
 #1 schedule ffffffff97768ed9                     #4 mdt_object_local_lock ffffffffc14d1dc2
 #2 ldlm_completion_ast ffffffffc0ea9aad          #5 mdt_object_lock_internal ffffffffc14d240
 #3 ldlm_cli_enqueue_local ffffffffc0eaab63       #6 mdt_reint_object_lock ffffffffc14d26f0
 #4 mdt_object_local_lock ffffffffc14d1dc2        #7 mdt_reint_unlink ffffffffc14e84b2
 #5 mdt_object_lock_internal ffffffffc14d240e     #8 mdt_reint_rec ffffffffc14ebc53
 #6 mdt_getattr_name_lock ffffffffc14d31b4        #9 mdt_reint_internal ffffffffc14cd18b
 #7 mdt_intent_getattr ffffffffc14d44d0           #10 mdt_reint ffffffffc14d8fa7
 #8 mdt_intent_opc ffffffffc14d04bb               #11 tgt_request_handle ffffffffc0f427ca
 #9 mdt_intent_policy ffffffffc14d8d58            #12 ptlrpc_server_handle_request ffffffffc0
 #10 ldlm_lock_enqueue ffffffffc0e8f2dd           #13 ptlrpc_main ffffffffc0eee7a2
 #11 ldlm_handle_enqueue0 ffffffffc0eb8c03        #14 kthread ffffffff970c1da1
 #12 tgt_enqueue ffffffffc0f3e892
 #13 tgt_request_handle ffffffffc0f427ca
 #14 ptlrpc_server_handle_request ffffffffc0eeb05b
 #15 ptlrpc_main ffffffffc0eee7a2
 #16 kthread ffffffff970c1da1
```

ps -l


```
crash> ps -l
[3621688788852009] [IN]  PID: 28644  TASK: ffff94a47bf84100  CPU: 2   COMMAND: "mdt00_013"
... 1700s ago
[3619975098195335] [IN]  PID: 6743   TASK: ffff94813e660000  CPU: 7   COMMAND: "kworker/7:1"
[3619964995355868] [IN]  PID: 18640  TASK: ffff9479eebb30c0  CPU: 0   COMMAND: "mdt00_069"
[3619922465196662] [IN]  PID: 24127  TASK: ffff94a47e22a080  CPU: 14  COMMAND: "mdt_seqm_0002"
... 6336s ago
[3615352056840100] [IN]  PID: 29072  TASK: ffff94a39afa6180  CPU: 15  COMMAND: "kworker/15:0"
[3615187577148168] [IN]  PID: 18714  TASK: ffff94a4785f9040  CPU: 0   COMMAND: "mdt00_117"
[3615157566040146] [IN]  PID: 18689  TASK: ffff94847fd6b0c0  CPU: 3   COMMAND: "mdt00_100"
[3614626563207010] [IN]  PID: 18667  TASK: ffff94a47e5f5140  CPU: 12  COMMAND: "mdt01_058"
[3614100469065774] [IN]  PID: 27971  TASK: ffff948472a8e180  CPU: 10  COMMAND: "cfs_rh_03"
[3614025495970123] [IN]  PID: 25805  TASK: ffff94847d738000  CPU: 8   COMMAND: "mdt01_041"
[3612352026539768] [IN]  PID: 3482   TASK: ffff94a47ebc0000  CPU: 8   COMMAND: "kworker/8:1"
[3612142134798789] [IN]  PID: 30482  TASK: ffff949e91689040  CPU: 11  COMMAND: "kworker/11:0"
[3612035313324469] [IN]  PID: 7506   TASK: ffff94a3a26cb0c0  CPU: 12  COMMAND: "mdt01_109"
[3611323248826132] [IN]  PID: 18734  TASK: ffff94a3e3e81040  CPU: 11  COMMAND: "mdt01_088"
[3611260896608780] [IN]  PID: 21311  TASK: ffff949ecd242080  CPU: 8   COMMAND: "kworker/u34:1"
[3608862966498563] [IN]  PID: 25786  TASK: ffff947dadf61040  CPU: 12  COMMAND: "mdt01_033"
[3605752338202911] [IN]  PID: 32477  TASK: ffff949f242c6180  CPU: 12  COMMAND: "kworker/12:1"
...
```

- stuck almost two hours
- very bottom `mdt01_033` was the unlink

Find the file

- bt -FF on stuck thread
  - (no luck) unlink got unstuck just as I was doing that, didn't get traces.
- But had stuck folder + changelogs: only one unlink there at this hour
- There also was an HSM restore on this file... Finished precisely when things got unstuck.
  - `ls` and `getattr`s were waiting for the unlink (lock on directory), which was waiting for the file (lock on file) with directory lock held
  - Pretty poor user experience as the whole folder got stuck...

Going further

- Many commands we didn't go through here:
  - dis(assemble), kmem, mount, dev, net, rd, bpf, etc. . .
- Learn how to read backtraces in more details
  - There is no need to guess what is what from slabs with a little bit of reflexion
- crash extensions:
  - There is a lustre extension that can get dk buffer amongst other things
  - https://people.redhat.com/anderson/extensions.html

perf probe

- Setup tracepoints just about anywhere in the kernel (or userspace!)

perf probe

- Setup tracepoints just about anywhere in the kernel (or userspace!)
- For this example, we want to dump all locks from a filesystem

perf probe

- Setup tracepoints just about anywhere in the kernel (or userspace!)
- For this example, we want to dump all locks from a filesystem
- A friend brought up `/sys/kernel/debug/lustre/ldlm/dump_namespaces`
  - echo 1 in that file and lustre will dump all resources and locks in `lctl dk` if `dlmtrace` is enabled
  - but the dk buffer is too small (even if we grow it, bug?) and we cannot get full list
- Let's work around that problem with perf!

ldlm dump_namespaces - `perf probe -L`


