Architecture of systemtap: a Linux trace/probe tool

张朝威

2017.2.13

大纲

- 设计目标
- 设计构架
- 编程脚本
- 库 tapset
- 安全考虑
- 引用

术语

- Probe Points are abstract names given to identify a particular place in kernel/user code, or a particular event (timers, counters) that may occur at any time.
- Handlers are subroutines written in the script language, which are run whenever the probe points are hit.
- A probe definition identifies one or more probe points, and a body of code to execute when any of them is hit.

设计目标

- 易用
- 可扩展性
- 性能
- 透明性
- 简单
- 弹性
- 安全

设计架构

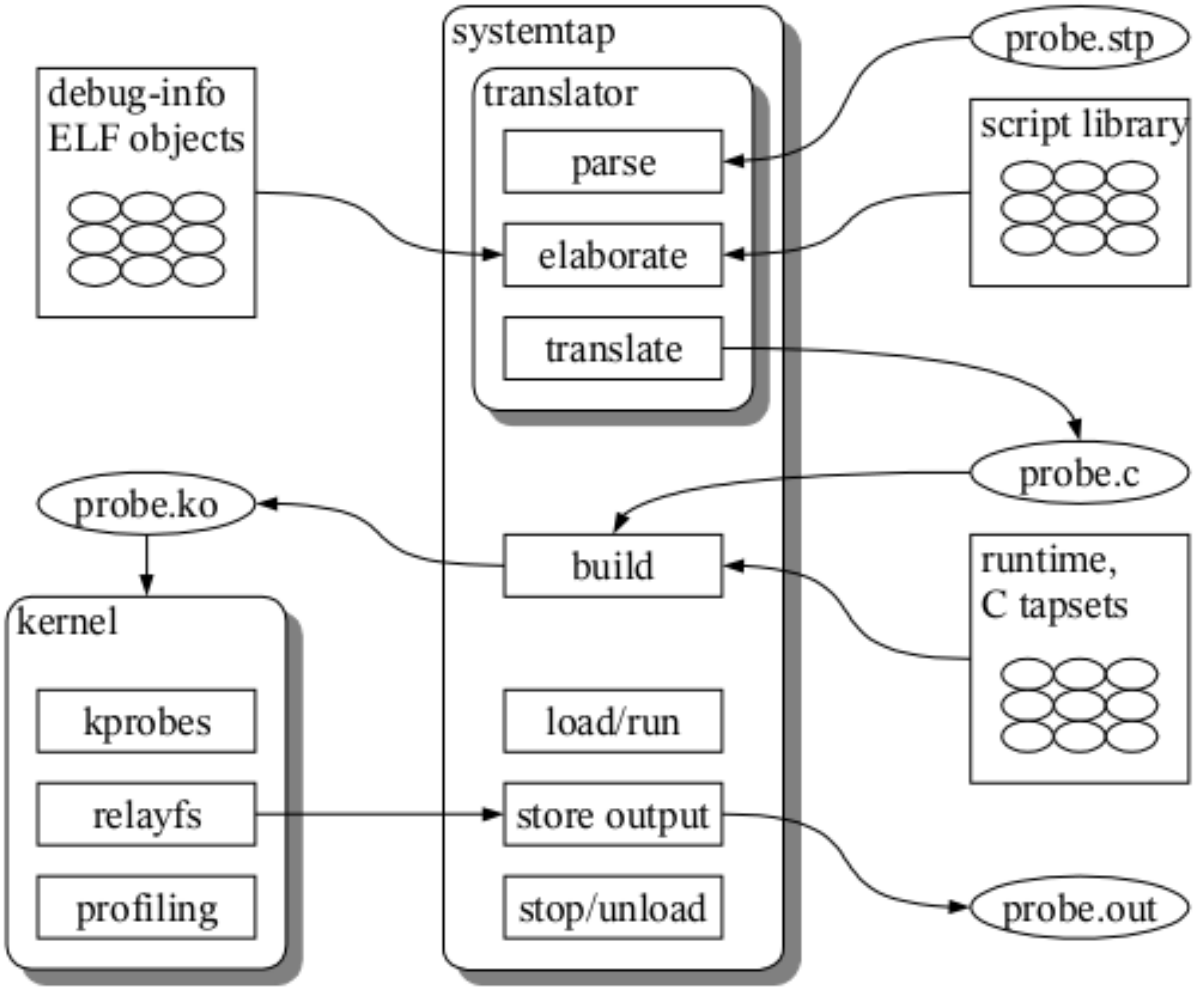

Figure 1: Systemtap processing steps

构成要素：

1. 脚本语言；
2. 脚本库；
3. 探测对象；
4. 运行时 C 库；
5. kernel 对象；

过程要素：

1. parse 解析；
2. elaborate 加工；
3. translate 转换；
4. build 构建；
5. load 加载；
6. run 运行；

