

Linux - dtrace, driver de video

Projeto Fase 03

Cassius Puodzius, Lucas Baraças, Marcelo Risse

2º Quadrimestre 2013

2 de Julho de 2013

Conteúdo

1	Introdução	2
2	Dtrace	2
2.1	Funcionalidades do Dtrace	2
2.2	Arquitetura do Dtrace	2
2.2.1	Dtrace Providers	2
2.2.2	Dtrace probe	3
2.2.3	Dtrace Predicates	3
2.2.4	Dtrace Actions	4
2.3	Scripts	4
2.3.1	fork.d	5
2.3.2	syscalls.d	5
3	Webcam no Linux	5
3.1	V4L2 - API de Multimídia no Linux	6
3.2	Aspectos dos V4L2 Drivers	6
4	Dtrace - monitorando o driver da webcam	7

1 Introdução

O dtrace é uma modificação no kernel do Linux feita originalmente pela Sun com o objetivo de facilitar o debugging de aplicativos, observando dados do kernel. Esta ferramenta proporciona colocar probes dentro do kernel e apresentar os resultados dependendo de certas circunstâncias desejadas.

Usando o dtrace pode-se monitorar as leituras e escritas em um driver, com isso, usaremos o dtrace para monitorar o driver de uma webcam a fim de observar quantos bytes são repassados da câmera por segundo.

2 Dtrace

O Dtrace foi desenvolvido pela Sun Microsystems, inicialmente para o sistema operacional Solaris10 em 2005. Desde então, a ferramenta Dtrace foi desenvolvida para funcionar em outros sistemas operacionais, como o FreeBSD, NetBSD, Mac OS X (a partir do Leopard) e em 2008 também passou a integrar o Linux. A implementação Linux DTrace é um módulo de kernel carregável, dessa forma o próprio núcleo não tem de ser modificado. No entanto, uma vez que o DTrace é carregado a instância do kernel será marcada como contaminada.

2.1 Funcionalidades do Dtrace

O Dtrace é um framework de rastreamento tanto estático quanto dinâmico. O Dtrace é uma ferramenta utilizada para analisar o comportamento de programas de usuário e do próprio sistema operacional como quantidade de memória, tempo de CPU, sistemas de arquivos e recursos de rede utilizados. O funcionamento do Dtrace é baseado em probes (provas), que são fired (acionadas) dependendo do que estiver ocorrendo no sistema monitorado. Com as probes é possível analisar diversos aspectos como: argumentos passados em uma função, qualquer variável global do Kernel, a data e hora que uma função foi chamada, informações da pilha, entre outros.

2.2 Arquitetura do Dtrace

2.2.1 Dtrace Providers

Os providers tornam as probes disponíveis para o framework do Dtrace. No caso, o Dtrace envia uma informação ao provider de quando ativar a probe. O provider por sua vez, quando uma probe é acionada, transfere o controle para o Dtrace. A seguir alguns exemplos de providers:

- DTrace: BEGIN, END, ERROR probes
- Syscall: entry e exit para cada system call
- Profile: ativa um intervalo específico de tempo (provas dinâmicas)

- sysinfo, vmstat, iostat, etc.: Rastreamento estático no Kernel: probes ficam em lugares específicos do sub-sistema
- Pid: Rastreamento estático em programas do usuário: MySQL, Perl, Java
- Pid: Rastreamento dinâmico em aplicações de usuário: pode fazer uma probe a cada instrução de um processo

2.2.2 Dtrace probe

As probes no Dtrace tem um formato definido, com 4 atributos:

- provider
- module
- function
- name

organizados da seguinte maneira: **provider:module:function:name**.

Desta forma, para cada campo que não for especificado, o Dtrace entenderá como all. Assim caso colocássemos a seguinte probe. `:::1` seria o mesmo que fazer `Dtrace -1`, e listará todas as probes.

Uma outra especificação, é que só é necessário manter os campos vazios que estão mais a direita. ou seja, uma probe sem especificação de provider e module poderia ser apenas escrita por: `function:name`, como no ex: `read:entry`, ao invés de `:::read:entry`. A seguir um exemplo de probe, em que são definidos apenas o provider e o nome.

```
syscall::entry           // probe description
/pid == $1/              // predicate
{
    @[probefunc] = count(); // action
}
```

2.2.3 Dtrace Predicates

Os predicados são expressões primárias entre barras, como no ex: `/pid == $1/`, que pode ser visto no seguinte script.

```
cat -n syscalls2.d
1  #!/usr/sbin/dtrace -qs
2
3  syscall::entry
4  /pid == $1/
5  {
6      @[probefunc] = count();
7  }
8  syscall::rexit:entry
9  {
10     exit(0);
11 }
```

¹Todos os campos em vazio.