```
# modprobe lustre
# perf probe -m ptlrpc -L ldlm_namespace_dump
...
     12          cfs_hash_for_each_nolock(ns->ns_rs_hash,
                                          ldlm_res_hash_dump,
     14                                   (void *)(unsigned long)leve
...
# perf probe -m ptlrpc -L ldlm_res_hash_dump
...
      7           ldlm_resource_dump(level, res);
...
# perf probe -m ptlrpc -L ldlm_resource_dump
```

Perf notes

- Module has to be loaded
- Can look at the code directly
- Can only hook on lines with a line number

ldlm dump_namespaces - `perf probe -V`


```
# perf probe -m ptlrpc -V ldlm_res_hash_dump:8
Available variables at ldlm_res_hash_dump:8
        @<ldlm_res_hash_dump+59>
                (unknown_type  __sp
                (unknown_type  arg
                int     level
                raw_spinlock_t* lock
                spinlock_t*     lock
                struct ldlm_resource*   res
                struct qspinlock*       lock
```

Perf notes

- Could pick different line numbers here
- But all lines won't provide the same variables, even if for C code they would!
- Some lines have multiple locations in the generated binary
  - A bit less efficient for perf (one probe per location)

ldlm dump_namespaces - `perf probe`


```
# perf probe -m ptlrpc ldlm_res_hash_dump:8 res \
             'res->lr_refcount' 'res->lr_name.name[0]'
```

Perf notes

- Can dump variables together with the probe
  - Multiple variables
  - Fields within a struct (type info from debuginfos)
- No complex type (cannot dump full struct)

```
# perf probe -v ...
Opening /sys/kernel/debug/tracing//kprobe_events write=1
Writing event: p:probe/ldlm_res_hash_dump
        ptlrpc:ldlm_res_hash_dump+59 res=%r12:u64
        lr_refcount=+24(%r12):u32 name=+64(%r12)
```

ldlm dump_namespaces - `perf record`


