

Performance e Precisão Relativas de Filesystem Benchmarks

Active Benchmarking da ferramenta iotrace recorrendo a DTrace em Solaris 11

Filipe Oliveira Departamento de Informática
Universidade do Minho
Email: a57816@alunos.uminho.pt

17 de Abril de 2016

Introdução – Contextualização da Ferramenta DTrace

A necessidade de recurso a ferramentas de traçado dinâmico como o DTrace está implicitamente associada à necessidade de recolha de informação dos sistemas de computação no seu todo – via agregação, ou de processos/kernels específicos. Com o aumento da complexidade dos sistemas existem “comportamentos” incorretos de kernels que apenas podem ser observados através da instrumentação dos kernels no próprio sistema e recolha de dados estatísticos da mesma. Ora, essa instrumentação pode ser via amostragem (“sampling”) ou via traçado dinâmico (“tracing”).

A grande vantagem do recurso à ferramenta DTrace está associada à segunda forma de instrumentação (apesar de a ferramenta também nos permitir a recolha de dados de amostragem). O DTrace consegue acoplar informação extremamente distinta, contando ainda com a mais valia de um overhead mínimo na grande maioria das medições dos valores de informação, via “instrumentation points” ou “probes”.

No sistema em estudo contamos com 94274 probes (organizadas por provider:module:function:name). De seguida apresenta-se a lista de providers disponíveis no mesmo, conjuntamente com um excerto da sua contextualização extraído do livro SPEC¹ e do guia online do Dtrace² :

- **cpc** : CPU performance counters
- **dtrace** provides several probes related to DTrace itself
- **fbt** : kernel-level dynamic tracing
- **fsinfo** allows tracing of file system events across different file system types, with file information for each event
- **io** : block device interface tracing (disk I/O)
- **ip** : IP protocol events: send and receive
- **iscsi** : iSCSI protocol events: connections, send and receive
- **lockstat** : lock contention statistics, or to understand virtually any aspect of locking behavior
- **mib** : makes available probes that correspond to counters in the illumos management information bases (MIBs)
- **proc** : process-level events: create, exec, exit
- **profile** : provides probes associated with a time-based interrupt firing every fixed, specified time interval
- **sched** : kernel scheduling events

¹Systems performance : enterprise and the cloud / Brendan Gregg.

²<http://dtrace.org/guide/preface.html>

- **sdt** : creates probes at sites that a software programmer has formally designated
- **shadowfs** ZFS Shadow Migration events
- **syscall** : system call trap table
- **sysevent** : system events
- **sysinfo** : system statistics
- **tcp** : TCP protocol events: connections, send and receive
- **udp** : UDP protocol events: connections, send and receive
- **vm** : virtual memory statistics
- **vminfo** : virtual memory statistics

É importante compreender o anteriormente enumerado para associar corretamente a informação que queremos recolher dos kernels/sistemas às probes disponibilizadas. No seguimento do discutido durante as componentes práticas da UCE de Engenharia de Sistemas de Computação foi então proposta a realização de alguns exercícios de ambientação com a ferramenta de medição DTrace, que passarei de seguida a solucionar.

1

Fazer o traçado das chamadas ao sistema `open()` que deverá imprimir a seguinte informação por linha:

- nome do ficheiro executável e respetivos: PID do processo, UID do utilizador e GID do grupo.
- Caminho absoluto para o ficheiro que for aberto.
- A cadeia de caracteres com as “flags” da chamada ao sistema `open()`, `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_APPEND`, `O_CREAT`
- O Valor de retorno de chamada de sistema

Testar o programa com as hipóteses que seguem:

- `cat /etc/inittab > /tmp/test`
- `cat /etc/inittab >> /tmp/test`
- `cat /etc/inittab | tee /tmp/test`
- `cat /etc/inittab | tee -a /tmp/test`

Opcional: Modificar o programa para que apenas os ficheiros com “/etc” no caminho sejam detetados.

