SystemTap Básico

(ou Como mergulhar no Kernel sem se afogar)

João Avelino Bellomo Filho
Linux Developer Conference Brazil

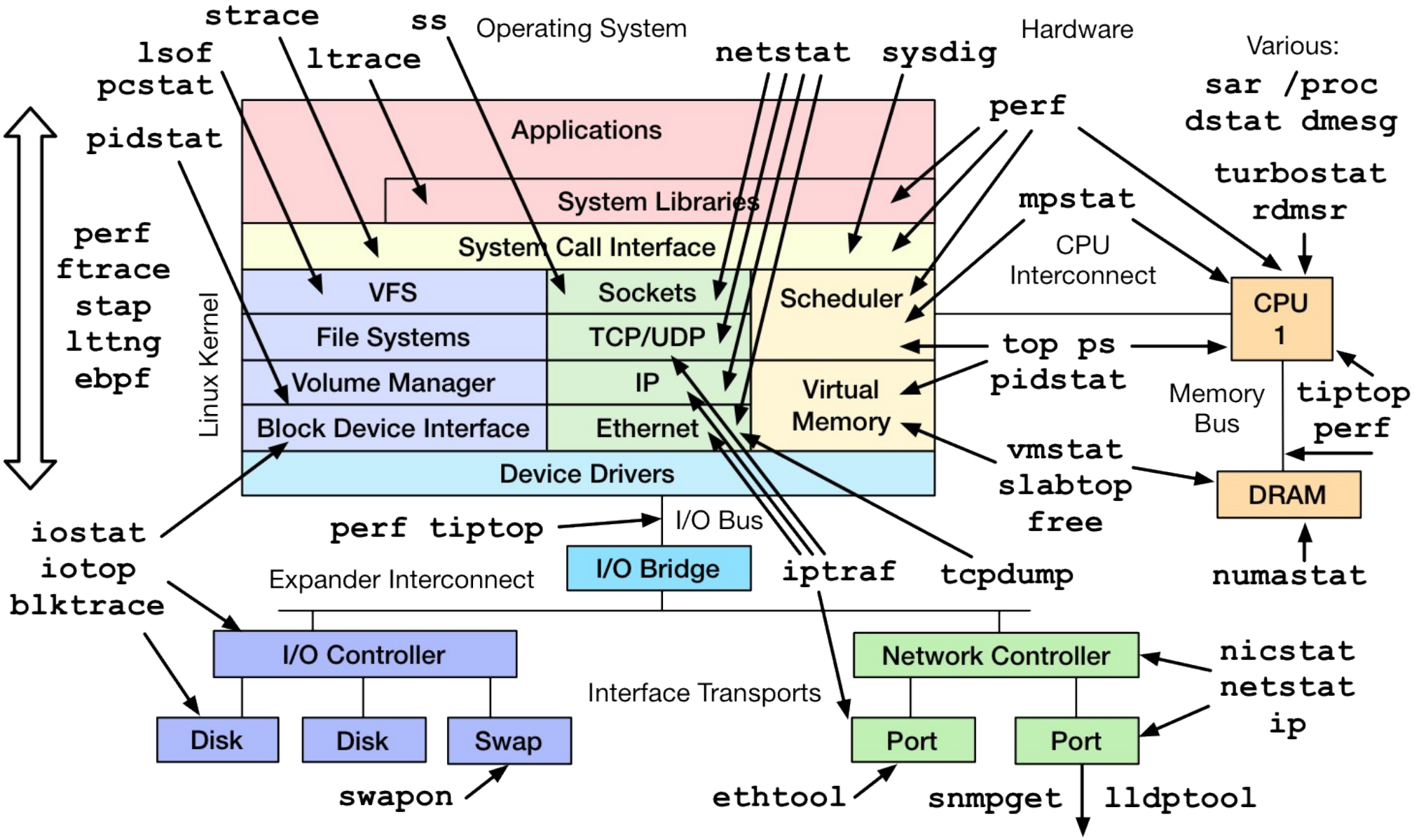
Debug no Linux

- No espaço do usuário (*user space*):
  - strace, ltrace, lsof, systat, etc...