```
# hostname vm0 # FIXME
# shine -L start; shine -L mount; echo foo > /mnt/lustre/foo
# lctl set_param debug=+dlmtrace
# perf record -e probe:ldlm_res_hash_dump sh -c \
      'echo 1 > /sys/kernel/debug/lustre/ldlm/dump_namespaces'
# lctl set_param debug=-dlmtrace; lctl dk > /dev/null
```

Perf notes

- Register one or more probe (`-e`, accepts wildcards, comma separated list. . .)
- Register a single command or everything (`-a`)
- Backtraces at probe point (`-g`)
- Plenty of other filters, see `perf record --help`!

ldlm dump_namespaces - `perf script`


```
# perf script
sh 6847 [009] 1312.402512: probe:ldlm... res=0xffffa0a931bfb500 lr_refcount=0x2 name=0x42
sh 6847 [009] 1312.415683: probe:ldlm... res=0xffffa0a8f36d8480 lr_refcount=0x2 name=0x2000013a1
sh 6847 [009] 1312.439815: probe:ldlm... res=0xffffa0af8134a9c0 lr_refcount=0x2 name=0x70616d65646f6e
sh 6847 [009] 1312.439846: probe:ldlm... res=0xffffa0af8134a3c0 lr_refcount=0x6 name=0x736d61726170
sh 6847 [009] 1312.439890: probe:ldlm... res=0xffffa0af8134a0c0 lr_refcount=0xa name=0x30736674736574
sh 6847 [009] 1312.439905: probe:ldlm... res=0xffffa0a931f69080 lr_refcount=0x4 name=0x30736674736574
sh 6847 [009] 1312.439911: probe:ldlm... res=0xffffa0af8134b500 lr_refcount=0x2 name=0x30736674736574
sh 6847 [009] 1312.439924: probe:ldlm... res=0xffffa0a92e792180 lr_refcount=0x2 name=0x70616d65646f6e
sh 6847 [009] 1312.439950: probe:ldlm... res=0xffffa0af80c5f8c0 lr_refcount=0x6 name=0x736d61726170
sh 6847 [009] 1312.439985: probe:ldlm... res=0xffffa0a92e792600 lr_refcount=0xa name=0x30736674736574
sh 6847 [009] 1312.439999: probe:ldlm... res=0xffffa0af7ee620c0 lr_refcount=0x4 name=0x30736674736574
sh 6847 [009] 1312.440006: probe:ldlm... res=0xffffa0a92e7935c0 lr_refcount=0x2 name=0x30736674736574
sh 6847 [009] 1312.440106: probe:ldlm... res=0xffffa0a8f36d8840 lr_refcount=0x2 name=0x2000013a1
sh 6847 [009] 1312.441070: probe:ldlm... res=0xffffa0a8f36d83c0 lr_refcount=0x2 name=0x42
```

Perf notes

- All ldlm namespaces are listed (OST*, MDT*, MGS)
- `0x42`: oid on OST, `0x2000013a1`: start of fid (MDT or OST), `0x30736674736574`: "`testfs0`" (MGT)...

ldlm dump_namespaces - `perf probe -d`