Tal como referido no enunciado, em Solaris 11 a chamada de sistema `open()` foi substituído por outra mais genérico `openat()`. Analisando a assinatura da função `openat()`³:

```
int openat(int fildes, const char *path, int oflag, /* mode_t mode */);
```

necessitamos então de associar a informação que necessitamos a probes disponíveis no sistema através do comando:

```
a57816@solaris11:/share/jade/a57816/ESC_DTRACE$ dtrace -l -f openat*
ID PROVIDER MODULE FUNCTION NAME
1583 syscall openat entry
1584 syscall openat return
1585 syscall openat64 entry
1586 syscall openat64 return
30132 fbt genunix openat32 entry
30133 fbt genunix openat32 return
30134 fbt genunix openat64 entry
30135 fbt genunix openat64 return
30649 fbt genunix openat entry
30650 fbt genunix openat return
a57816@solaris11:/share/jade/a57816/ESC_DTRACE$
```

Desta forma confirmamos a existência das probes necessárias para a correcta realização do enunciado, sedo estas as de id 1583 a 1586 no nosso sistema de computação, escolha que iremos justificar de seguida. Ora, os 4 primeiros campos requeridos (nome do ficheiro executável e respetivos: PID do processo, UID do utilizador e GID do grupo) podem ser obtidos através das variáveis acessíveis na ferramenta DTrace **execname**, **pid**, **uid**, **gid**, disponíveis em qualquer das probes seleccionadas anteriormente. Contudo, o caminho absoluto para o ficheiro que for aberto pode apenas ser acedido através da variável **arg1** nas probes **syscall::openat:entry** e **syscall::openat64:entry**, sendo ressalvada ainda a necessidade de copiar a string que contém o caminho absoluto da zona de memória do utilizador para o kernel. Das premissas anteriores sabemos que precisaremos da seguinte regra no nosso script `exec1.d`:

```
syscall::openat*:entry {
    self->pathname = copyinstr(arg1);
    ....
    ....
    ....
}
```

Por forma a obtermos a cadeia de caracteres com as “flags” da chamada ao sistema `open()`, `O_RDONLY`, `O_WRONLY`, `O_RDWR`, `O_APPEND`, `O_CREAT`, necessitamos de aceder aos valores do terceiro argumento das system calls `openat()` e `openat64()`. Ora, tal informação será

³https://docs.oracle.com/cd/E26502_01/html/E28556/gkxro.html

accedida também através das duas probes anteriormente descritas na variável `arg2` (terceira variável das funções), sendo necessário proceder à correcta interpretação e posterior impressão dos seus valores. A solução encontrada passa por guardar a cadeia de caracteres numa variável local `self->flags = arg2` e proceder posteriormente à impressão.

Relativamente à impressão das flags podemos dividir a mesma em duas grandes porções. Temos de uma lado o modo de abertura relativos às permissões (`O_RDONLY`, `O_WRONLY`, `O_RDWR`) e as restantes duas flags do nosso interesse (`O_APPEND`, `O_CREAT`). Relativamente ao modo de abertura podemos considerar que temos apenas de realizar duas verificações, tendo em conta que os condicionais em DTrace se expressam pelo operador ternário. Assim, resolvemos a primeiro porção do problema com a seguinte expressão

```
/*if its not O_WRONLY or O_RDWR then its implicitly O_RDONLY */
printf( "%-9s" , self->flags & O_WRONLY ? " O_WRONLY " : self->flags & ←
O_RDWR ? " O_RDWR " : " O_RDONLY " );
```

Uma vez que para o ficheiro ser aberto em modo `O_WRONLY` o & lógico entre `self->flags` e `O_WRONLY` terá que retornar o valor `O_WRONLY`. O mesmo se aplica para a flag de abertura `O_RDWR`. Caso a cadeia de caracteres não corresponder positivamente a nenhuma das anteriores verificações, implicará obrigatoriamente a abertura do ficheiro em modo `O_RDONLY` (modo de leitura).

Relativamente aos modos `O_APPEND` e `O_CREAT` as verificações são feitas separadamente pelas seguintes expressões:

```
printf( "%-9s" , self->flags & O_APPEND ? "| O_APPEND " : " " );
printf( "%-9s" , self->flags & O_CREAT ? "| O_CREAT " : " " );
```

Contudo, o valor de retorno de chamada de sistema não pode ser accedido através das duas probes anteriores, dado que, no início da system call, ainda não existe a informação relativa ao sucesso ou insucesso da mesma e respectivo descritor de ficheiro em caso de sucesso. Assim, necessitamos de acrescentar uma outra regra ao nosso ficheiro `exec1.d` que, aquando do término das system calls `openat()` e `openat64()`, verifica o valor retornado na variável `filides`, acessível através da variável DTrace `arg1`.

