Engenharia de Sistemas de Computação

TP2: Dtrace - Desenvolvimento de programas

André Ramalho a76387

Vítor Gomes a75362

4 de Julho de 2020

1 Introdução

Neste relatório são apresentadas soluções para análise de processos ou do sistema em geral utilizando scripts para a ferramenta *DTrace*. Os exercícios resolvidos exploram, maioritariamente, a utilização do *provider* `syscall`, assim como, a utilização de agregações, variáveis de sondas, macros e funções *DTrace*.

2 Exercício 1

Neste exercício procura-se fazer o traçado da chamada de sistema `open()` e imprimir, para cada chamada, o nome do executável, PID, UID, GID, caminho para o ficheiro a abrir, flags usadas na abertura e o valor de retorno da chamada. Para além disto, decidiu-se que esta informação só deveria ser impressa quando se procura abrir um ficheiro com "/etc"no caminho.

Através do *provider syscall* e a sua sonda *entry*, sempre que a *system call* `openat` é iniciada, são guardados o caminho do ficheiro aberto e as flags usadas no *system call* `openat`.

```
syscall::open:entry
{
    self->file = copyinstr(arg0);
    self->flags = arg1;
}
```

Quando a *system call* `openat` termina, é verificado se a o ficheiro aberto encontra-se em "/etc/"através do predicado apresentado a seguir, e caso isto se verifique, são impressas as variáveis previamente guardadas, assim como, o nome do executável, PID, GID e UID do processo armazenados em variáveis predefinidas. Para imprimir as flags em formato textual é feita uma análise para determinar que flags foram utilizadas entre 5 possibilidades.

```
syscall::openat:return
/self->file != 0 && substr(copyinstr(self->file), 0, 5) ==
↪  "/etc/"/
{
        self->openMode = (!(self->flags & 3)) ? "O_RDONLY" : "";
        self->openMode = (self->flags & 1) ? "O_WRONLY" :
        ↪  self->openMode;
        self->openMode = (self->flags & 2) ? "O_RDWR" :
        ↪  self->openMode;
        self->ap = (self->flags & 8) ? " | O_APPEND" : "";
        self->cr = (self->flags & 256) ? " | O_CREAT" : "";

        printf(" %-11.11s  %5d   %5d   %5d   %-30.30s %3d
        ↪  %s%s%s\n", execname,pid,uid,gid,
        ↪  copyinstr(self->file),arg0, self->openMode,
        ↪  self->ap,self->cr);

        self->openMode = "";
        self->ap = "";
        self->cr = "";
}
```

Depois de imprimir,ou não, toda a informação pretendida, todas as variáveis são igualadas a 0 para libertar o espaço em memória.

```
syscall::openat:return
{
    self->file = 0;
    self->flags = 0;
}
```

O script completo encontra-se no apêndice A.

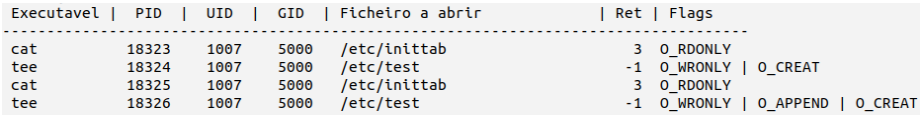
De seguida apresentam-se os resultados obtidos com diferentes comandos executados:

Executavel	PID	UID	GID	Ficheiro a abrir	Ret	Flags
bash	17926	1007	5000	/etc/test	-1	O_WRONLY | O_CREAT
cat	17927	1007	5000	/etc/inittab	3	O_RDONLY

Figura 1: Resultados obtidos com `cat /etc/inittab > /etc/test` e `cat /etc/inittab > test`

Executavel	PID	UID	GID	Ficheiro a abrir	Ret	Flags
bash	18217	1007	5000	/etc/test	-1	O_WRONLY | O_APPEND | O_CREAT

Figura 2: Resultados obtidos com `cat /etc/inittab >> /etc/test`



```
Executavel |  PID  |  UID  |  GID  | Ficheiro a abrir            | Ret | Flags
-------------------------------------------------------------------------------
cat          18323    1007    5000   /etc/inittab                   3  O_RDONLY
tee          18324    1007    5000   /etc/test                     -1  O_WRONLY | O_CREAT
cat          18325    1007    5000   /etc/inittab                   3  O_RDONLY
tee          18326    1007    5000   /etc/test                     -1  O_WRONLY | O_APPEND | O_CREAT
```

