SystemTap
Linux下的万能观测工具

褚霸
核心系统数据库组
chuba@taobao.com
http://yufeng.info
2010/11/18

Agenda

介绍SystemTap

安装和系统要求

实践例子

参考和杂项

结论

SystemTap是什么？

According to http://sourceware.org/systemtap/

> SystemTap provides free software (GPL) infrastructure to simplify the gathering of information about the running Linux system. This assists diagnosis of a performance or functional problem. SystemTap eliminates the need for the developer to go through the tedious and disruptive instrument, recompile, install, and reboot sequence that may be otherwise required to collect data.

观察活体系统最佳工具，前提是你懂得如何观察！

SystemTap是如何工作的

1. write or choose a script describing what you want to observe
2. stap translates it into a kernel module
3. stap loads the module and communicates with it
4. just wait for your data

五步走

```
# stap -uv test.stp
Pass 1: parsed user script and 74 library script(s) using
86868virt/20488res/1792shr kb, in 190usr/20sys/209real ms.
Pass 2: analyzed script: 1 probe(s), 0 function(s), 0 embed(s), 0 global(s) using
87264virt/21148res/1976shr kb, in 10usr/0sys/7real ms.
Pass 3: translated to C into
"/tmp/stapz2iv97/stap_aef621603e006af62084b361e0a0c981_553.c" using
87264virt/21332res/2144shr kb, in 0usr/0sys/0real ms.
Pass 4: compiled C into "stap_aef621603e006af62084b361e0a0c981_553.ko" in
1230usr/160sys/1384real ms.
Pass 5: starting run.
Pass 5: run completed in 10usr/20sys/12331real ms.
```

SystemTap 探测点例子

SystemTap is all about executing certain actions when hitting certain probe points.

- syscall.read
  when entering read() system call
- syscall.close.return
  when returning from the close() system call
- module("floppy").function("*")
  when entering any function from the "floppy" module
- kernel.function("*@net/socket.c").return
  when returning from any function in le net/socket.c
- kernel.statement("*@kernel/sched.c:2917")
  when hitting line 2917 of le kernel/sched.c

更多探测点例子

- timer.ms(200)
  every 200 milliseconds

- process("/bin/ls").function("*")
  when entering any function in /bin/ls (not its libraries orsyscalls)

- process("/lib/libc.so.6").function("*malloc*")
  when entering any glibc function which has "malloc" in its name

- kernel.function("*exit*").return
  when returning from any kernel function which has "exit" in its name

RTFM for more (man stapprobes).

SystemTap编程语言

- mostly C-style syntax with a feeling of awk
- builtin associative arrays
- builtin aggregates of statistical data
  very easy to collect data and do statistics on it (average, min, max, count,. . .)
- many helper functions (builtin and in tapsets)

RTFM: SystemTap Language Reference shipped with SystemTap (langref.pdf)

Performances and safety

- language-level safety features
  - no pointers
  - no unbounded loops
  - type inference
  - you can also write probe handlers in C (with -g) but don't complain if you break stuff
- runtime safety features
  - stap enforces maximum run time for each probe handler
  - various concurrency constraints are enforced
  - overload processing (don't allow stap to take up all the CPU time)
  - many things can be overriden manually if you really want
  - see SAFETY AND SECURITY section of stap(1)

The overhead depends a lot of what you are trying to do but in general stap will try to stop you from doing something stupid (but then you can still force it to do it).

Some helper functions you'll see a lot

pid() which process is this?
uid() which user is running this?
execname() what is the name of this process?
tid() which thread is this?
gettimeofday_s() epoch time in seconds
probefunc() what function are we in?
print_backtrace() figure out how we ended up here

There are many many more. RTFM (man stapfuncs) and explore /usr/share/systemtap/tapset/.