Adicionando apenas uma regra para aquando do início da execução do nosso script imprimir o cabeçalho ficamos com o seguinte ficheiro final:

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

/*
*****
* Copyright(C) 2016 Filipe Oliveira
* HPC Group, Computer Science Dpt.
* University of Minho
* All Rights Reserved.
*****
* Content : simple openat and openat64 system calls tracer
*
*****/

#pragma D option quiet

dtrace::BEGIN {
    printf("%-10s%-8s%-8s%-8s%-50s%-27s%-5s\n", "EXEC" , "PID" , "UID" , "GID" , "ABS PATH←
" , "FLAGS" , "RETURNED VALUE");
}

/* will catch openat and openat64 */
syscall::openat::entry {
    self->pathname = copyinstr(arg1);
    self->flags = arg2;
}

/* will catch openat and openat64 */
syscall::openat::return
{
    printf("%-10s%-8d%-8d%-8d%-50s", execname, pid, uid, gid, self->pathname);
    /*if its not O_WRONLY or O_RDWR then its implicitly O_RDONLY */
    printf( "%-9s" , self->flags & O_WRONLY ? " O_WRONLY " : self->flags & O_RDWR ? " ←
O_RDWR " : " O_RDONLY " );
    printf( "%-9s" , self->flags & O_APPEND ? "| O_APPEND " : " " );
    printf( "%-9s" , self->flags & O_CREAT ? "| O_CREAT " : " " );
    printf("%5i\n", arg1);
}
```

1.1 Resultados de execução dos comandos exemplo

Podemos agora executar os 4 comandos requeridos, apresentando de seguida os resultados de execução:

1.1.1

```
cat /etc/inittab > /tmp/test1
```

EXEC	PID	UID	GID RETURNED	ABS PATH VALUE	FLAGS ↔
bash	22658	29220	5000	/tmp/test1	O_WRONLY ↔
cat	22658	29220	5000	/var/ld/ld.config	O_RDONLY ↔
cat	22658	29220	5000	/lib/libc.so.1	O_RDONLY ↔
cat	22658	29220	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY ↔
cat	22658	29220	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY ↔
cat	22658	29220	5000	/etc/inittab	O_RDONLY ↔

1.1.2

```
cat /etc/inittab >> /tmp/test1
```

Atente na primeira linha retornada, na flag O_APPEND tal como previsível:

EXEC	PID	UID	GID RETURNED	ABS PATH VALUE	FLAGS ↔
bash	22662	29220	5000	/tmp/test1	O_WRONLY O_APPEND ↔
cat	22662	29220	5000	/var/ld/ld.config	O_RDONLY ↔
cat	22662	29220	5000	/lib/libc.so.1	O_RDONLY ↔
cat	22662	29220	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY ↔
cat	22662	29220	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY ↔
cat	22662	29220	5000	/etc/inittab	O_RDONLY ↔

1.1.3

```
cat /etc/inittab | tee /tmp/test1
```

EXEC	PID	UID	GID RETURNED	ABS PATH VALUE	FLAGS ↔
tee	22666	29220	5000	/var/ld/ld.config	O_RDONLY ↔
tee	22666	29220	5000	/lib/libc.so.1	O_RDONLY ↔
tee	22666	29220	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY ↔
tee	22666	29220	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY ↔
tee	22666	29220	5000	/tmp/test1	O_WRONLY ↔
cat	22665	29220	5000	/var/ld/ld.config	O_RDONLY ↔
cat	22665	29220	5000	/lib/libc.so.1	O_RDONLY ↔
cat	22665	29220	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY ↔
cat	22665	29220	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY ↔
cat	22665	29220	5000	/etc/inittab	O_RDONLY ↔

1.1.4