Figura 3: Resultados obtidos com `cat /etc/inittab | tee /etc/test` e `cat /etc/inittab | tee -a /etc/test`

3 Exercício 2

Neste exercício procura-se agregar estatísticas sobre o número de tentativas de abrir ficheiros existentes, o número de tentativas de criar ficheiros e o número das tentativas anteriores que foram bem sucedidas. Esta informação deve ser impressa, por PID e respetivo nome do comando, para períodos de um tempo em segundos passado como argumento do script, assim como a hora e o dia em que as estatísticas são impressas.

No script, começa-se por definir o valor do período que corresponde ao tempo, em segundos, que deve imprimir, repetidamente, hora e dia atual e as estatísticas recolhidas por PID e o respetivo nome do comando. O valor do período deve ser igual ao argumento da linha de comandos caso seja um número válido (maior que 0) ou o valor por omissão 5.

```
BEGIN
{
    period = ($1 > 0) ? $1 : 5;
    seconds = period;
}
```

DTrace fornece várias funções para agregar os dados que as sondas individuais recolhem. Para contar o número de tentativas para abrir e criar ficheiros foi usada a função count().

Para distinguir entre uma tentativa de abertura ou de criação de um ficheiro, considerou-se uma tentativa de criação de um ficheiro sempre que se chama `openat()` com as seguintes flags: O_CREAT(256), O_WRONLY(1) e O_TRUNC(512). Esta decisão foi feita na base de que uma chamada com estas flags é equivalente a uma chamada da função `creat()`. Assim, verifica-se se as flags referidas estão presentes para contabilizar a abertura de um novo ficheiro utilizando o seguinte fragmento de código.

```
syscall::openat:return
/(self->flags & 1) && (self->flags & 256) && ( self->flags &
↪  512)/
{
    @Create[pid, execname] = count();
}
```

Caso contrário, assumimos que está a ser aberto um ficheiro existente.

```
syscall::openat:return
/ !((self->flags & 1) && (self->flags & 256) && (self->flags &
↪  512))/
{
    @Open[pid, execname] = count();
}
```

Quando ocorre algum erro numa chamada ao sistema, a variável `errno` é definida para um valor diferente de 0. Com esta variável, podemos determinar que uma chamada ao sistema foi executada com sucesso se o valor de `errno` for igual a 0.

```
syscall::openat:return
/ errno == 0/
{
    @Success[pid, execname] = count();
}
```

Utilizando o provider `profile`, é subtraído o valor da variável `seconds` por 1 em cada segundo.

```
profile:::tick-1sec
/seconds != 0/
{
    seconds = seconds -1 ;
}
```

Quando o valor da variável `seconds` for igual a 0, é impressa a data e hora e as estatísticas pretendidas por processo. A variável `seconds` volta a ser igualada ao valor do período.

```
profile:::tick-1sec
/seconds == 0/
{
    seconds = period;
    system("date");
    printf("%Y\n",walltimestamp);
    printf("Tentativas de criação de um ficheiro:\n");
    printa(@Create);
    trunc(@Create);
    printf("Tentativas de abrir um ficheiro existente:\n");
    printa(@Open);
    trunc(@Open);
    printf("Sucessos:\n");
    printa(@Success);
    trunc(@Success);
}
```

O script completo apresenta-se no apêndice B.