```
# perf probe -d '*'
```

ldlm dump_namespaces - recap


```
# perf probe -m ptlrpc ldlm_res_hash_dump:8 res \
            'res->lr_refcount' 'res->lr_name.name[0]'
# lctl set_param debug=+dlmtrace
# perf record -o /tmp/perf.data -e probe:* sh -c \
      'echo 1 > /sys/kernel/debug/lustre/ldlm/dump_namespaces'
# lctl set_param debug=-dlmtrace; lctl dk > /dev/null
# perf script -i /tmp/perf.data | \
    sed -ne "s/.*lr_refcount=(0x[9a-f][0-9a-f]\{2\}|
             [0-9a-f]\{4,\}) name=(0x[0-9a-f]*).*/\1 \2/p" | \
    while read refcount oid; do
      debugfs -c /dev/md[0-1]
              -R "stat /O/0/d$((oid%32))/$((oid))" 2>&1 | \
        sed -ne "s/.*fid: parent=([^ ]*).*/\1 $((refcount))/p";
    done | sort -k 2nr
```

flame graphs / off-cpu flame graph

- No time to get into it in detail
- Have a look at Brendan Gregg website!
  - cpu flame graphs[1]
  - off-cpu flame graphs[2]

[1] http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html
[2] http://www.brendangregg.com/FlameGraphs/offcpuflamegraphs.html

flame graph

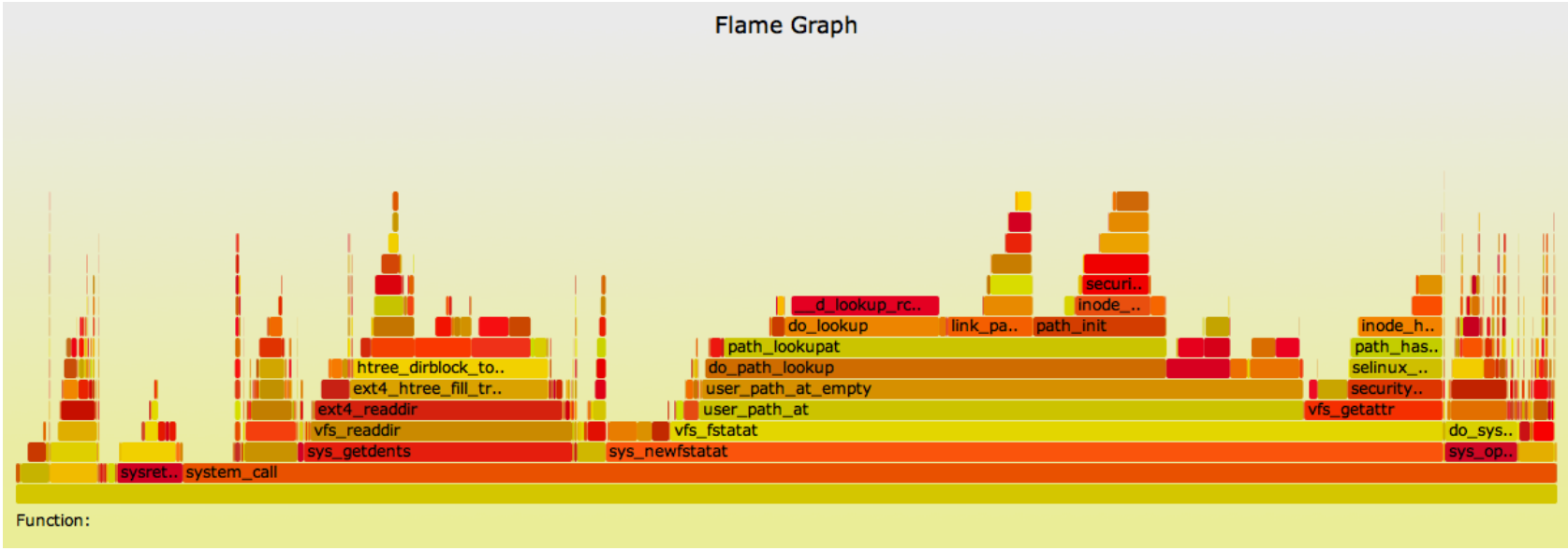

Figure 1: Exemple de flame graph

SystemTap

- More powerful and intrusive than perf
- Compiles a small kernel module and inserts it: can break everything
  - Which means we can use it to workaroun issues as well!

Example 1 - modify return value