```
cat /etc/inittab | tee -a /tmp/test1
```

EXEC	PID	UID	GID	ABS PATH	RETURNED VALUE	FLAGS	←
tee	22656	29220	5000	/var/ld/ld.config	-1	O_RDONLY	←
tee	22656	29220	5000	/lib/libc.so.1	3	O_RDONLY	←
tee	22656	29220	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	3	O_RDONLY	←
tee	22656	29220	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	3	O_RDONLY	←
tee	22656	29220	5000	/tmp/test1	3	O_WRONLY	←
cat	22655	29220	5000	/var/ld/ld.config	-1	O_RDONLY	←
cat	22655	29220	5000	/lib/libc.so.1	3	O_RDONLY	←
cat	22655	29220	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	3	O_RDONLY	←
cat	22655	29220	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	3	O_RDONLY	←
cat	22655	29220	5000	/etc/inittab	3	O_RDONLY	←

1.2 Resolução do exercício opcional

Para modificar o programa para que apenas os ficheiros com `"/etc"` no caminho sejam detetados, necessitamos apenas de adicionar o predicado `/strstr(self->pathname, "/etc") != NULL/` às duas probes detectadas por `syscall::openat*:return`, produzindo o ficheiro `ex1.opt.d`:

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

/*
*****
* Copyright(C) 2016 Filipe Oliveira
* HPC Group, Computer Science Dpt.
* University of Minho
* All Rights Reserved.
*****
* Content : simple openat and openat64 system calls tracer
* with /etc/ on its pathname
*
*****/

#pragma D option quiet

dtrace::BEGIN {
    printf("%-10s%-8s%-8s%-8s%-30s%-27s%-5s\n", "EXEC", "PID", "UID", "GID", "ABS PATH", "RETURNED VALUE", "FLAGS");
}

/* will catch openat and openat64 */
syscall::openat*:entry
{
    self->pathname = copyinstr(arg1);
    self->flags = arg2;
}

/* will catch openat and openat64 */
syscall::openat*:return
{
    /strstr(self->pathname, "/etc") != NULL/
    {
        printf("%-10s%-8d%-8d%-8d%-30s", execname, pid, uid, gid, self->pathname);
        /*if its not O_WRONLY or ORDWR then its implicitly O_RDONLY */
        printf(" %-9s", self->flags & O_WRONLY ? " O_WRONLY " : self->flags & O_RDWR ? " O_RDWR " : " O_RDONLY ");
        printf(" %-9s", self->flags & O_APPEND ? "| O_APPEND " : " ");
        printf(" %-9s", self->flags & O_CREAT ? "| O_CREAT " : " ");
        printf("%5i\n", arg1);
    }
}
```

Com o seguinte exemplo de retorno de execução:

EXEC	PID	UID	GID	ABS PATH	FLAGS	RETURNED	←
cat	22676	29220	5000	/etc/acct	O_RDONLY	3	
cat	22676	29220	5000	/etc/aliases	O_RDONLY	3	
cat	22676	29220	5000	/etc/amd64	O_RDONLY	3	
cat	22676	29220	5000	/etc/anthy	O_RDONLY	3	
cat	22676	29220	5000	/etc/apache2	O_RDONLY	3	
cat	22676	29220	5000	/etc/auto_home	O_RDONLY	3	
cat	22676	29220	5000	/etc/auto_master	O_RDONLY	3	
cat	22676	29220	5000	/etc/auto_share	O_RDONLY	3	
cat	22676	29220	5000	/etc/avahi	O_RDONLY	3	
cat	22676	29220	5000	/etc/bash	O_RDONLY	3	

