SystemTap For Runtime Analysis of Kernel Modules such as AFS

Yadavendra Yadav
Prashant Sodhiya

IBM

Agenda

- ✓ Introduction
- ✓ Architecture
- ✓ Advantages of SystemTap
- ✓ Using SystemTap
- ✓ SystemTap Language
- ✓ Introduction & Examples of Tapsets
- ✓ Tapsets for OpenAFS
- ✓ SystemTap Usage in AFS
- ✓ Case Studies
- ✓ Performance Measurement

Introduction

- SystemTap provides free software (GPL) infrastructure to simplify the gathering of information about the running Linux system.
- It is based on kprobes / kretprobe.
- Eliminates the tedious and disruptive process of instrumentation, recompile, install, and reboot sequence that may be otherwise required to collect data.
- Provides a simple command line interface and scripting language for writing instrumentation for a live running kernel.

Introduction (cont..)

SystemTap Target Audience:

- ✓ **Kernel Developer**: I wish I could add debug statements easily without going through the insert / build / reboot cycle.
- ✓ **Technical Support**: How can I get additional data out of a customer's kernel easily and safely ?
- ✓ **System Admin**: Occasionally jobs take significantly longer than usual to complete, or do not complete. Why ?
- ✓ **Student:** How can I learn more about the call flow of a kernel subsystem ?

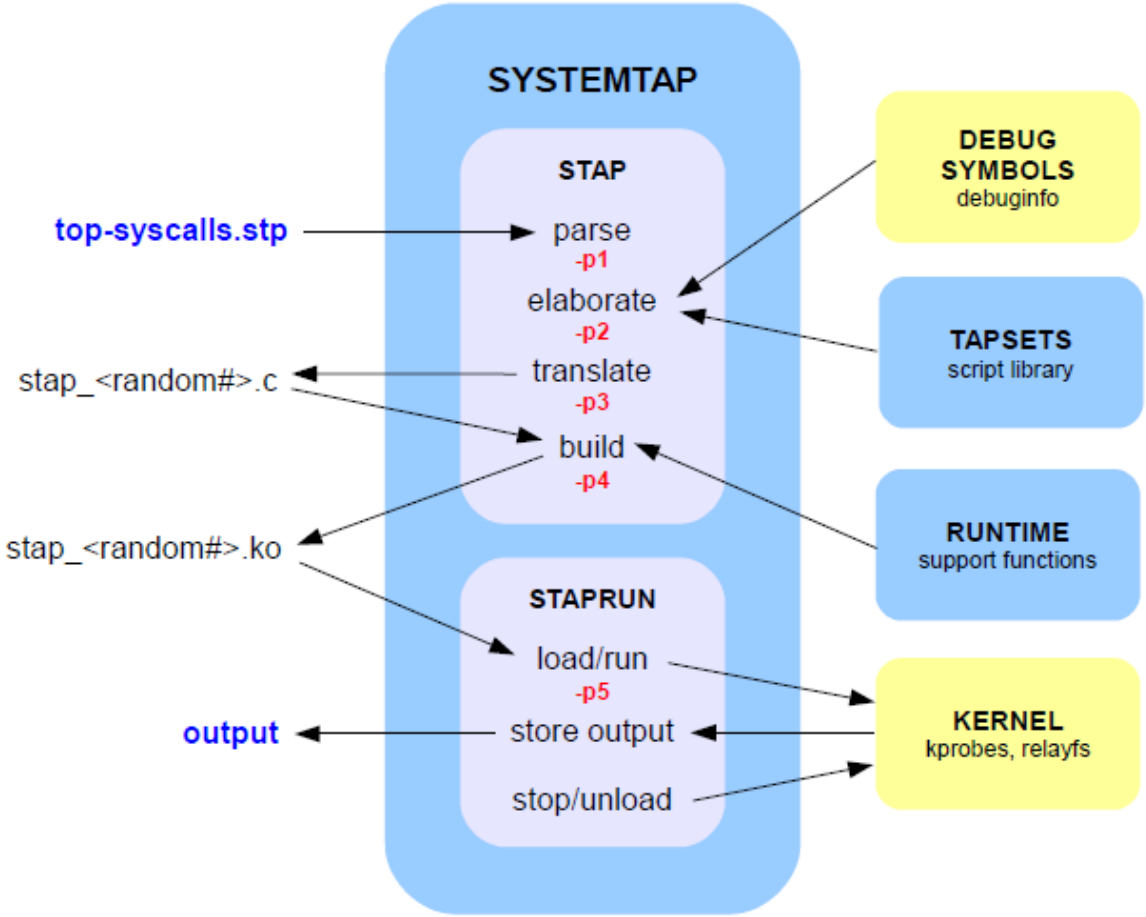
Architecture

- System tap uses Kprobes / Kretprobe for dynamic probing.

