ESC-TP2

José Pinto A81317 - Pedro Barbosa A82068

I. Introdução

O Dtrace é uma ferramenta que permite a monitorização e a análise de performance não só de programas em específico mas também do sistema operativo em si.

Como com qualquer ferramenta, de forma a utilizar o Dtrace para analisar um programa, é necessário primeiro adquirir conhecimentos sobre o seu funcionamento e as suas capacidades. A melhor forma de adquirir esses conhecimentos é através de experiência prática.

Os exercícios abordam a invocação de chamadas de sistema, especialmente a abertura de ficheiros.

A resolução das perguntas é orientada ao Solaris 11. A utilização noutros sistemas requer algumas modificações.

II. Pergunta 1

Para obter informação sobre a abertura de ficheiros, são utilizadas as *probes* relativas à *system call openat*.

O nome do executável, o PID do processo, o UID do utilizador e o GID do grupo podem ser obtidos através das respetivas variáveis *built-in*.

```
syscall::openat:entry
{
    self->execname = execname;
    self->pid = pid;
    self->uid = uid;
    self->gid = gid;
    self->path = copyinstr(arg1);
}
```

Segundo a documentação da chamada *openat*, o caminho do ficheiro a abrir corresponde ao segundo argumento, que na sintaxe do dtrace corresponde a *arg1*. Como o *arg1* corresponde a um apontador da memória, é necessário invocar *copyinstr* para obter o conteúdo da string.

O inteiro *arg2* corresponde às *flags* com que o ficheiro é aberto. É possível determinar se uma *flag* foi utilizada observando os bits correspondentes. Para isso é utilizado o operador "&" que permite fazer *match* entre os bits da *flag* utilizada e os bits de uma das possíveis *flags*.

O argumento tem que incluir obrigatoriamente umas das *flags* do modo de abertura, "O_RDONLY", "O_WRONLY" ou "O_RDWR"

```
syscall::openat:entry
/(arg2 & 1) == 0 && (arg2 & 2) == 0/
{
    self->flags = "O_RDONLY";
}

syscall::openat:entry
/(arg2 & 1) == 1 && (arg2 & 2) == 0/
{
    self->flags = "O_WRONLY";
}

syscall::openat:entry
/(arg2 & 2) == 2/
{
    self->flags = "O_RDWR";
}
```

Existem diversas *flags* opcionais. Dentro destas, o *script* deteta a utilização de "O_APPEND" e de " O_CREAT".

```
syscall::openat:entry
/(arg2 & 8) == 8/
{
    self->flags = strjoin(self->flags,
       ", O_APPEND");
}

syscall::openat:entry
/(arg2 & 256) == 256/
{
    self->flags = strjoin(self->flags,
       ", O_CREAT");
}
```

Caso uma destas *flags* opcionais seja utilizada é necessário fazer *strjoin* para adicionar a *flag* à *flag* obrigatória.

O valor de retorno da chamada é dado pelo *arg0* da *probe syscall::openat:return*. Também é utilizado o disparo dessa *probe* para imprimir a informação reunida.

```
syscall::openat:return
{
printf("Nome do executavel: %s \nPID
   do processo: %d \nUID do grupo: %d
   \nGID do grupo: %d \nPath:
   %s\nFlags:%s \nValor de retorno: %d
   \n",
    self->execname,
    self->pid,
    self->uid,
    self->gid,
    self->path,
    self->flags,
    arg0
    );
}
```

Para apenas detetar ficheiros com *etc/* no caminho basta adicionar a seguinte condição na última *probe*, sendo *strstr* uma função que retorna a localização da segunda string na primeira.

```
strstr(self->path, "etc/") != NULL
```

De seguida apresenta-se o output relevante da execução do comando "cat /etc/inittab >/tmp/test", sendo possível observar uma versão detalhada nos anexos:

```
Nome do executavel: bash
PID do processo: 22829
UID do grupo: 1015
GID do grupo: 5000
Path: /tmp/teste
Flags:O_WRONLY, O_CREAT
Valor de retorno: 4
--------------------------------------
Nome do executavel: cat
PID do processo: 22829
UID do grupo: 1015
GID do grupo: 5000
Path: /etc/inittab
Flags:O_RDONLY
Valor de retorno: 3
```

Output relevante da execução do comando "cat /etc/inittab ≫ /tmp/test":

```
Nome do executavel: bash
PID do processo: 22973
UID do grupo: 1015
GID do grupo: 5000
Path: /tmp/teste
Flags:O_WRONLY, O_APPEND, O_CREAT
Valor de retorno: 4
--------------------------------------
Nome do executavel: cat
PID do processo: 22973
UID do grupo: 1015
GID do grupo: 5000
Path: /etc/inittab
Flags:O_RDONLY
Valor de retorno: 3
```

Output relevante da execução do comando "cat /etc/inittab | tee /tmp/test":

```
Nome do executavel: cat
PID do processo: 23027
UID do grupo: 1015
GID do grupo: 5000
Path: /etc/inittab
Flags:O_RDONLY
Valor de retorno: 3
--------------------------------------
Nome do executavel: tee
PID do processo: 23028
UID do grupo: 1015
GID do grupo: 5000
Path: /tmp/teste
Flags:O_WRONLY, O_CREAT
Valor de retorno: 3
```