cat	22676	29220	5000	/etc/bonobo-activation	0_RDONLY	3	12
cat	22676	29220	5000	/etc/brand	0_RDONLY	3	13
cat	22676	29220	5000	/etc/brltty	0_RDONLY	3	14
cat	22676	29220	5000	/etc/certs	0_RDONLY	3	15
cat	22676	29220	5000	/etc/compizconfig	0_RDONLY	3	16
cat	22676	29220	5000	/etc/ConsoleKit	0_RDONLY	3	17
cat	22676	29220	5000	/etc/cron.d	0_RDONLY	3	18
cat	22676	29220	5000	/etc/crypto	0_RDONLY	3	19
cat	22676	29220	5000	/etc/cups	0_RDONLY	3	20
cat	22676	29220	5000	/etc/dacf.conf	0_RDONLY	3	21
cat	22676	29220	5000	/etc/dat	0_RDONLY	3	22
cat	22676	29220	5000	/etc/datmsk	0_RDONLY	3	23
cat	22676	29220	5000	/etc/dbus-1	0_RDONLY	3	24
cat	22676	29220	5000	/etc/default	0_RDONLY	3	25
cat	22676	29220	5000	/etc/defaultrouter	0_RDONLY	3	26
cat	22676	29220	5000	/etc/dev	0_RDONLY	3	27
cat	22676	29220	5000	/etc/devices	0_RDONLY	3	28
cat	22676	29220	5000	/etc/devlink.tab	0_RDONLY	3	29
cat	22676	29220	5000	/etc/dfs	0_RDONLY	3	30
cat	22676	29220	5000	/etc/dhcp	0_RDONLY	3	31
cat	22676	29220	5000	/etc/dladm	0_RDONLY	3	32
cat	22676	29220	5000	/etc/drirc	0_RDONLY	3	33
cat	22676	29220	5000	/etc/driver	0_RDONLY	3	34
cat	22676	29220	5000	/etc/driver_aliases	0_RDONLY	3	35
cat	22676	29220	5000	/etc/driver_classes	0_RDONLY	3	36
cat	22676	29220	5000	/etc/dumpadm.conf	0_RDONLY	3	37
cat	22676	29220	5000	/etc/dumpdates	0_RDONLY	3	38
cat	22676	29220	5000	/etc/emulexDiscConfig	0_RDONLY	3	39
cat	22676	29220	5000	/etc/emulexRMConfig	0_RDONLY	3	40
cat	22676	29220	5000	/etc/emulexRMOptions	0_RDONLY	3	41
cat	22676	29220	5000	/etc/flash	0_RDONLY	3	42
cat	22676	29220	5000	/etc/fm	0_RDONLY	3	43
cat	22676	29220	5000	/etc/fonts	0_RDONLY	3	44
cat	22676	29220	5000	/etc/foomatic	0_RDONLY	3	45
cat	22676	29220	5000	/etc/format.dat	0_RDONLY	3	46
cat	22676	29220	5000	/etc/fs	0_RDONLY	3	47
cat	22676	29220	5000	/etc/ftpd	0_RDONLY	3	48
cat	22676	29220	5000	/etc/ftpusers	0_RDONLY	3	49
cat	22676	29220	5000	/etc/gconf	0_RDONLY	3	50
cat	22676	29220	5000	/etc/gdm	0_RDONLY	3	51
cat	22676	29220	5000	/etc/gnome-vfs-2.0	0_RDONLY	3	52
cat	22676	29220	5000	/etc/gnome-vfs-mime-magic	0_RDONLY	3	53
cat	22676	29220	5000	/etc/gnu	0_RDONLY	3	54
cat	22676	29220	5000	/etc/group	0_RDONLY	3	55
cat	22676	29220	5000	/etc/gss	0_RDONLY	3	56
cat	22676	29220	5000	/etc/gtk-2.0	0_RDONLY	3	57
cat	22676	29220	5000	/etc/hal	0_RDONLY	3	58
cat	22676	29220	5000	/etc/hba.conf	0_RDONLY	3	59
cat	22676	29220	5000	/etc/hostid	0_RDONLY	3	60
cat	22676	29220	5000	/etc/hosts	0_RDONLY	3	61
cat	22676	29220	5000	/etc/hp	0_RDONLY	3	62
cat	22676	29220	5000	/etc/ibadm	0_RDONLY	3	63
cat	22676	29220	5000	/etc/ima.conf	0_RDONLY	3	64
cat	22676	29220	5000	/etc/inet	0_RDONLY	3	65
cat	22676	29220	5000	/etc/inetd.conf	0_RDONLY	3	66

2

Mostrar para os processos que estão a correr no sistema as seguintes estatísticas, com valores obtidos durante cada iteração: a)

- número de tentativas de abrir ficheiros existentes;
- número de tentativas para criar ficheiros;
- número de tentativas bem-sucedidas.

b) Repetidamente, com um período (especificado em segundos) passado como argumentos da linha de comandos deve imprimir:

- hora e dia atual em formato legível.
- as estatísticas recolhidas por PID e respetivo o nome.