```
#!/usr/bin/stap -g

global EIO = -5
global EAGAIN = -11

probe begin {
    print("LU-5642 stap started. Press ^C to exit\n")
}

function syslog(msg: string) %{
    printk("stap lu_5642: %s\n", STAP_ARG_msg);
%}

probe module("lustre").function("ll_xattr_cache_refill").return {
    if ($return == EIO && uid() > 1000) {
        slurm_jobid = env_var("SLURM_JOBID")
        syslog(sprintf("JOBID %s: %s(%d) got EIO, changing to EAGAIN"
                       " (inode %p)",
                       slurm_jobid, execname(), pid(), $inode))
        $return = EAGAIN
    }
}
```

Example 1 - run probe


```
# stap -g -v lu_5642.stp
Pass 1: parsed user script and 113 library script(s) using 220336virt/37716res
        /3048shr/35148data kb, in 200usr/20sys/212real ms.
Pass 2: analyzed script: 4 probe(s), 32 function(s), 3 embed(s), 0 global(s) using
        319160virt/137688res/4208shr/133972data kb, in 1790usr/700sys/2448real ms.
Pass 3: translated to C into "/tmp/stap7H7tB3/stap_46c9da9de15aa2342b70f3ccee7b96d8_
        20032_src.c" using 319160virt/137992res/4512shr/133972data kb, in
        70usr/70sys/139real ms.
Pass 4: compiled C into "stap_46c9da9de15aa2342b70f3ccee7b96d8_20032.ko" in 8550usr
        /1540sys/10158real ms.
Pass 5: starting run.
LU-5642 stap started. Press ^C to exit
```

- Or copy .ko file in `/lib/modules/$(uname -r)/systemtap` and `staprun <modulename>` in a service.

Example 1 - introspection

Types de probes

```
# stap --dump-probe-types
...
kernel.function(string)
...
module(string).function(string)
module(string).function(string).return
module(string).statement(number)
...
```

Common probes

```
# stap --dump-probe-aliases
```

Example 1 - introspection

Liste des functions

```
# stap -l 'module("lustre").function("*xattr*")'
...
module("lustre").function("ll_xattr_cache_refill@/usr/src
    /debug/lustre-2.12.3/lustre/llite/xattr_cache.c:376")
...
```

Variables disponibles

```
# stap -L 'module("lustre").function("ll_xattr_cache_refill")'
module("lustre").function("ll_xattr_cache_refill
 @/usr/src/debug/lustre-2.12.3/lustre/llite/xattr_cache.c:376")
$inode:struct inode* $oit:struct lookup_intent
$req:struct ptlrpc_request* $__func__:char[] const
```

Example 2 - debug

- More powerful than perf to dump full types: `$var$$`
- Let's do the same as the earlier perf probe...

Example 2 - debug

- More powerful than perf to dump full types: `$var$$`
- Let's do the same as the earlier perf probe...

```
# stap -L 'module("ptlrpc").function("ldlm_resource_dump")'
module("ptlrpc").function("ldlm_resource_dump@/usr/src
  /debug/lustre-2.12.3/lustre/ldlm/ldlm_resource.c:1763")
$level:int $res:struct ldlm_resource*
$granted:unsigned int $__func__:char[] const
```

Example 2 - debug

- More powerful than perf to dump full types: `$var$$`
- Let's do the same as the earlier perf probe...

```
# stap -L 'module("ptlrpc").function("ldlm_resource_dump")'
module("ptlrpc").function("ldlm_resource_dump@/usr/src
  /debug/lustre-2.12.3/lustre/ldlm/ldlm_resource.c:1763")
$level:int $res:struct ldlm_resource*
$granted:unsigned int $__func__:char[] const
```

```
probe begin {
    print("probe started, hit ^C to stop\n")
}

probe module("ptlrpc").function("ldlm_resource_dump") {
    print($res$$, "\n")
}
```

Example 2 - debug


```
# stap probe.stp
probe started, hit ^C to stop
{.lr_ns_bucket=0xffffa0a931614518, .lr_hash={.next=0x0, .pprev=0xffffa0a931614438},
.lr_refcount={.counter=2}, .lr_lock={<union>={.rlock={.raw_lock={.val={.counter=1}}}}},
.lr_granted={.next=0xffffa0a9172542a0, .prev=0xffffa0a9172542a0},
.lr_waiting={.next=0xffffa0af8134a9f0, .prev=0xffffa0af8134a9f0},
.lr_name={.name=[31632319872003950, ...]}, <union>={.lr_itree=0x0,
.lr_ibits_queues=0x0}, <union>={.lr_contention_time=0, .lr_lvb_inode=0x0},
.lr_type=10, .lr_lvb_len=0, .lr_lvb_mutex={.count={.counter=1}, .

