DE LA RECHERCHE À L'INDUSTRIE

www.cea.fr

Debugging kernel

Outils pour comprendre ce qu'il se passe la dedans

Dominique Martinet

CEA

January 9, 2020

À quoi s'attendre

- Formation recettes de cuisines, pricipalement sur crash
- Pas une solution magique, il faut toujours :
  - Chercher à comprendre ce qu'on analyse
  - Lire du code
  - Jouer à Sherlock Holmes
- Mais vous devriez ressortir d'ici avec au moins un outil de plus pour tout ça.

Prérequis

- Bases de C
  - Bases d'assembly peut aider, mais pas vu ici
- Commandes unix standard
- Bon à connaitre (revus rapidement lors du TP) :
  - Savoir ce qu'est une stack
  - Connaitre un peu les types de base du kernel (en praticulier listes)

- Une image de VM est disponible sur simple mail, n'hésitez pas à demander par email. Elle est nécessaire pour reproduire les TPs.
- Importez l'image disque dans un outil tout fait ou lancez une commande semblable à la suivante :

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

- En l'absence d'image cloud-init, vous devriez avoir un prompt de login où vous pourrez vous connecter avec root :toor.

Login

- Se connecter sur les postes si ce n'est pas déjà fait.
- Se connecter à inti avec les mêmes identifiants
- Pour pouvoir copier/coller des choses si besoin ou revenir sur une ancienne slide, celles-ci sont disponible sur `~stag01/presentation.fr.pdf`

VM pcocc

Démarrer une VM.

Le script génère une clé ssh, crée un script cloud-init avec cette clé et lance la VM pcocc préparée pour la formation. La VM est un simple linux ocean avec les packets debuginfo installés, les outils utilisés, et des crash dumps copiés d'ailleurs.

```
$ module load datadir/formation
$ $FORMATION_CCCSCRATCHDIR/start_debug_vm.sh
```

Usage

- `pcocc ssh root@vm0`
- `pcocc scp root@vm0:foo bar`
- `pcocc console vm0`, `pcocc reset vm0`, etc etc...

Liste des paquets

lustre lustre-debuginfo kernel-debuginfo crash systemtap perf bcc-tools gdb cscope ripgrep bash-completion htop bpftool (llvm-toolset-7.0 devtoolset-8 bpftrace)

Crash

- Outil pour analyser l'état du kernel
- Principalement un gros wrapper autour de gdb

Live

- Ne peut pas donner les stacks des threads actifs (comme /proc/pid/stack)
- Selon le driver peut modifier la mémoire de la machine (pas par défaut)
- Ne prend pas de locks donc attention aux choses qui bougent

Post-mortem (`kexec-kdump`)

- dans `/var/crash/<ip>-<date>/vmcore`

Lancer 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>
```

Notes crash

- Les infos de la machine données ici peuvent être réaffichées si besoin avec `sys`
- Le process listé est celui qui était actif sur le cpu qui a causé le crash
- Ce process est utilisé pour toutes les commandes qui prennent un pid/task si on n'en précise pas
- En cas d'urgence, `help` / `help <command>` sont de bons amis

Analyse - 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
```

Notes crash

- On peut utiliser des pipes comme dans un shell
- On peut aussi rediriger dans un fichier !

Analyse - 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
```

Notes crash

- Bien plus de détails avec switchs, en particulier :
  - `-l` pour line number
  - `-FF` pour frame (full ?)
- La position des registres compte : entrée de syscall (ici) ou interrupt
  - calling convention x86_64 pour syscalls : RAX = syscall#, puis RDI, RSI, RDX, R10, R8, R9
  - ici, le premier argument de ioctl est un fd, dans RDI.

Analyse - 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
```

Notes crash

- Logs trinity pas dispo ici mais permettrait de remonter dans les logs ou code utilisateur

Analyse - `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
```

Notes crash

- On retrouve la dentry et le file de tout à l'heure
- la syntaxe `[addr:slab]` n'indique que l'appartenance à un pool d'allocation : ce n'est pas nécessairement le début de la structure !
- les pools de même taille sont regroupés, e.g. même si le code a un `kmem_cache_create()` on n'aura pas forcément l'indice

Analyse - `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:
...
```

Notes crash

- `mod -s <module>` ou `mod -S`
- ... reste à lire le code et comprendre ce qu'il s'est passé !
- cscope est votre ami

Conclusion crash client

- Crash n'est pas un outil magique, il faut vraiment regarder dans le code ce qui peut nous faire arriver là.