De seguida apresentam-se os resultados obtidos com um período de 2 segundos.

```
-----------------------------------------
2020 Jun  4 12:16:18
Tentativas de criação de um ficheiro:

Tentativas de abrir um ficheiro existente:

     2804  dtrace                                                    2
     1210  vmtoolsd                                                 337
Sucessos:

     2804  dtrace                                                    2
     1210  vmtoolsd                                                 331
-----------------------------------------
2020 Jun  4 12:16:20
Tentativas de criação de um ficheiro:

        1  init                                                      2
Tentativas de abrir um ficheiro existente:

     2776  sshd                                                      1
        1  init                                                      2
     1210  vmtoolsd                                                  6
Sucessos:

     2776  sshd                                                      1
     1210  vmtoolsd                                                  2
        1  init                                                      4
```

Figura 4: Resultados obtidos com um período de 2 segundos

4 Exercício 3

No exercício 3 procura-se replicar o comportamento do programa `strace -c`. O script produzido contabiliza por cada tipo de chamada ao sistema, para a execução de um dado programa, o número de chamadas ao sistema, o número de erros ocorridos, o tempo gasto nestas chamadas, a média de tempo gasto por chamada e a percentagem de tempo gasto em cada chamada.

Para a resolução deste exercício foram criadas 3 variáveis globais. A variável `globalTime` conta o tempo despendido total em chamadas ao sistema. Para contar o número total de erros nestas chamadas, a variável `globalErrors` é incrementada sempre que o valor de retorno de uma chamada seja menor que 0, assumindo-se que ocorre um erro sempre que uma chamada devolva um valor negativo. A variável `calls` contabiliza o número total de chamadas ao sistema.

Para contar o tempo despendido em cada tipo de chamada ao sistema, utiliza-se a função `sum()` nas agregações `percI` e `percD` com a parte inteira e decimal da percentagem de tempo gasto, respetivamente, e as variáveis `timeI` e `timeD` com o tempo em segundos. Para estas agregações foi necessário declarar uma parte inteira e outra decimal para, no fim, imprimir resultados com

2 casas decimais visto que o `DTrace` apenas permite operações aritméticas em valores inteiros. A agregação `calls` é incrementada sempre que uma chamada ao sistema acabe de ser executada. Por fim, a agregação `avgTime` utiliza a função `avg()` para contabilizar o tempo médio (em microsegundos) necessário para completar cada chamada ao sistema.

O script criado encontra-se no apêndice C.

Na figura 5 apresenta-se o resultado obtido quando o script é executado com o comando `ps`.

```
a75362@solaris:~/TP2$ dtrace -s tp2.3 -c ps
  PID TTY          TIME CMD
18847 pts/3        0:03 dtrace
18848 pts/3        0:00 ps
17877 pts/3        0:00 bash

% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
  0.00    0.000000           0         1         0 rexit
  0.05    0.000003           3         1         0 getuid
  0.06    0.000003           3         1         0 getpid
  0.07    0.000004           4         1         0 lwp_private
  0.08    0.000004           4         1         0 lseek
  0.08    0.000004           4         1         0 sigpending
  0.09    0.000005           5         1         0 memcntl
  0.13    0.000007           3         2         0 fcntl
  0.13    0.000007           3         2         0 sysconfig
  0.15    0.000008           8         1         0 getrlimit
  0.20    0.000011          11         1         0 resolvepath
  0.24    0.000013           6         2         0 setcontext
  0.50    0.000028           9         3         0 brk
  0.56    0.000031           6         5         0 modctl
  0.61    0.000034          34         1         0 mmap
  0.94    0.000052          26         2         0 ioctl
  1.73    0.000096          96         1         0 mmapobj
  1.81    0.000101          25         4         0 write
  2.30    0.000129          25         5         1 fstatat
 10.69    0.000598         299         2         0 getdents
 17.21    0.000964           8       114         0 close
 29.54    0.001654          14       117         3 openat
 32.78    0.001836          16       112         0 read
------ ----------- ----------- --------- --------- ----------------
100.00    0.005603                   380         4 total
```

Figura 5: Resultados obtidos com o comando `ps`

5 Conclusão

No desenvolvimento deste trabalho foram desenvolvidas algumas competências na utilização da ferramenta *DTrace* para analisar processos específicos ou o sistema em geral. Especificamente, foram criados scripts para resover problemas

de análise às chamadas de sistema realizadas por processos, explorando várias funcionalidades básicas do *DTrace*.