{.lr_ns_bucket=0xffffa0a931614518, .lr_hash={.next=0x0, .pprev=0xffffa0a931614448},
.lr_refcount={.counter=6}, .lr_lock={<union>={.rlock={.raw_lock={.val={.counter=1}}}}},
.lr_granted={.next=0xffffa0a917254ba0, .prev=0xffffa0a91529c960},
.lr_waiting={.next=0xffffa0af8134a3f0, .prev=0xffffa0af8134a3f0},
.lr_name={.name=[126913623515504, ...]}, <union>={.lr_itree=0x0,
.lr_ibits_queues=0x0}, <union>={.lr_contention_time=0, .lr_lvb_inode=0x0},
.lr_type=10, .lr_lvb_len=0, .lr_lvb_mutex={.count={.counter=1}, .wa
```

- Struct truncated at fixed size (512 chars)
- Array truncated (`lr_name...`)

- https://sourceware.org/systemtap/langref.pdf
- https://github.com/calio/systemtap-cheat-sheet

centos7 warning

- There's a bug with `asm_volatile_goto`, used in the kernel but not supported by clang[3] - fix not backported on el7

```
#ifdef asm_volatile_goto
#undef asm_volatile_goto
#endif
#define asm_volatile_goto(x...) \
    asm volatile("invalid use of asm_volatile_goto")
```

- Can add these lines to `arch/x86/include/asm/jump_label.h` as a workaround
- The provided VM comes with a newer version of bcc which has a proper fix; bcc was however only upgraded for bpftrace and all bcc-tools examples listed here work on centos7 version of bcc with just that workaround.

[3] https://github.com/iovisor/bcc/issues/2119

execsnoop


```
# /usr/share/bcc/tools/execsnoop
PCOMM        PID    PPID   RET ARGS
ls           13411  11518    0 /usr/bin/ls --color=auto
unix_chkpwd  13413  13412    0  root chkexpiry
bash         13414  13412    0 /bin/bash -c run-parts /etc/cron.hourly
run-parts    13414  13412    0 /bin/run-parts /etc/cron.hourly
basename     13416  13414    0 /bin/basename /etc/cron.hourly/0anacron
logger       13417  13414    0 /bin/logger -p cron.notice -t run-parts
...
```

opensnoop


```
# /usr/share/bcc/tools/opensnoop
```

```
13434 shine 7 0 /sys/fs/lustre
13434 shine 7 0 /proc/fs/lustre
13434 shine 7 0 /proc/fs/lustre
13434 shine 7 0 /proc/fs/lustre/mgs/MGS/mntdev
13434 shine 7 0 /proc/mounts
13434 shine 7 0 /proc/fs/lustre/lov
13434 shine 7 0 /proc/mounts
13434 shine 7 0 /proc/fs/lustre
13434 shine 7 0 /proc/fs/lustre/mdc
13434 shine 7 0 /proc/fs/lustre/osc
13434 shine 7 0 /proc/fs/lustre/mdc/testfs0-MDT0000-mdc-...6bc7800/state
13434 shine 7 0 /proc/fs/lustre/osc/testfs0-OST0000-osc-...6bc7800/state
13434 shine 7 0 /proc/fs/lustre/osc/testfs0-OST0001-osc-...6bc7800/state
13434 shine 7 0 /sys/fs/lustre
13434 shine 7 0 /proc/fs/lustre
13434 shine 7 0 /proc/fs/lustre
13434 shine 7 0 /proc/fs/lustre/osd-ldiskfs/testfs0-MDT0000/mntdev
13434 shine 7 0 /proc/mounts
13434 shine 7 0 /proc/fs/lustre/mdt/testfs0-MDT0000/recovery_status
...
```

.., and more