2.1

Utilizando o script desenvolvido no exercício anterior, será necessário adicionar variáveis de agregação e contar cada tipo de abertura de ficheiro. Ora, por “**número de tentativas de abrir ficheiros existentes**” podemos considerar o número de aberturas de ficheiro sem a flag `O_CREAT`, o que se traduz no predicado `/(arg2&O_CREAT) == 0/`. Analogamente “**número de tentativas para criar ficheiros**” será traduzido no predicado `/(arg2&O_CREAT) == O_CREAT/`, ambos predicados a serem incluídos para as 2 probes descritas por `syscall::openat*:entry`.

Resta-nos traduzir “**número de tentativas bem-sucedidas**”. Ora, analisando a descrição da função `openat()` percebemos que apenas quando o descritor de ficheiro toma o valor -1 poderemos considerar que não foi bem sucedida a operação de abertura, que se traduz no predicado `//arg1 >= 0/` para as 2 probes descritas por `syscall::openat*:return`.

Definidas as iterações que estão ou não incluídas em cada regra resta-nos criar as variáveis que agregam a informação sendo estas:

```
@successfull[ pid ];
@open_request[ pid ];
@create_request[ pid ];
```

a serem alteradas a cada iteração (via `count()`) nas regras especificadas no ficheiro completo `ex2a.d`. Denote também que no início e fim do script são impressos o cabeçalho e as variáveis que agregam a informação por `pid`:

```
#!/usr/sbin/dtrace -s
/*
*****
* Copyright(C) 2016 Filipe Oliveira
* HPC Group, Computer Science Dpt.
* University of Minho
* All Rights Reserved.
*****
* Content : simple openat and openat64 system calls tracer and agregator
* by
* by pid and opening mode
*
*****
*/
#pragma D option quiet
dtrace::BEGIN {
printf("*****\n");
```



```

printf("openat and openat64 syscalls aggregator\n");
printf("!=O_CREAT :: opening files already in system\n");
printf("  O_CREAT :: opening files with flag to create\n");
printf("    #SUCC :: sucessfull openat and openat64 system calls\n");
printf("*****\n");
}

/* will catch openat and openat64 number of tries to create file */
syscall::openat::entry
/( arg2 & O_CREAT ) == 0 /
{
    @open_request[ pid ] = count();
}

/* will catch openat and openat64 number of tries to create file */
syscall::openat::entry
/ ( arg2 & O_CREAT ) == O_CREAT /
{
    @create_request[ pid ] = count();
}

/* will catch openat and openat64 sucessfull file open
 * from linux man: " On success, openat() returns a new file descriptor.
 * On error, -1 is returned and errno is set to indicate the error." */
syscall::openat::return
/arg1 >= 0/
{
    @sucessfull[ pid ] = count();
}

dtrace::END {
    printf("*****\n");
    printf(" %-6s\t%10s\t%10s\t%10s\n", "pid", "!=O_CREAT", "O_CREAT", "#SUCC" );
    printf("\n");
    printa( "%6d\t%10d\t%10d\t%10d\n", @open_request, @create_request, @sucessfull );
    clear(@open_request);
    clear(@create_request);
    clear(@sucessfull);
}

```

Atente no exemplo do output do script:

```

a57816@solaris11:/share/jade/a57816/ESC_DTRACE$ ./ex2a.d
*****
openat and openat64 syscalls aggregator
!=O_CREAT :: opening files already in system
  O_CREAT :: opening files with flag to create
    #SUCC :: sucessfull openat and openat64 system calls
*****
^C
*****
pid      !=O_CREAT      O_CREAT      #SUCC
*****
1351      1      0      1
1447      3      0      3
22701     4      0      3
22487     5      0      5
22699     7      0      4
22700    11      0      8
22698    196     0     190

```

2.2

Relativamente às estatísticas agregadas e impressas repetidamente, com um período (especificado em segundos) passado como argumentos da linha de comandos, podemos recorrer à variável **wall-timestamp** por forma a obter o valor da hora e dia atual em formato legível. Para obtermos o valor em segundos da linha de comandos resta-nos verificar o valor da variável **\$1** e recorrer a **tick-\$1s** para imprimir a um ritmo de \$1 segundos.