设计架构——构成要素

- 脚本语言

  1. 主要由探测点和处理代码构成；

  2. 语言构成类似于 C 和 AWK ；

- 加工

  1. 类似于 C 中的编译和链接从而生成可执行的单独文件；

  2. 同时置放探测点和处理代码；

-

设计架构——构成要素

- 转换

  1. 转换为了相应的 C 代码，同时有锁保护、安全检查、控制流等；

  2. 探测点是插入在用户程序中的，而处理代码则是被探测点注册的函数进行了封装；

  3. 另外还包括了对运行时的引用；

- 执行

  1. 使用 insmod 来加载内核模块 , 插入探测，然后坐等处理代码被触发，在最后卸载的时候再移除探测；

编程脚本

- 三要素：探测定义；辅助函数定义；全局变量声明；（不分顺序，允许前向声明）
- 探测定义主要包括探测点和处理代码；
- 辅助函数定义，包括内建函数，限制递归调用次数；
- 全局变量声明
- 对于脚本执行过程中遇到的致命错误，处理方式为early abort

编程脚本——探测点

- 在探测定义中，使用逗号来分隔多个探测点；探测点则使用点号来分隔；
- 示例： kernel.function("foo").return

  前半部分 kernel or module(“foo”) 用于标识目标程序，用来搜索符号信息来解析剩余部分的模式；

编程脚本——探测点（函数）

- 可以使用 function(“fn”) 来标识一个函数
- 可以增加后缀 @filename 或 @filename:lineno 来标识；
- 可以使用 * ? 进行多匹配；
- 可以使用 function(address) 函数地址进行标识；
- 紧接着是替换部分 callee
- 最后是 return, 指明在何处探测

编程脚本——探测点（语句）

- 对函数内的语句进行探测
- 可使用 statement(“foo”) 再加上“文件 + 行号”进行界定，可以在行前或行后进行插入探测；
- 可以使用 statement(“foo”) 再加上“相对函数的行数”进行界定；
- 可以使用 statement(“foo”) 再加上 label("need_resched") 来标识函数内的某个 label ；
- 可以使用 statement （ address ）通过绝对行号来指定；

编程脚本——探测点（事件）

- 探测点可以定义在抽象事件上
- begin 特殊事件： during systemtap initialization, before normal probes are enabled
- End 特殊事件： during late shutdown, after all normal probes have been disabled

编程脚本——探测点示例

`kernel.function("sys_read").return`
a return probe on the named function.

`module("ext3").function("*@fs/ext3/inode.c")`
every function in the named source file, a part of ext3fs

`kernel.function("kmalloc").callees`
every function known statically to be callable from kmalloc

`module("usb-storage").statement(0x0233)`
the given address, which must be at an instruction boundary

`kernel.function(0xffffffff802202dc).return`
a return probe on whichever function that contains the given address

编程脚本——语言元素

- 标识（ C 标识； $ 符号； ）
- 类型（自动推导），主要有数字、字符串、关联数组、统计；
- 分号用于分隔不同的语句；
- 注释，可便用 C/c++/shell 三类注释；
- 空白，与 C 一致；

编程脚本——语句和表达式

- 与 C 语句一致（不包括 goto switch 这二者）
- 同时支持 AWK 的部分语句；
- 支持 C 的部分表达式，同时支持 AWK 的部分表达式；自身有增强的部分；
- 不支持 C 表达式如右

  - `<struct> . <field>`
  - `* <pointer>`
  - `<pointer> -> <field>` (But see `<macro> -> <field>`.)
  - `& <lvalue>`
  - `sizeof`
  - type casts – e.g., `(long) val`
  - , (comma operator)

编程脚本——辅助函数

- 辅助函数与 AWK 非常类似
- 关键字 function, 函数参数列表，以及大括号括起来的语句块；
- 自动来推导函数类型以及参数类型；

TAPSET

- 主要有 2 类：脚本类； C 类（未介绍）；
- 主要介绍了脚本类 TAPSET（见原文示例）
-

其他

- 主要介绍了安全相关
- 主要介绍了１）内核态数据向用户态数据传输的优化；２）输出的性能优化；
-

引用

- Bryan M. Cantrill, Michael W. Shapiro, and Adam H. Levinthal. Dynamic Instrumentation of Production Systems. In Proceedings of the 2004 USENIX Technical Conference, pages 15–28, June 2004.
- Richard J. Moore. A universal dynamic trace for Linux and other operating systems. In FREENIX, 2001