Some cool stap options

-x trace only speci ed PID (only for userland probing)

-c run given command and only trace it and its children
(will still trace all threads for kernel probes)

-L list probe points matching given pattern along with
available variables

-d load given module debuginfo to help with symbol resolution in
backtraces

-g embed C code in stap script unsafe, dangerous and fun

Agenda

介绍SystemTap

安装和系统要求

实践例子

参考

结论

Requirements

- SystemTap探测用户空间程序需要utrace的支持，但是这个特性还没有被Linux上游吸收。Redhat的发行版本目前支持这个特性。

- 源码级别跟踪需要安装符号信息
  包层面需要安装package-debuginfo on RPM distros
  用户自己的程序需要gcc -g -gdwarf-2 -g3编译

- stap脚本是编译成内核模块运行的，需要root权限

安装SystemTap

RHEL5U4需要安装内核符号信息:

```
rpm -i kernel-debuginfo-common-2.6.18-164.el5.x86_64.rpm
rpm -i kernel-debuginfo-2.6.18-164.el5.x86_64.rpm
```

由于5U4带的SystemTap是0.97版本，需要升级到1.3:

```
./configure prefix=/usr && make && make install
```

如何验证是否成功:

```
# stap topsys.stp
                SYSCALL      COUNT
                 read         48
                 fcntl        42
                  ...
                 fstat         1
-------------------------------------------------------------
```

Agenda

介绍SystemTap

安装和系统要求

实践例子

参考和杂项

结论

Example: 谁在执行我们的程序

Listing: exec.stp

```
probe syscall.exec*{
printf("exec %s %s\n", execname(), argstr)
}
```

```
$ stap -L 'syscall.exec*'
syscall.execve name:string filename:string args:string argstr:string
$filename:char* $argv:char** $envp:char** $regs:struct pt_regs*
```

```
# stap exec.stp
exec sshd /usr/sbin/sshd "-R"
exec sshd /bin/bash
```

例子: 谁杀了我的程序

Listing: sigkill.stp

```
probe signal.send{
if(sig_name == "SIGKILL")
printf("%s was sent to %s (pid:%d) by %s uid :%d\n", sig_name,
pid_name , sig_pid, execname(), uid())
}
# kill -9 `pgrep top`

# stap sigkill.stp
SIGKILL was sent to top (pid:19281) by bash uid :50920
```

Example tac.c: 工具函数

```
#include <stdio.h>
#include <stddef.h>
#include <string.h>
char* haha = "wahaha\n";
char* read_line(FILE* fp, char* buf, size_t len){  return fgets(buf, len,
fp);}
char* reverse_line(char* line, size_t l){
  char *s = line, *e = s + l - sizeof("\n"), t;
  while(s < e) {  t =*s, *s = *e, *e = t; s++, e--; }
  return line;
}
void write_line(char* line){ fputs(line, stdout);}
```

Example tac.c continued : 主程序

```
int main(int argc, char * argv[]){
  char buf[4096], *line;
  FILE* fp = stdin;
  if(argc != 1 ) {fp = fopen(argv[1], "r");}
  if(fp == NULL){fprintf(stdout, "usage: %s filename\n", argv[0]);return
-1;}

  while((line = read_line(fp, buf, sizeof(buf)))){
    line = reverse_line(line, strlen(line));
    write_line(line);
  }

  if(argc != 1) fclose(fp);
  return 0;
}
```

编译tac

必须要带调试信息
gcc -g -gdwarf-2 -g3 tac.c

确认符号信息的存在
stap -L 'process("a.out").function("*")'
process("/tmp/a.out").function("main@/tmp/tac.c:25") $argc:int $argv:char**
$buf:char[] $line:char* $fp:FILE*
process("/tmp/a.out").function("read_line@/tmp/tac.c:7") $fp:FILE* $buf:char*
$len:size_t
process("/tmp/a.out").function("reverse_line@/tmp/tac.c:11") $line:char* $l:size_t
$s:char* $e:char* $t:char
process("/tmp/a.out").function("write_line@/tmp/tac.c:21") $line:char*

Example 1: 读出程序的参数

```
function get_argv_1:long(argv:long) %{ /* pure */
  THIS->__retvalue =(long) ((char**)THIS->argv)[1];
%}

probe process("a.out").function("main"){
  filename = "stdin";
  if($argc > 1) {
    filename = user_string(get_argv_1($argv));
  }
  println(filename);
}
```

Example 1 continued:

```
# echo "hi"|./a.out
# ./a.out tac.c


# stap -gu ./ex1.stp
:)
stdin
tac.c
```

Example 2: callgraph for anything

```
function trace(entry_p, extra) {
  %( $# > 1 %? if (tid() in trace) %)
  printf("%s%s%s %s\n",
         thread_indent (entry_p),
         (entry_p>0?"->":"<-"),
         probefunc (),
         extra)
}

probe $1.call   { trace(1, $$parms) }
probe $1.return { trace(-1, $$return) }
```