- Kprobes requires that you:
  - ✓ Write a kernel module.
  - ✓ Specify an address and handler for each probe point.
  - ✓ Be careful ! Mistakes can crash the system.
  - ✓ Powerful, but cumbersome to use.

Architecture (cont…)

SystemTap Processing Steps:



Advantages of SystemTap

- ✓ No module writing required. Create and insert probes quickly and easily using a simple scripting language.
- ✓ No kprobes knowledge required.
- ✓ No kernel addresses required. Automates gathering of symbol information.
- ✓ Provides pre-written probes for common kernel areas.
- ✓ Growing set of pre-written scripts.
- ✓ Powerful and simple to use.

Using SystemTap

Installation & Setup

To deploy SystemTap, install the following RPMs

- Systemtap
- Systemtap-runtime

Installing requires Kernel Information RPM

- Kernel-debuginfo
- kernel-debuginfo-common-arch
- kernel-devel
- For probing OpenAFS we need to install openafs-debuginfo package.

Using SystemTap (cont…)

Verifying Installation

stap -v -e 'probe vfs.read {printf("read performed\n"); exit()}'.

```
Pass 1: parsed user script and 45 library script(s) in 340usr/0sys/358real ms.
Pass 2: analyzed script: 1 probe(s), 1 function(s), 0 embed(s), 0 global(s) in 290usr/260sys
/568real ms.
Pass 3: translated to C into "/tmp/stapiArgLX/stap_e5886fa50499994e6a87aacdc43cd392_399.c"
in 490usr/430sys/938real ms.
Pass 4: compiled C into "stap_e5886fa50499994e6a87aacdc43cd392_399.ko" in 3310usr/430sys
/3714real ms.
Pass 5: starting run.
read performed
Pass 5: run completed in 10usr/40sys/73real ms.
```

Using SystemTap (cont…)

System Tap Command

stap [options] script.stp

Option	Description
-v	Increase verbosity
-g	Guru mode, embedded C allowed
-k	Keep temporary directory
-m	Set probe module name
-x	Sets target() to PID
-c	Start probes, run command , exit when it finishes
-r	Cross-compile to kernel RELEASE

* See stap(5) man page for complete list and details

Using SystemTap (cont…)

Cross Instrumentation

- Production environment will not have development & debuginfo packages. So how to run systemtap there ?
- All kernel development & debuginfo packages can be installed on a single host machine.
- On host machine below command will provide kernel module (e.g. file_op.ko)

  stap -r `uname -r` file_op.stp -m file_op -p4
- Target system only one RPM needs to be installed i.e. systemtap-runtime.
- On target systems run

  staprun <kernel module>

Using SystemTap (cont…)

Required Privileges

- ✓ Running stap and staprun requires elevated privileges to the system
- ✓ To allow ordinary users to run SystemTap without root access, add them to both of these user groups

1. **stapdev**

   Members of this group can use stap to run SystemTap scripts, or staprun to run SystemTap instrumentation modules.

2. **stapusr**

   Members of this group can only use staprun to run SystemTap instrumentation modules

SystemTap Language

- Probes & Probe Aliases – function entry & exit, source line
- kernel address, timer, begin/end
- Wildcarding
- Functions
- Types – string, 64-bit long, associative array, aggregation
- Comparison – if else & ternary operators
- Looping - while, for, foreach
- Usual binary & numeric operators
- String manipulation – sprint, sprintf, . & .= operators
- Output – log, print, printf
- Target variables – accessible with '$' prefix
- Embedded C – raw C code, not covered by safety checks

Introduction of Tapsets

- Probe set that encapsulates kernel subsystem knowledge. Defines probes, data, auxiliary functions.
- Abstracts away subsystem implementation details.
- Probes are usable and extendable by other scripts.
- Tested and packaged with SystemTap.
- Located in either:
  - /usr/local/share/systemtap/tapset if installed from source
  - /usr/share/systemtap/tapset if installed from rpm

Example of Tapsets

VFS tapset

```
probe generic.fop.open = kernel.function("generic_file_open")
{
      dev = __file_dev($filp)
      devname = __find_bdevname(dev, __file_bdev($filp))
      ino = $inode->i_ino
      file = $filp

      filename = __file_filename($filp)
         flag = $filp->f_flags
         size = $inode->i_size

      name = "generic_file_open"
      argstr = sprintf("%d, %d, %s", $inode->i_ino, $filp->f_flags, filename)
}
probe generic.fop.open.return = kernel.function("generic_file_open").return
{
      name = "generic_file_open"
      retstr = sprintf("%d", $return)
}
```

Tapsets for OpenAFS

Sample Tapset Routines:

- AfsLockInfo
- GetVFid
- PrintVcache
- PrintDcache
- …..

```
function PrintVcache:long (vcache:long)
{
      __fvcache = &@cast(vcache,"vcache","kernel:openafs")->f
      __afslock = &@cast(vcache,"vcache","kernel:openafs")->lock

      fcache_info = Getfvcache(__fvcache)
      printf("\t\tVcache Information [%p] : \n",vcache);
      printf("\t\t\t Fvcache Information [%p] : [%s]\n",__fvcache,fcache_info)
      printf("\t LockInformation : \n %s\n",AfsLockInfo(__afslock))
}
```

Tapsets for OpenAFS (cont…)

Tapset Routines

```
function PrintDcache:string (dcache:long)
{
      __lock = &@cast(dcache,"dcache","kernel:openafs")->lock
      __tlock = &@cast(dcache,"dcache","kernel:openafs")->tlock
      __mflock = &@cast(dcache,"dcache","kernel:openafs")->mflock
      validPos = @cast(dcache,"dcache","kernel:openafs")->validPos
      index = @cast(dcache,"dcache","kernel:openafs")->index
      refCount = @cast(dcache,"dcache","kernel:openafs")->refCount
      dflags = @cast(dcache,"dcache","kernel:openafs")->dflags
      mflags = @cast(dcache,"dcache","kernel:openafs")->mflags
      __fcache = &@cast(dcache,"dcache","kernel:openafs")->f
      fcache_str = Getfcache(__fcache)
      lock_str = AfsLockInfo(__lock)
      tlock_str = AfsLockInfo(__tlock)
      mflock_str = AfsLockInfo(__mflock)

      printf("DCACHE [%p] :: Index [%d] validPos [%d] refCount [%d] dflags [%c] mflags [%c] fcache [%s]
Lock [%s] Tlock [%s] mflock [%s]
\n",dcache,index,validPos,refCount,dflags,mflags,fcache_str,lock_str,tlock_str,mflock_str)
}
```

Tapsets for OpenAFS (cont…)

Tapset Routines

```
probe openafs.aops.writepage = module("openafs").function("afs_linux_writepage")
{

      __page = $pp
      dev = __page_dev(__page)
      ino = __page_ino(__page)
      for_reclaim = "N/A"
      for_kupdate = "N/A"

      if (@defined($wbp)) {
            for_reclaim = $wbc->for_reclaim
            for_kupdate = $wbc->for_kupdate
      }

      __inode = __address_inode(__page)
      __vcache = VTOAFS(__inode)

      name = "openafs.aop.writepage"
      page_index = $pp->index

}


probe openafs.aop.writepage.return = module("openafs").function("afs_linux_writepage").return
{
      name = "openafs.aop.writepage.return"
      retstr = sprintf("Return Value : %d",$return)
}
```

SystemTap Usage in AFS

- Defect Analysis and simulation
- Defect Testing
- Fault Injection
- Performance
- Tapset

Case Study-1

Problem Statement: User application returned EIO error during msync operation.

Initial Analysis : Need to find which AFS function is failing & how EIO is returned back to a application.

STEP 1: Find which AFS function is failing

- Script

```
global Agg
global RetAgg
probe begin {
        printf("Started probe\n")
}
probe module("openafs").function("*").call {
        if (tid() == target()) {
                Agg[probefunc()]++;
        }
}
probe module("openafs").function("*").return {
        if ((tid() == target()) && @defined($return))
        {
                        RetAgg[probefunc()] <<< $return
        }
}
probe end {
        foreach (Iter in Agg) {
                printf("Function Called %s : Count %d\n",Iter,Agg[Iter])
        }
        foreach (Iter1 in RetAgg) {
                printf("Func %s::",Iter1)
                print(@hist_linear(RetAgg[Iter1], 0, 10, 1))
        }
}
```

Case Study-1 (cont…)

Statistics of Functions Return Value

- Output

```
Func afs_EvalFakeStat::value |-------------------------------------------------- count
    0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@                       28
    1 |                                                    0
    2 |                                                    0

Func afs_linux_permission::value |-------------------------------------------------- count
    0 |@@@@@@@@@@@@@@@@@@                                 18
    1 |                                                    0
    2 |                                                    0

Func afs_linux_writepage::value |-------------------------------------------------- count
    0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@  34086
    1 |                                                       0
    2 |                                                       0
      ~
    9 |                                                       0
   10 |                                                       0
  >10 |@@@@@@@@@@@@@                                       9141
```

Case Study-1 (cont…)

STEP 2: Stack Trace of the failing AFS function

- Script

```
probe openafs.aop.writepage.return {
        if ((tid() == target()) && @defined($return) && $return)
        {
           printf("%s <- %s\n",name,retstr)
           print_backtrace()
        }
}
```