O predicado pode utilizar variáveis e constantes. Assim como no restante da linguagem D, 0 é falso e qualquer outro valor é verdadeiro.

2.2.4 Dtrace Actions

As ações no Dtrace tem as seguintes especificações:

- Ações são feitas quando uma probe é atingida
- Ações são completamente programáveis em linguagem D
- A maioria das ações gravam um estado específico do sistema
- Algumas ações podem mudar o estado do sistema. Essas ações são chamadas de destructive actions. Esse tipo de ação não é permitido por default.²

2.3 Scripts

Segundo a estrutura de script em linguagem D apresentada anteriormente, apresentamos dois scripts como exemplo.

²Esse tipo de ação não será abordado neste trabalho.

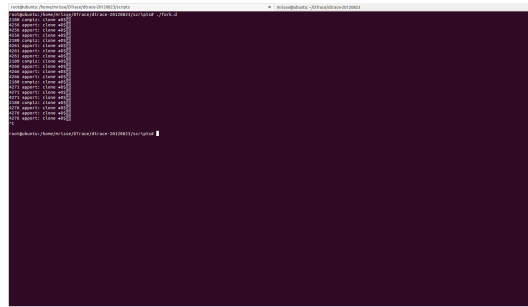


Figura 1: Execução do fork.d ao abrir uma pasta

2.3.1 fork.d

O primeiro, chamado de *fork.d*, utiliza o provider syscall e analisa as system calls fork, vfork e clone no início de suas execuções. A cada system call encontrada o script imprime algumas informações do processo que chamou a system call: *pid*, *execname* (nome da system call), *probefunc* (função da ponta de prova do DTrace).

A figura 1 mostra a execução do *fork.d* ao abrir uma pasta.

```
dtrace -n '
#pragma D option quiet
syscall::fork*:entry,
syscall::vfork*:entry,
syscall::clone*:entry
{
    printf("%d %s: %s %s\n", pid, execname, probefunc, copyinstr(arg0));
}
'
```

2.3.2 syscalls.d

O segundo exemplo também usa o provider syscall, mas analisa todas as system calls executadas no sistema operacional. A variável @num[probefunc] é um vetor das pontas de prova e a contagem de suas execuções, a contagem é feita pela função count().

```
#!/bin/sh
# Syscalls by function/process
echo "System-calls ordered by frequency. Press Ctrl-C to terminate and view."
set -x
dtrace -n 'syscall:::entry { @num[probefunc] = count(); }'
```

3 Webcam no Linux

As primeiras webcams foram desenvolvidas para funcionar em portas seriais, no entanto, com o advento das portas USBs, as webcams passaram a focar sua utilização nessas portas, principalmente devido à facilidade para a instalação do driver. Essas câmeras pertencem à classe de câmeras *UVC* (USB Video Class) e possuem uma API específica

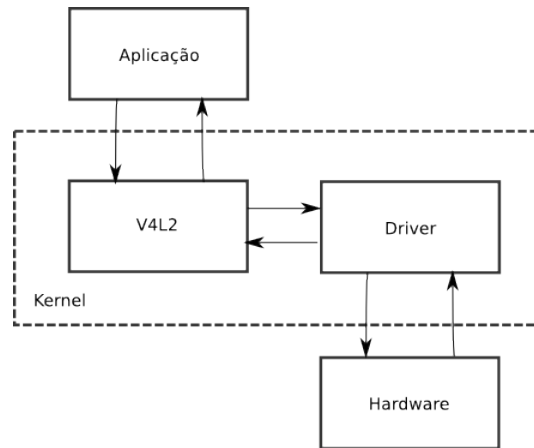


Figura 2: V4L2 - API para multimídia no Linux

para o Linux, que permite a implementação dos drivers das câmeras e, que em conjunto com o driver da USB, permite o *plug-and-play* de câmeras no Linux.

3.1 V4L2 - API de Multimídia no Linux

O V4L2 (Video for Linux Two) é uma API para multimídia no Linux que faz a interface entre a aplicação (modo usuário) e, no escopo do nosso trabalho, o driver da câmera (modo kernel). A API funciona como mostrado na Fig. 2, onde é possível ter uma visão geral do funcionamento do V4L2. A aplicação utiliza a API V4L2, enquanto o driver se encarrega de implementar os detalhes para o controle do hardware de vídeo. A comunicação entre a aplicação e o V4L2 é feita, basicamente, utilizando-se syscalls para I/O's de arquivos e ioctl's (I/O control). Os dispositivos V4L2 são nós do sistema de arquivos do sistema (geralmente em `/dev`), que permitem a troca de informações entre a aplicação e hardware (driver) através das syscalls `write` e `read`, enquanto que as mensagens de ioctl são feitas diretamente na memória do processo (mediada pelo Linux), e permitem uma performance muito maior a custo de menores mensagens. Dessa forma, a aplicação que faz uso da câmera vai receber os dados da câmera através da syscall `read` e realizar os controles através da syscall `ioctl`.