Output relevante da execução do comando "cat /etc/inittab | tee -a /tmp/test":

```
Nome do executavel: cat
PID do processo: 23066
UID do grupo: 1015
GID do grupo: 5000
Path: /etc/inittab
Flags:O_RDONLY
Valor de retorno: 3
--------------------------------------
Nome do executavel: tee
PID do processo: 23067
UID do grupo: 1015
GID do grupo: 5000
Path: /tmp/teste
Flags:O_WRONLY, O_APPEND, O_CREAT
Valor de retorno: 3
```

III. Pergunta 2

A. Alínea A

```
BEGIN{
    APPEND = 8;
    CREAT = 256;
}

syscall::openat:entry
{
    @attempts[execname, pid, "opens"]
       = count();


    creation_flags = "";

    ((arg2 & CREAT) != CREAT) ?
        creation_flags = "open
           existing file" : 0;

    ((arg2 & CREAT) == CREAT) ?
        creation_flags = "create new
           file" : 0;

    @attempts[execname, pid,
       creation_flags] = count();
}

syscall::openat:return
/arg0 >= 0/
{
    @attempts[execname, pid,
       "successful"] = count();
}
```

À semelhança da primeira pergunta, o segundo argumento (*arg2*) representa as *flags* com que o ficheiro é aberto, podendo ser utilizado o mesmo método para determinar qual das *flags* é utilizada.

A agregação *attempts* é utilizada para contabilizar a informação sobre as aberturas dos ficheiros. A chave "opens" corresponde às tentativas de abrir um ficheiro.

Se a flags O_CREAT for usada, é registada uma tentativa de criar um ficheiro novo. Caso contrário, é registada uma tentativa de abrir um ficheiro existente.

Se a *probe return* disparar e o valor de retorno seja não negativo então a tentativa é considerada bem sucedida.

B. Alínea B

```
tick-$1s
{
    printf("\%Y\n", walltimestamp);
    printa(@attempts);
}
```

A proposta da alínea B era imprimir os resultados obtidos anteriormente de X em X segundos, período de tempo indicado pelo utilizador, acompanhado da data e hora atual. Para isso, foi adicionada uma nova *probe tick* que contém dois *prints*, um para a data e hora atual (foi utilizado o walltimestamp para ir buscar esta informação) e outro para imprimir a agregação. O $1 corresponde ao intervalo de tempo introduzido pelo utilizador.

Resultado de uma iteração:

```
2020 May 21 22:36:39
```

execname	pid	action	amount
vmtoolsd	1212	create new file	1
dtrace	14118	open existing file	2
dtrace	14118	opens	2
dtrace	14118	successful	2
sshd	708	open existing file	3
sshd	708	opens	3
sshd	708	successful	3
sshd	14121	open existing file	3
sshd	14121	opens	3
sshd	14121	successful	3
sstored	905	open existing file	3
sstored	905	opens	3
sstored	905	successful	3
sshd	14119	successful	50
sshd	14119	open existing file	56
sshd	14119	opens	56
vmtoolsd	1212	successful	328
vmtoolsd	1212	open existing file	337
vmtoolsd	1212	opens	338

IV. Pergunta 3

```
syscall:::entry
/pid == $target/
{
    self->ts = timestamp;
    @totalCount[probefunc] = count();
}

syscall:::return
/pid == $target/
{
    @totalTime[probefunc] =
       sum(timestamp - self->ts);
}

END
{
    printa(@totalTime, @totalCount);
}
```

Inicialmente, quando é invocada uma *system call*, é registada na variável *self->ts* o valor do tempo atual. Para além disso também é atualizada a agregação *totalCount* onde vão ser guardadas o número de execuções de uma *system call*. Quando a *system call* termina, calcula-se o tempo de execução e soma-se esse valor ao correspondente contido na agregação *totalTime*.

No final do programa, imprimem-se ambas as agregações.

Execname	Total time	Total count
rexit	0	1
sysconfig	5870	1
getpid	7241	1
lwp_private	7580	1
sigpending	9380	1
getrlimit	14797	1
resolvepath	16856	1
setcontext	22250	2
close	26242	3
memcntl	28438	2
write	31943	1
getdents	33072	2
ioctl	43309	3
mmapobj	43517	1
brk	47218	5
fstatat	53353	5
openat	143704	6
mmap	317398	2

V. Conclusão

Como a programação do Dtrace não corresponde na totalidade aos paradigmas que estamos mais habituados, a resolução dos exercícios foi essencial para obter uma maior familiaridade com o uso da ferramenta.

VI. Tabela da pergunta 1

Execname	PID	UID	GID	Path	Flags	Return value
bash	22829	1015	5000	/tmp/teste	O_WRONLY, O_CREAT	4
cat	22829	1015	5000	/var/ld/64/ld.config	O_RDONLY	-1
cat	22829	1015	5000	/lib/64/libc.so.1	O_RDONLY	3
cat	22829	1015	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8	O_RDONLY	4
cat	22829	1015	5000	/usr/lib/locale/common/amd64/methods_unicode.so.3	O_RDONLY	-1
cat	22829	1015	5000	/usr/lib/locale/pt_PT.UTF-8/pt_PT.UTF-8	O_RDONLY	-1
cat	22829	1015	5000	/usr/lib/locale/en_US.UTF-8/LC_MESSAGES/solaris_linkers.mo	O_RDONLY	-1
cat	22829	1015	5000	/usr/lib/locale/en_US.UTF-8/LC_MESSAGES/solaris_lib_libc.mo	O_RDONLY	3
cat	22829	1015	5000	/etc/inittab	O_RDONLY	3

TABLE I

Output para o comando cat /etc/inittab >/tmp/test