- No espaço do kernel (*kernel space*):
  - ps, ftrace, perf, vmstat, ss, etc...

Debug no Linux

Linux Performance Observability Tools



Debug no Linux

- Para maior precisão:
  - Acesso aos fontes
  - Definição de *breakpoints*
  - Instrumentação, por meio de inserção de código de *debug*
- Procedimentos:
  - No *user space*: recompilar e executar
  - No *kernel space*: recompilar, instalar, reiniciar

SystemTap

É uma ferramenta de monitoramento e investigação das atividades do sistema, principalmente o kernel do Linux.

Por que usar?

- Permite acompanhar eventos no *kernel space* sem a necessidade de inserir código, recompilar, instalar e reiniciar o sistema.
- Possibilita a administradores e desenvolvedores identificar a causa de *bugs* ou problemas de performance.
- Permite que a investigação e monitoração de grande parte das funções do kernel, chamadas de sistema e outros eventos, possa ser feita por meio de *scripts* simples.

Recursos

- Monitoração de subsistemas do kernel
- Monitoração de eventos do *user space*.
- Criação de *breakpoints*

A Linguagem de *script*

- Estruturas da linguagem:

  `if``(`*`expr`*`)` *`statement`* **`else`** *`statement`*

  `while``(`*`condition`*`)` *`statement`*

  `for``(`*`expr`*`;` *`expr`*`;` *`expr`*`)` *`statement`*

  `etc ...`

- Declaração de variáveis (de tipos simples ou complexos como *strings* e *arrays*);
- Declaração de funções
- Declaração de sondas (*probes*)

Pontos de sondagem – *kernel space*

- probe kernel
  - kernel.function("xxx")
  - kernel.statement("xxx.c:nnn")
  - kernel.function("xxx").return
  - ...
- probe module
  - module("xxx").function("xxx")
  - module("xxx").statement("xxx.c:nnn")
  - ...

Pontos de sondagem – *kernel space*

- Variações:
  - kernel.function("xxx")
  - kernel.function("xxx").call
  - kernel.function("xxx").return
  - kernel.function("xxx").inline
  - ...

Pontos de sondagem – *user space*

- probe process
  - process("xxx").function("xxx")
  - process("xxx").statement("xxx.c:nnn")
  - process("xxx").syscall
  - process("xxx").thread

Pontos de sondagem – *user space*

- Variações
  - process("xxx").function("xxx").call
  - process("xxx").function("xxx").return
  - process("xxx").thread.begin
  - process("xxx").thread.end
  - process("xxx").syscall.begin
  - process("xxx").syscall.end
  - ...

Sondas especiais

- probe begin
- probe end
- probe error

Funções e tapsets

- De contexto:
  - backtrace(), caller(), cpu(), execname(), etc...
- Exibição de dados:
  - print(), printk(), etc...
- De Rede:
  - htol(), format_ipaddr(), etc ...
- Tratamento de strings:
  - strlen(), strtol(), etc...
- etc...

tapsets

- Biblioteca de sondas e funções
- Systemtap scripts
- Funções definidas no systemtap
- Código fonte em C (entre '%{' e '%}')
- Localizados no diretório: `/usr/share/systemtap/tapset`