Example 2 continued:

```
# echo "hi"|./a.out

# sudo stap ./ex2.stp 'process("a.out").function("*")'
:)
     0 a.out(18123):->main argc=0x1 argv=0x7fff351ee0c8
    30 a.out(18123): ->readline fp=0x3f7bb516a0 buf=0x7fff351ecfd0 len=0x1000
   590 a.out(18123): <-readline return=0x7fff351ecfd0
   611 a.out(18123): ->reverse_line line=0x7fff351ecfd0 l=0x3
   625 a.out(18123): <-reverse_line return=0x7fff351ecfd0
   642 a.out(18123): ->write_line line=0x7fff351ecfd0
   731 a.out(18123): <-write_line
   748 a.out(18123): ->readline fp=0x3f7bb516a0 buf=0x7fff351ecfd0 len=0x1000
   762 a.out(18123): <-readline return=0x0
   770 a.out(18123):<-main return=0x0
```

Example 3: 获取行长度

```
global line_len

probe process("a.out").statement("reverse_line@tac.c+1"){
  line_len <<< ($e - $s + 2);
}

probe end{
  if(@count(line_len) >0) print(@hist_linear(line_len, 8, 128, 8));
}
```

Example 3 continued:

```
# ls -al|./a.out
# ./ex3.stp
:)
value |-------------------------------------------------- count
   <8 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                   64
    8 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                 69
   16 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                 68
   24 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                 68
   32 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                 68
   40 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                 68
   48 |@@@@@@@@@@@@@@@@@@@@@@@@@                          50
   56 |                                                    0
   64 |                                                    0
```

Example 4: 行反转平均时间

```
global t, call_time

probe process("a.out").function("reverse_line"){
  t = gettimeofday_ns()
}

probe process("a.out").function("reverse_line").return{
  call_time <<< (gettimeofday_ns() - t)
}

probe end{
  if(@count(call_time) > 0) printf("avg reverse_line execute time: %d
ns\n", @avg(call_time))
}
```

Example 4 continued :

```
# ls -al|./a.out
# ./ex4.stp
:)
avg reverse_line execute time: 6651 ns
```

Example 5: 列出调用栈

```
probe process(@1).function(@2){
  print_ubacktrace();
  exit();
}
```

Example 5 continued:

```
# ls -al|./a.out
# stap ./ex5.stp './a.out' '*_line'
:)
 0x40066d : reverse_line+0xc/0x61 [a.out]
 0x40078f : main+0xaf/0x100 [a.out]
 0x3bd441d994 [libc-2.5.so+0x1d994/0x357000]
```

Example 6: 修改程序的行为

```
global line
function alert_line(line:long) %{ /* pure */
  strcpy((char*)THIS->line, "abcdefg\n");
%}

probe process("a.out").function("reverse_line"){
  line = user_string($line);
}

probe process("a.out").function("reverse_line").return{
  if(isinstr(line, "tac")) $return = $haha;
  else if (isinstr(line, "hello")) alert_line($return);
}
```

Example 6 continued:

```
# stap ./ex6.stp

# echo tac|./a.out
wahaha
# echo hello|./a.out
abcdefg
# echo world|./a.out
dlrow
```

Agenda

介绍SystemTap

安装和系统要求

实践例子

参考和杂项

结论

Emacs Systemtap mode

- 在这里下载 systemtap-mode.el: http://coderepos.org/share/browser/lang/elisp/systemtap-mode/systemtap-mode.el?format=txt
- 在.emacs里面添加以下二行:
  - (autoload 'systemtap-mode "systemtap-mode")
  - (add-to-list 'auto-mode-alist '("\.stp$" . systemtap-mode))

参考文献

http://sourceware.org/systemtap/langref/

http://sourceware.org/systemtap/tapsets/

http://baike.corp.taobao.com/images/d/df/Systemtap-haxogreen-2010072301.pdf

http://sourceware.org/systemtap/wiki/AddingUserSpaceProbingToApps

http://github.com/posulliv/stap

http://www.slideshare.net/posullivan/monitoring-mysql-with-dtracesystemtap

Agenda

介绍SystemTap

安装和系统要求

实践例子

参考和杂项

结论

结论

SystemTap is often described as "DTrace for Linux".

OProfile takes sample every $N CPU cycles so you can try to fi gureout what each CPU is spending its time on.

SystemTap，居家必备！！！

谢谢大家！

Any question?