A Exercício 1

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

BEGIN
{
        printf(" Executavel |  PID  |  UID  |  GID  | Ficheiro a
        ↪  abrir             | Ret | Flags\n");
}

syscall::openat:entry
{
        self->file = arg1;
        self->flags = arg2;
}

syscall::openat:return
/self->file != 0 && substr(copyinstr(self->file), 0, 5) ==
↪  "/etc/"/
{
        self->openMode = (!(self->flags & 3)) ? "O_RDONLY" : "";
        self->openMode = (self->flags & 1) ? "O_WRONLY" :
        ↪  self->openMode;
        self->openMode = (self->flags & 2) ? "O_RDWR" :
        ↪  self->openMode;
        self->ap = (self->flags & 8) ? " | O_APPEND" : "";
        self->cr = (self->flags & 256) ? " | O_CREAT" : "";

        printf(" %-11.11s  %5d   %5d   %5d   %-30.30s %3d
        ↪  %s%s%s\n", execname,pid,uid,gid,
        ↪  copyinstr(self->file),arg0, self->openMode,
        ↪  self->ap,self->cr);
        self->openMode = "";
        self->ap = "";
        self->cr = "";
}

syscall::openat:return
{
        self->file = 0;
        self->flags = 0;
}
```

B Exercício 2

```
#!/usr/sbin/dtrace -qs

#pragma D option quiet
#pragma D option destructive
#pragma D option defaultargs

BEGIN
{
    period = ($1 > 0) ? $1 : 5;
    seconds = period;
}

syscall::openat:entry
{
    self->execname = execname;
    self->pid = pid;
    self->file = arg1;
    self->flags = arg2;
}

syscall::openat:return
/ errno == 0/
{
    @Success[pid, execname] = count();
}

syscall::openat:return
/(self->flags & 1) && (self->flags & 256) && ( self->flags &
↪  512)/
{
    @Create[pid, execname] = count();
}

syscall::openat:return
/ !((self->flags & 1) && (self->flags & 256) && (self->flags &
↪  512))/
{
    @Open[pid, execname] = count();
}

syscall::openat:return
{
    self->execname = 0;
    self->pid = 0;
```

```
    self->file = 0;
    self->flags = 0;
}

profile:::tick-1sec
/seconds != 0/
{
    seconds = seconds -1 ;
}

profile:::tick-1sec
/seconds == 0/
{
    seconds = period;
    printf("-----------------------------------------\n");
    printf("%Y\n",walltimestamp);
    printf("Tentativas de criação de um ficheiro:\n");
    printa(@Create);
    trunc(@Create);
    printf("Tentativas de abrir um ficheiro existente:\n");
    printa(@Open);
    trunc(@Open);
    printf("Sucessos:\n");
    printa(@Success);
    trunc(@Success);
}
```

C Exercício 3

```
#!/usr/sbin/dtrace -qs
#pragma D option quiet
BEGIN
{
    globalTime = 0;
}

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

syscall:::return
/pid == $target && errno != 0/
```

```
{
    globalErrors++;
    @errors[probefunc] = count();
}

syscall:::return
/pid == $target/
{
    self->time = timestamp - self->time;
    @timeI[probefunc] = sum(self->time);    /* Parte inteira */
    @timeD[probefunc] = sum((self->time));  /* Parte decimal */
    @avgTime[probefunc] = avg(self->time);
    @percD[probefunc] = sum(self->time);
    @percI[probefunc] = sum(self->time);
    globalTime += self->time;
    self->time = 0;
    calls++;
}

END
{
    printf("\n%% time     seconds  usecs/call     calls    errors
    ↪  syscall\n------ ----------- ----------- ---------
    ↪  --------- ----------------\n");
    normalize(@timeI,1000000000);
    normalize(@percI,globalTime/100);
    normalize(@percD,globalTime/10000);
    normalize(@timeD,1000);
    normalize(@avgTime,1000);
    printa("%@06d\033[6D%@3d.\033[2C %@011d\033[11D%@4d.\033[6C
    ↪  %@11d %@9d %@9.0d %s\n",
    @percD,
    @percI,
    @timeD,
    @timeI,
    @avgTime,
    @calls,
    @errors);
    printf("------ ----------- ----------- --------- ---------
    ↪  ----------------\n");


    printf("100.00 %4d.%06d             %9d %9d
    ↪  total",globalTime/1000000000,globalTime/1000%1000000,calls,globalErrors);
}
```