Conclusion crash client

- Crash n'est pas un outil magique, il faut vraiment regarder dans le code ce qui peut nous faire arriver là.

```
@@ 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);
```

Conclusion crash client

- Crash n'est pas un outil magique, il faut vraiment regarder dans le code ce qui peut nous faire arriver là.

```
@@ 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);
```

- Quelque chose a remis `fd_grouplock` à zero sans changer le flag ?
- Ou bien `fd_flags` a été modifié sans toucher au grouplock ?

Conclusion crash client

- Crash n'est pas un outil magique, il faut vraiment regarder dans le code ce qui peut nous faire arriver là.

```
@@ 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);
```

- Quelque chose a remis `fd_grouplock` à zero sans changer le flag ?
- Ou bien `fd_flags` a été modifié sans toucher au grouplock ?
- . . . Il y a autre ioctl qui permet de mettre des valeurs arbitraires dans `fd->fd_flags` sans aucune vérification
- LU-13257 ouvert pour l'occasion (Bug trouvé pour cette formation !)

Autre problème

- On passe sur un serveur
- Pas de crash franc : causé par un sysrq-c fait parce que le serveur ne répondait plus bien.
- Qu'est-ce qu'on peut regarder ?

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>
```

Notes crash

- Pas de crash franc : SysRq crash
- Il faut faire un peu plus le tour de la machine

Analyse - 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]
...
```

Notes crash

>, 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

Analyse - `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"
```

Notes crash

- première colonne entre crochet : dernière date à laquelle le thread a été schedulé (uptime en nanoseconde)
- processes triés sur cette colonne : utile pour trouver des groupes de threads bloqués
- $1834107714279 - 1833456118992 = 652ms$
  - loin des traces à partir de 120s en UN ou RU sans tourner
- `dmesg | tail` -> `[ 1834.107266] ...`

Analyse - 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
```

Notes crash

- plein d'arguments possibles : regex (`'mdt0.*'`), state (`active`, `RU`, `UN`)...
- `foreach bt -R lock_res_and_lock`, `foreach bt -FF`, `foreach files`...

Analyse - `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
...
```

Analyse - `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
```

Notes crash

- 0*xb*0 = 176 : chercher kmalloc-192
- Une seule valeur qui tombe rond dans bt -FF la dedans...

Analyse - `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,
```

Notes crash

`lr_name` souvent un fid, mais ici sur un OST on a l'oid
`lr_refcount` Nombre de références à la resource
`lr_granted/waiting` listes de locks

Analyse - `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",
```

Notes crash

- Déréférences consécutives non triviales mais possibles

Parenthèse 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
```

Parenthèse linked list kernel


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

Notes

- double-linked list
- Souvent tête de liste séparée : `list -H`
  - i.e. `lustre/ldlm/ldlm_internal.h:38: extern struct list_head ldlm_srv_namespace_list;`
  - `/!\` même type pour la tête que le reste

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,
...
```

Analyse - `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
```

Notes crash

- ici (voir code) `lr_granted`/`lr_waiting` sont des têtes de listes à part
- Contiennent des `struct ldlm_lock` sur le champ `l_res_link`

Analyse - `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
```

Analyse - `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
...
```

Notes crash

- `-S` plus rapide mais pas de résolution des symboles
- plusieurs `LCK_PW` en granted avec grandes majorités de `PR`, à priori incompatibles, mais `l_req_extent` différents.

Analyse - `struct ldlm_lock` 3 - trouver un nid

Analyse - `struct ldlm_lock` 3 - trouver un nid


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

Analyse - `struct ldlm_lock` 3 - trouver un 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
```

Analyse - `struct ldlm_lock` 3 - trouver un 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
  },
```

Notes crash

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

Analyse - industrialisation


```
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
```

Analyse - industrialisation 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 - industrialisation 3

Notes crash

- On utilise rd et pas `struct foo.bar` pour moins passer par gdb (beaucoup plus lent, un patch est en cours pour avoir quelque chose comme le `-S` vs `-s` de `list`)
- Abuser des fichiers et commandes externes
- L'extension `crash-extscript` pourrait aider si besoin, mais au final on s'y habitue...

Analyse - `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
```

- 0x450 = 1104 : kmalloc-2048 ou `ptlrpc_cache`
- à part le nid dans `rq_peer`, le reste est assez difficilement exploitable