```
# ls /usr/share/bcc/tools/ | column -c 120
```

```
argdist         deadlock_detector   killsnoop       pythoncalls  syscount
bashreadline    deadlock_detector.c lib             pythonflow   tcpaccept
biolatency      doc                 llcstat         pythongc     tcpconnect
biosnoop        execsnoop           mdflush         pythonstat   tcpconnlat
biotop          ext4dist            memleak         reset-trace  tcpdrop
bitesize        ext4slower          mountsnoop      rubycalls    tcplife
bpflist         filelife            mysqld_qslower  rubyflow     tcpretrans
btrfsdist       fileslower          nfsdist         rubygc       tcpstates
btrfsslower     filetop             nfsslower       rubyobjnew   tcpsubnet
cachestat       funccount           nodegc          rubystat     tcptop
cachetop        funclatency         nodestat        runqlat      tcptracer
capable         funcslower          offcputime      runqlen      tplist
cobjnew         gethostlatency      offwaketime     runqslower   trace
cpudist         hardirqs            oomkill         slabratetop  ttysnoop
cpuunclaimed    javacalls           opensnoop       softirqs     vfscount
criticalstat    javaflow            phpcalls        solisten     vfsstat
dbslower        javagc              phpflow         sslsniff     wakeuptime
dbstat          javaobjnew          phpstat         stackcount   xfsdist
dcsnoop         javastat            pidpersec       statsnoop    xfsslower
dcstat          javathreads         profile         syncsnoop
```

- https://github.com/iovisor/bcc
- https://github.com/iovisor/bcc/blob/master/docs/reference_guide.md
- https://github.com/iovisor/bcc/blob/master/docs/tutorial.md

- Requires newer llvm/gcc/bcc...
- Installed here by abusing llvm-toolset-7 and devtoolset-8: https://medium.com/@tle211212/use-bpftrace-on-centos-7-6ab920624084
- Works out of the box on centos8

Exemples


```
# bpftrace -e 'kprobe:do_nanosleep {
                 printf("PID %d sleeping\n", pid);
               }'
Attaching 1 probe...
PID 3525 sleeping
PID 3525 sleeping
^C
# bpftrace -e 'tracepoint:raw_syscalls:sys_enter {
                 @[comm]=count();
               }'
Attaching 1 probe...
^C
@[gmain]: 2
@[tuned]: 4
@[in:imjournal]: 8
@[irqbalance]: 15
@[bpftrace]: 43
@[bash]: 113
@[qemu-ga]: 228
@[sshd]: 259
@[shine]: 6540
```

Exemples 2


```
# bpftrace -e 'tracepoint:syscalls:sys_enter_openat {
                printf("%s %s\n", comm, str(args->filename));
              }'
# bpftrace -e 'kprobe:do_sys_open {
                printf("%s %s\n", comm, str(arg1));
              }'
shine /etc/shine/shine.conf
shine /var/cache/shine/conf
shine /var/cache/shine/conf/testfs.xmf
...
shine /etc/hosts
shine /proc/fs/lustre/lov
```

ldlm dump_namespaces

- Once again, let's dump ldlm locks.
- bpftrace cannot guess types from debuginfos (it does have most basic types from eBPF kernel headers), so e.g. lustre types need to be redefined
  - On the good side, that means we do not need to install bulky debuginfo packages
- With 4.18+ kernels with `CONFIG_DEBUG_INFO_BTF=y` (BPF Type Format), bpftrace will be able to figure types out (not yet available in centos8.1, but e.g. fedora has it)
- Meanwhile we can redefine types or get offsets by hand...