Exemplos de tapset

/usr/share/systemtap/tapset/string.stp

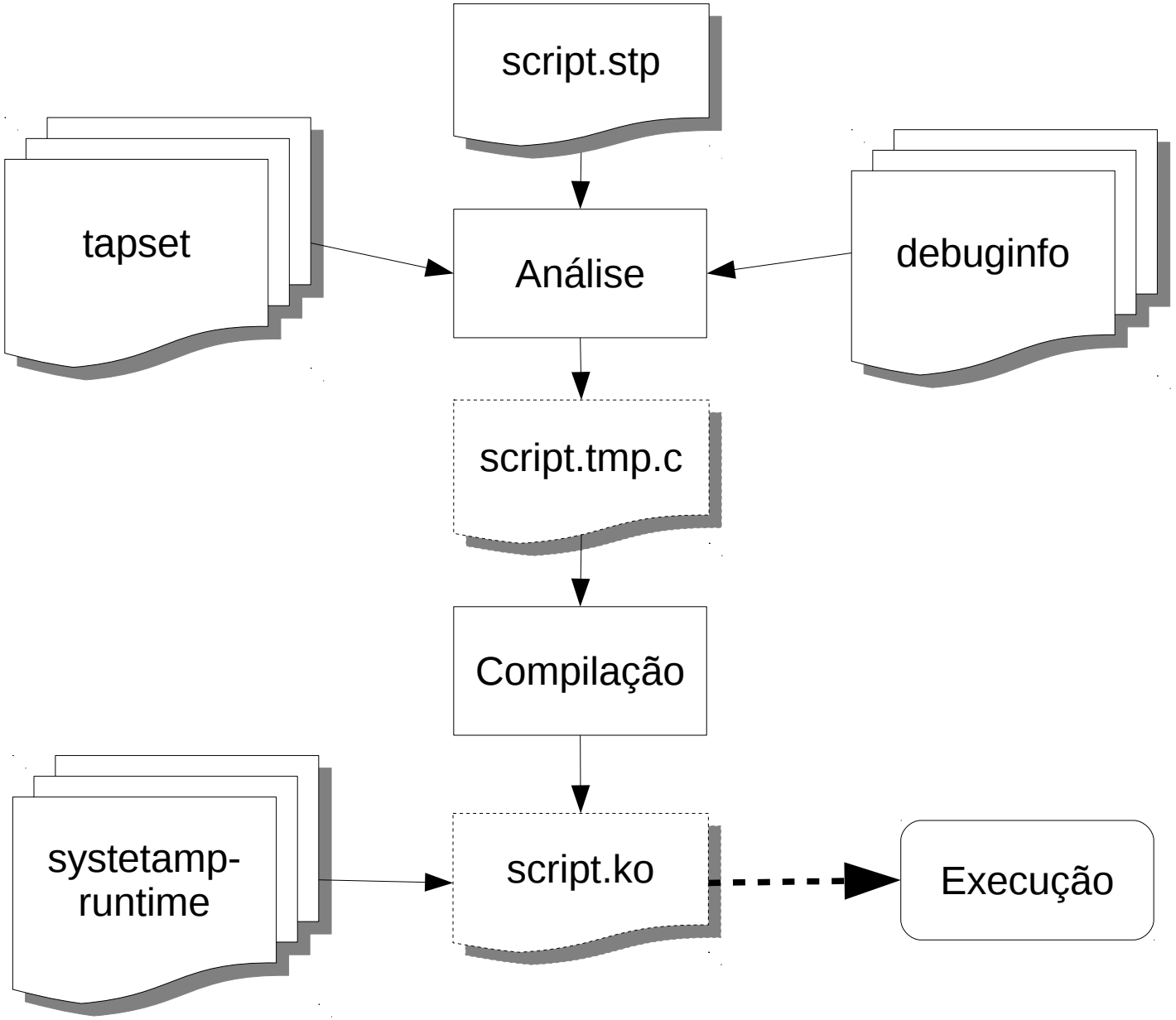
```
...
function substr:string(
          str:string, start:long, length:long)
%{ /* pure */ /* unprivileged */
    int64_t length = clamp_t( int64_t,
             STAP_ARG_length + 1, 0, MAXSTRINGLEN);
    if (STAP_ARG_start >= 0 &&
        STAP_ARG_start < strlen(STAP_ARG_str))
        strlcpy(STAP_RETVALUE,
             STAP_ARG_str + STAP_ARG_start, length);
%}
...
```

Exemplos de tapset

/usr/share/systemtap/tapset/linux/networking.stp

```
...
probe netdev.receive
        =  kernel.function("netif_receive_skb")
{
        dev_name = kernel_string($skb->dev->name)
        length = $skb->len
        protocol = $skb->protocol
        truesize = $skb->truesize
}
...
```

Funcionamento



Instalação

- Por exemplo, no Fedora 26:
  - Instalação:

```
dnf -y install systemtap systemtap-runtime
dnf config-manager --set-enabled fedora-debuginfo
dnf -y install --enablerepo=* \
        kernel-debuginfo-$(uname -r) \
        kernel-devel-$(uname -r)
```

  - Teste:

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

O teste

```
# stap -v -e 'probe vfs.read {printf( ...
Pass 1: parsed user script and 111 library script(s)
using 218600virt/41708res/6584shr/35640data kb, in
110usr/10sys/120real ms.
Pass 2: analyzed script: 1 probe(s), 1 function(s), 4
embed(s), 0 global(s) using
369528virt/187996res/8072shr/186568data kb, in
1310usr/210sys/1531real ms.
Pass 3: using cached
/root/.systemtap/cache/e2/stap_e27fde8...5_1624.c
Pass 4: using cached
/root/.systemtap/cache/e2/stap_e27fde8...5_1624.ko
Pass 5: starting run.
read performed
Pass 5: run completed in 0usr/20sys/331real ms.
```

Listando sondas (probes)

- Verificando se existe:

```
# stap -l 'kernel.function("run_cmd")'
```

```
kernel.function("run_cmd@kernel/reboot.c:392")
```

- Visualizando os argumentos e variáveis locais:

```
# stap -L 'kernel.function("run_cmd")'
```

```
kernel.function("run_cmd@kernel/reboot.c:392")
$cmd:char const* $envp:char*[]
```

- Procurando de acordo com um padrão:

```
# stap -l 'kernel.function("SYSC*")'

...
```

Exemplos de Scripts

begin-end.stp

```
probe begin {
  printf( "\n\nIniciando monitoração ...\n\n" );
  println( "Pressione Ctrl+C para encerrar.");
}

probe end {
  println( "Encerrando monitoramento." );
}
```

begin-end-timer.stp

```
probe begin {
  printf( "\n\nIniciando monitoração ...\n\n" );
  println( "Encerra em 5 segundos." );
}

probe timer.s(5) {
  exit();
}

probe end {
  println( "Encerrando monitoramento." );
}
```

begin-end-contador.stp

```
global contador=0;

probe begin {
  printf( "\n\nIniciando monitoração ...\n\n" );
  println( "Encerra em 5 segundos." );
}
probe timer.s(1) {
  contador++;
  println( contador );
}
probe timer.s(5) {
  exit();
}
probe end {
  println( "Encerrando monitoramento." );
}
```

ls-functions.stp

```
probe begin {
  printf( "\n\nMonitorando o comando ls ...\n\n" );
}

probe process("ls").function("*") {
  println( ppfunc() );
}

probe end {
  println( "Encerrando monitoramento." );
}
```

ls-syscall.stp

```
probe begin {
  printf( "\n\nMonitorando os comandos ls ...\n\n" );
}

probe syscall.* {
  if( execname() == "ls" )
    printf( "%d %s\n", pid(), name );
}

probe end {
  println( "Encerrando monitoramento." );
}
```

e1000-module.stp

```
probe begin {
  printf( "\n\nMonitorando modulo e1000 ...\n\n" );
}

probe module("e1000").function("*") {
  printf( "%s\n", ppfunc() );
}

probe end {
  println( "Encerrando monitoramento." );
}
```

e1000-module-all.stp

```
probe begin {
  printf( "\n\nMonitorando modulo e1000 ...\n\n" );
}

probe module("e1000").function("*").call {
  printf( "[%s] ENTER  (%s)\n", ppfunc(), $$parms$$ );
}

probe module("e1000").function("*").return {
  printf( "[%s] RETURN (%s)\n", ppfunc(), $$return$$ );
}

probe end {
  println( "Encerrando monitoramento." );
}
```

Outros Exemplos

https://sourceware.org/systemtap/examples/

- process/strace.stp
- network/netfilter_summary.stp
- process/proctop.stp
- etc ...

Questões?

References

- http://www.brendangregg.com/linuxperf.html
- https://www.centos.org/docs/5/html/Deployment_Guide-en-US/s2-systemtap-implementation.html
- https://sourceware.org/systemtap/
- https://sourceware.org/systemtap/wiki/SystemtapOnFedora
- https://www.sourceware.org/systemtap/wiki/
- https://sourceware.org/systemtap/wiki/HomePage
- http://www.redhat.com/magazine/011sep05/features/systemtap
- https://fedoraproject.org/wiki/Building_a_custom_kernel