Analyse - `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` bloqués dans un dossier utilisateur pendant que je préparais mes slides. . .
- Petit tour rapide : vous devriez pouvoir refaire le même type de raisonnements

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 ffffffffc0ea9aad
 #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 ffffffffc14d240e
 #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 quelques heures !
- unlink en premier

retrouver le fichier

- bt -FF sur les threads bloqués
  - (manque de chance) le unlink s'est débloqué juste à ce moment, je n'ai pas eu le thread du unlink avec le fichier
- Mais on a le dossier + changelogs : fichier retrouvé (un seul fichier unlink ici)
- il y avait un HSM restore sur le fichier... Fini juste au moment où ça s'est débloqué.
  - les ls/getattr attendaient sur le unlink (lock sur le dossier), qui attendait sur le hsm restore (lock sur le fichier) avec la lock du dossier prise.
  - expérience utilisateur pas géniale vu qu'au final tout le dossier est bloqué...

Aller plus loin

- Il reste plein de commandes pas explorées ici
  - dis(assemble), kmem, mount, dev, net, rd, bpf, etc. . .
- Vraiment lire le détail des backtraces
  - En réfléchissant un peu on peut retrouver exactement quelle position contient quelle variable sans deviner avec les slabs, mais ça serait une autre formation. . .
- extensions crash :
  - extension lustre pour tirer un dk du dump ou autre
  - https://people.redhat.com/anderson/extensions.html

perf probe

- Introduire des tracepoints un peu n'importe où dans le kernel (ou l'userspace !)

perf probe

- Introduire des tracepoints un peu n'importe où dans le kernel (ou l'userspace !)
- Pour l'exemple, on veut dumper toutes les locks d'un fs

perf probe

- Introduire des tracepoints un peu n'importe où dans le kernel (ou l'userspace !)
- Pour l'exemple, on veut dumper toutes les locks d'un fs
- Un ami™ nous parle de `/sys/kernel/debug/lustre/ldlm/dump_namespaces`
  - echo 1 dans le fichier va dump les resources et leurs locks associées dans `lctl dk` si `dlmtrace` est actif
  - sauf que le buffer dk est trop petit même si on augmente sa taille (bug ?)
- Perf va nous sauver !

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
```

Notes perf

- Besoin du module chargé
- Permet d'explorer le code un peu
- On ne peut s'accrocher qu'aux lignes numérotées

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
```

Notes perf

- On a le choix d'où se mettre
- Toutes les lignes n'ont pas toutes les variables ! (même si ça devrait niveau C)
- Certaines lignes ont plusieurs emplacements une fois optimisées
  - Moins bien pour perf (une probe par locution)

ldlm dump_namespaces - `perf probe`


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

Notes perf

- On peut dumper des variables au passage de la probe
  - Plusieurs
  - Y compris déréférencées
- Pas de type complexe (ne peut pas demande toute une 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`


```
# 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
```