3.2 Aspectos dos V4L2 Drivers

O drivers implementados para a interface V4L2 devem prover algumas funcionalidades obrigatórias, como o suporte para `read` and `write`, por exemplo. A aplicação usuária deve poder controlar como o exemplo a seguir:

```

> vidctrl /dev/video --input=0 --format=YUYV --size=352x288
> dd if=/dev/video of=myimage.422 bs=202752 count=1

```

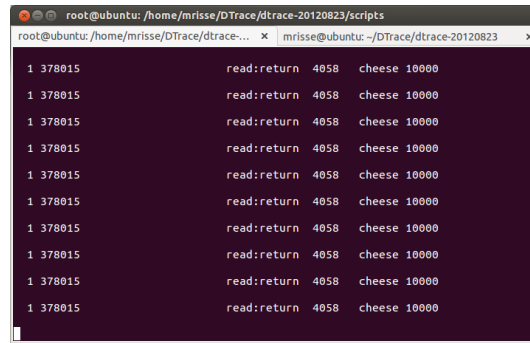
No exemplo acima, *vidctrl* é uma aplicação fictícia (retirada de [5]) e na linha seguinte frames de imagens da webcam são lidos utilizando-se o comando *dd*. Como vemos, a troca dos dados é feita mediante arquivo, nó do sistema de arquivos.

4 Dtrace - monitorando o driver da webcam

Nossa estratégia foi monitorar a leitura dos frames no V4L2. A aplicação escolhida para fazer uso da webcam foi o *Cheese*, que como explicado na seção 3.1, realiza a leitura dos frames através da syscall *read* em um dispositivo. Escrevemos, então, um script que monitorava o retorno da função *read* feita pelo *Cheese*, pois o retorno contém o número de bytes que foram lidos efetivamente. Assim pudemos nos desviar dos detalhes de implementação do driver e monitorar o driver através da interface. A seguir apresentamos o script de monitoramento do driver da webcam, escrito pelo grupo:

```
dtrace -n '  
#pragma D option strsize=10000  
syscall::read:entry  
/execname == "cheese"/  
{  
    self->buf = arg1;  
    self->len = arg2;  
}  
  
syscall::read:return  
/self->buf != NULL/  
{  
    this->text = (char *)copyin(self->buf, self->len);  
  
    printf("%5d %8s %d\n", pid, execname, sizeof(stringof(this->text)));  
    self->buf = NULL;  
    self->len = 0;  
}  
,
```

O script funciona copiando a string lida pelo *read* na string *self->text* e depois verificando o tamanho dessa string, mediante *sizeof(...)*. É importante observar que o tamanho padrão para strings no Dtrace é de 256B, por isso tivemos que reconfigurar esse parâmetro (escolhemos deixar 10KB após alguns testes) usando *#pragma D option strsize=10000*. Com isso, obtivemos reads de 4058 bytes, como mostrado na figura 3.



```
root@ubuntu: /home/mrisse/DTrace/dtrace-20120823/scripts
1 378015 read:return 4058 cheese 10000
1 378015 read:return 4058 cheese 10000
1 378015 read:return 4058 cheese 10000
1 378015 read:return 4058 cheese 10000
1 378015 read:return 4058 cheese 10000
1 378015 read:return 4058 cheese 10000
1 378015 read:return 4058 cheese 10000
1 378015 read:return 4058 cheese 10000
1 378015 read:return 4058 cheese 10000
1 378015 read:return 4058 cheese 10000
1 378015 read:return 4058 cheese 10000
1 378015 read:return 4058 cheese 10000
```

Figura 3: Screenshot do monitoramento do driver de video pelo Dtrace

Referências

- [1] illumos dynamic tracing guide. <http://dtrace.org/guide/preface.html>.
- [2] Oracle Corporation and/or its affiliates. Dtrace user guide. <http://docs.oracle.com/cd/E19253-01/819-5488/>, 2010.
- [3] dtrace.org. About dtrace. <http://dtrace.org/blogs/about/>.
- [4] Brendan Gregg and Jim Mauro. Dtrace: Dynamic tracing in oracle solaris, mac os x, and freebsd. http://www.dtracebook.com/index.php/Main_Page, 2011.
- [5] Martin Rubli. Building a webcam infrastructure for gnu/linux. Master's thesis, School of Computer and Communication Sciences, Swiss Federal Institute of Technology, Lausanne, Switzerland, 2006.
- [6] Rickey C. Weisner. Tutorial: Dtrace by example. <http://www.oracle.com/technetwork/server-storage/solaris/dtrace-tutorial-142317.html>, 2009.
- [7] Colin Wheeler. Dtrace for cocoa developers. <http://www.viddler.com/v/4bc1e00f>, 2008.