- Output

```
Returning from:  0xffffffffa04b23b0 : afs_linux_writepage+0x0/0x4c0
[openafs]
Returning to   :  0xffffffff8112caa7 : __writepage+0x17/0x40 [kernel]
 0xffffffff8112ddb9 : write_cache_pages+0x1c9/0x4b0 [kernel]
 0xffffffff8112e0c4 : generic_writepages+0x24/0x30 [kernel]
 0xffffffff8112e105 : do_writepages+0x35/0x40 [kernel]
 0xffffffff8111a4bb : __filemap_fdatawrite_range+0x5b/0x60 [kernel]
 0xffffffff8111a51a : filemap_write_and_wait_range+0x5a/0x90 [kernel]
 0xffffffff811b1a0e : vfs_fsync_range+0x7e/0xe0 [kernel]
 0xffffffff811b1add : vfs_fsync+0x1d/0x20 [kernel]
 0xffffffff8114c951 : sys_msync+0x151/0x1e0 [kernel]
 0xffffffff8100b072 : system_call_fastpath+0x16/0x1b [kernel]
```

Case Study-1 (cont…)

STEP 3: Which function is returning EIO error

- Script

```
probe kernel.function("do_writepages").return {
      if ((tid() == target()) &&  @defined($return))
            printf("%s Returned %d\n",probefunc(),$return)
}

probe kernel.function("__filemap_fdatawrite_range").return {
      if ((tid() == target()) &&  @defined($return))
            printf("%s Returned %d\n",probefunc(),$return)
}

probe kernel.function("filemap_write_and_wait_range").return {
      if ((tid() == target()) &&  @defined($return))
            printf("%s Returned %d\n",probefunc(),$return)
}
```

- Output

```
wait_on_page_writeback_range Returned -5
filemap_write_and_wait_range Returned -5
write_cache_pages Returned 0
generic_writepages Returned 0
do_writepages Returned 0
__filemap_fdatawrite_range Returned 0
__filemap_fdatawrite_range Returned 0
wait_on_page_writeback_range Returned 0
vfs_fsync_range Returned -5
vfs_fsync Returned -5
sys_msync Returned -5
```

Case Study-2

Problem Statement : Dcache readlock leak in case “afs_dir_GetVerifiedBlob” fails inside “afs_linux_readdir”.

- Mainly afs_PutDcache was called without releasing a readlock

Simulation: To simulate this defect “afs_dir_GetVerifiedBlob” should fail. For this we used SystemTap as a fault injection mechanism.

- Script

```
probe module("openafs").function("afs_dir_GetVerifiedBlob").return {
        printf("Ret value [%d] going to change to non-zero\n",$return);
        $return = -1;
        printf("Ret value changed to[%d] \n",$return);
}
```

Case Study-2 (cont…)

- ✓ With above fault-injection we were able to test the fix.
- ✓ To verify that there is no such lock leak in other places, we added probe during return of "afs_PutDCache" which checks for lock leak.

```
probe module("openafs").function("afs_PutDCache").return
{
__dcache = $adc
__lock = &@cast(__dcache,"dcache","kernel:openafs")->lock
__mflock = &@cast(__dcache,"dcache","kernel:openafs")->mflock
__tlock = &@cast(__dcache,"dcache","kernel:openafs")->tlock
if ((CheckForDcacheLockLeak(__lock,__dcache) ||
     CheckForDcacheLockLeak(__mflock,__dcache) ||
     CheckForDcacheLockLeak(__tlock,__dcache)))
     {
               printf("ALERT::Lock Leak Detected\n")
               print_backtrace()
     }
}
```

Performance Measurement

SystemTap can used to gather performance statistics

- Script

```
global entry, timings
function collect_entry()
{ entry[probefunc(),tid()] = gettimeofday_us()
}

function collect_exit()
{ timings[probefunc()] <<< (gettimeofday_us() - entry[probefunc(),tid()])
}

probe module("openafs").function("afs_GetDCache").call
{
      collect_entry()
}

probe module("openafs").function("afs_GetDCache").return
{
      collect_exit()
}

probe end
{
      printf("function count min_us avg_us max_us\n")
      foreach (i in timings)
      {
            printf("%-25s %7d %8d %8d %8d\n", i,
                  @count(timings[i]),@min(timings[i]),
                  @avg(timings[i]), @max(timings[i]))
      }
}
```

- Output

```
function                  count   min_us   avg_us   max_us
afs_GetDCache               271        0      200    52396
```

References

- http://sourceware.org/systemtap/examples
- https://access.redhat.com/documentation/enUS/Red_Hat_Enterprise_Linux/6/html/SystemTap_Beginners_Guide/
- http://sourceware.org/systemtap/wiki/

Thank You