Notes perf

- Enregistre une probe (`-e`) ou plus (wildcards, comma separated list. . .)
- Enregistre une commande ou tout (`-a`)
- Enregistre les backtraces au niveau de la probe (`-g`)
- Plein d'autres filtres possibles, `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
```

Notes perf

- Tous les namespaces ldlm sont listés ici pêle-mêles (OST*, MDT*, MGS)
- `0x42` : oid d'un OST, `0x2000013a1` : debut de fid (MDT ou 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

- Pas le temps de présenter en détail
- Allez voir le site de Brendan Gregg
  - 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

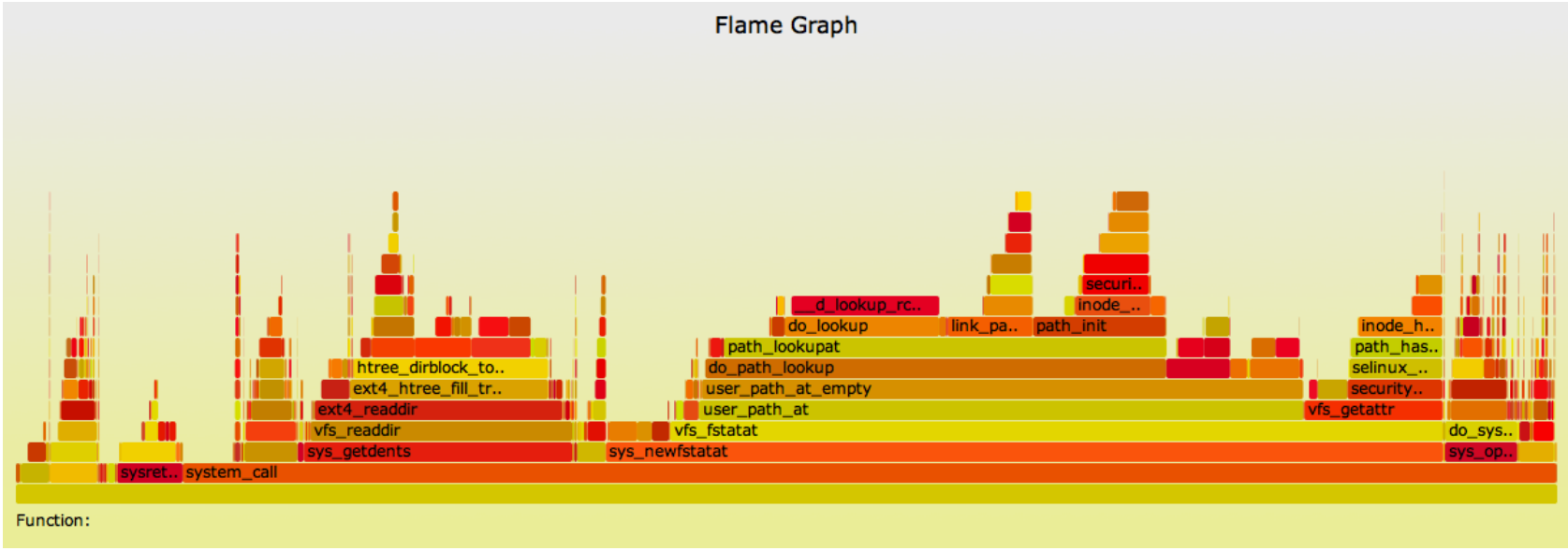

FIGURE 1 – Exemple de flame graph

- Plus puissant / intrusif que perf
- Compile un petit module kernel et l'insert : on peut tout casser
  - Mais du coup on peut l'utiliser pour contourner des problèmes !

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
```

- Ou bien copier le .ko dans `/lib/modules/$(uname -r)/systemtap` et `staprun <modulename>` dans un 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

- Plus puissant que perf pour dumper des types complexes : `$var$$`
- Essayons de faire comme la probe perf. . .

Example 2 - debug

- Plus puissant que perf pour dumper des types complexes : `$var$$`
- Essayons de faire comme la probe perf. . .

```
# 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

- Plus puissant que perf pour dumper des types complexes : `$var$$`
- Essayons de faire comme la probe perf. . .

```
# 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 tronquée à taille fixe (512 chars)
- Tableaux tronqués (`lr_name...`)

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

warning el7

- Petit bug avec `asm_volatile_goto` utilisé dans nos kernels mais non supporté par clang [3], pas backporté en el7

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

- Peut s'ajouter dans `arch/x86/include/asm/jump_label.h` en workaround
- La VM fournie a une nouvelle version de bcc qui contient le fix, cette version était nécessaire pour bpftrace, mais les exemples bcc-tools fonctionnent avec les outils de base el7 et ce contournement.

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

bpftrace

- Besoin de llvm/gcc/bcc plus récent. . .
- Abusons de llvm-toolset-7 et devtoolset-8 : https://medium.com/@tle211212/use-bpftrace-on-centos-7-6ab920624084
- Marche de base sous el8

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

- Regardons encore une fois la liste des locks ldlm...
- Malheureusement bpftrace ne sait pas deviner les types et se base sur ceux définis dans les headers eBPF du kernel
  - Avantage : pas besoin des debuginfos pour faire tourner les probes contrairement aux autres outils
- en 4.18+ avec `CONFIG_DEBUG_INFO_BTF=y` (BPF Type Format) ça devient automatique (pas encore dispo en el8.1, tester sous fedora ou autre)
- En attendant, on peut le faire à la main...

```
# 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
```

- Plus d'exemples : 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 est un simple compilateur BPF :
  - ça prend du code C et fait du bytecode BPF
- Les outils utilisent bcc pour compiler une programme BPF, choisissent des kprobe/kretprobe qui les intéressent, et y accrochent le programme
- Le programme BPF va remonter des informations à l'outil à travers des évènements `perf`

```
# 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()
```

Merci !