```
# gdb /usr/lib/debug/lib/modules/3.10.0-957.21.3.el7.x86_64
      /extra/lustre/fs/ptlrpc.ko.debug --quiet --batch
  --ex 'p &((struct ldlm_resource*)0)->lr_refcount'
$1 = (atomic_t *) 0x18
# gdb /usr/lib/debug/lib/modules/3.10.0-957.21.3.el7.x86_64
      /extra/lustre/fs/ptlrpc.ko.debug --quiet --batch
  --ex 'p &((struct ldlm_resource*)0)->lr_name.name[0]'
$1 = (__u64 *) 0x40 <lock_res_and_lock+16>
```

ldlm dump_namespaces 2


```
# bpftrace -e 'kprobe:ldlm_resource_dump {
                printf("res %p, %d, 0x%llx:0x%llx:0x0\n",
                       arg1, *(arg1+0x18),
                       *(arg1+0x40), *(arg1+0x48))
              }'
```

```
Attaching 1 probe...
res 0xffff9a28e3644a80, 2, 0x2:0x0:0x0
res 0xffff9a2934376600, 2, 0x200000401:0x1:0x0
res 0xffff9a29b5739800, 2, 0x70616d65646f6e:0x4:0x0
res 0xffff9a29b5738c00, 5, 0x736d61726170:0x3:0x0
res 0xffff9a2906ea2d80, 9, 0x30736674736574:0x0:0x0
res 0xffff9a2906ea20c0, 3, 0x30736674736574:0x2:0x0
res 0xffff9a29b5738900, 2, 0x30736674736574:0x5:0x0
res 0xffff9a2fd3bf2000, 2, 0x70616d65646f6e:0x4:0x0
res 0xffff9a2fd3bf2240, 5, 0x736d61726170:0x3:0x0
res 0xffff9a23b48acb40, 9, 0x30736674736574:0x0:0x0
res 0xffff9a2e6fbd12c0, 3, 0x30736674736574:0x2:0x0
res 0xffff9a2fd3bf20c0, 2, 0x30736674736574:0x5:0x0
res 0xffff9a28e3644900, 2, 0x200000401:0x1:0x0
res 0xffff9a28e3644000, 2, 0x2:0x0:0x0
```

- More examples: https://github.com/iovisor/bpftrace
- https://github.com/iovisor/bpftrace/blob/master/docs/reference_guide.md
- https://github.com/iovisor/bpftrace/blob/master/docs/tutorial_one_liners.md

bcc tools internals

- bcc is just a BPF compiler:
  - Takes C code, outputs BPF bytecode
- Tools shown use bcc to compile a BPF program, pick some kprobe and attach said program.
- the BPF program will communicate informations back to userspace through perf events.

```
# bpftool prog
25: tracepoint  name sys_enter  tag 74384c5fbd30b4d5  gpl
        loaded_at Mar 04/18:08  uid 0
        xlated 264B  jited 190B  memlock 4096B  map_ids 22
# bpftool map
22: hash  flags 0x0
        key 16B  value 8B  max_entries 4096  memlock 364544B
23: perf_event_array  name printf  flags 0x0
        key 4B  value 4B  max_entries 16  memlock 4096B
```

/usr/share/bcc/tools/execsnoop

```
1  bpf_text = """
2  ...
3      events.perf_submit(ctx, data, sizeof(struct data_t));
4  ...
5  """
6  # initialize BPF
7  b = BPF(text=bpf_text)
8  execve_fnname = b.get_syscall_fnname("execve")
9  b.attach_kprobe(event=execve_fnname, fn_name="syscall__execve")
10 b.attach_kretprobe(event=execve_fnname, fn_name="do_ret_sys_execve")
11 ...
12 def print_event(cpu, data, size):
13    ...accumulate and print...
14 # loop with callback to print_event
15 b["events"].open_perf_buffer(print_event)
16 while 1:
17     try:
18         b.perf_buffer_poll()
19     except KeyboardInterrupt:
20         exit()
```

Thanks!