Por forma a imprimirmos apenas os valores agregados entre duas medições necessitamos de limpar os valores presentes nas variáveis agregadas. Tal é realizado recorrendo ao método **trunc**:

```
trunc(@open_request);
trunc(@create_request);
trunc(@successfull);
```

1
2
3

Dado considerar que esta alínea do exercício ser um complemento do primeiro requisito foram adicionadas as variáveis:

```
@all_successfull[ pid, execname ];
@all_open_request[ pid, execname ];
@all_create_request[ pid, execname ];
```

1
2
3

que mantêm a possibilidade do utilizador visualizar no término da script os valores completos agregados. Assim, o ficheiro **ex2b.d** apresenta o seguinte formato:

```
#!/usr/sbin/dtrace -s
/*
*****
*   Copyright(C) 2016 Filipe Oliveira
*   HPC Group, Computer Science Dpt.
*   University of Minho
*   All Rights Reserved.
*****
*   Content : simple openat and openat64 system calls tracer and agregator
*             by
*             by pid and opening mode at a constant time rate passed by
*             argument
*
*****
*/
#pragma D option quiet
dtrace::BEGIN {
    printf("
*****
n");
    printf("openat and openat64 syscalls aggregator by constant time rate
passed by argument\n");
    printf("\n TIME RATE %d seconds \n", $1);
    printf("START TIME %Y \n\n", walltimestamp);
    printf("!=O_CREAT :: opening files already in system\n");
    printf(" O_CREAT :: opening files with flag to create\n");
    printf(" #SUCC :: successfull openat and openat64 system calls\n");
    printf("*****
* * * * * \n");
    printf(" %-6s\t%-20s\t%10s\t%10s\t%10s\n", "pid", "execname", "!=O_CREAT",
, "O_CREAT", "#SUCC" );
    printf("*****
* * * * * \n");
}
}
```

1
2
3
4* ←
5
6
7
8
9* ←
10
11
12
13* ←
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30

```

/* will catch openat and openat64 number of tries to create file */
syscall::openat*:entry
/( arg2 & O_CREAT ) == 0 /
{
    @open_request[ pid, execname ] = count();
    @all_open_request[ pid, execname ] = count();
}

/* will catch openat and openat64 number of tries to create file */
syscall::openat*:entry
/ ( arg2 & O_CREAT ) == O_CREAT /
{
    @create_request[ pid, execname ] = count();
    @all_create_request[ pid, execname ] = count();
}

/* will catch openat and openat64 sucessfull file open
 * from linux man: " On success, openat() returns a new file descriptor.
 * On error, -1 is returned and errno is set to indicate the error." */
syscall::openat*:return
/arg1 >= 0/
{
    @successfull[ pid, execname ] = count();
    @all_successfull[ pid, execname ] = count();
}

tick-$1s {
    printf("\n[ %20Y * * * * * \n", walltimestamp);
    printa( "%6d\t%-20s\t%@10d\t%@10d\t%@10d\n", @open_request, ←
        @create_request, @successfull );
    trunc(@open_request);
    trunc(@create_request);
    trunc(@successfull);
    printf("
        * * * * * \n");
}

dtrace:::END {
    printf("\n***** AGGREGATED RESULTS ←
        *****\n");
    printf("
        %20Y \n\n", walltimestamp);
    printa( "%6d\t%-20s\t%@10d\t%@10d\t%@10d\n", @all_open_request, ←
        @all_create_request, @all_successfull );
    printf("\n←
        *****\n");
    clear(@all_open_request);
    clear(@all_create_request);
    clear(@all_successfull);
    clear(@open_request);
    clear(@create_request);
    clear(@successfull);
}

```

Atente no exemplo do output do script:

```
*****
openat and openat64 syscalls aggregator by constant time rate passed by argument

    TIME RATE 1 seconds
START TIME 2016 Apr 10 03:52:13

!=O_CREAT :: opening files already in system
O_CREAT :: opening files with flag to create
#SUCC :: successfull openat and openat64 system calls
*****
pid      execname      !=O_CREAT      O_CREAT      #SUCC
```

```

* * * * *
[ 2016 Apr 10 03:52:13 * * * * * * * * * *
22713 ex2b.d 2 0 2
* * * * * * * * * * * * * * * * * * * *]
[ 2016 Apr 10 03:52:14 * * * * * * * * * *
259 utmpd 3 1 4
* * * * * * * * * * * * * * * * * * * *]
[ 2016 Apr 10 03:52:15 * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * *]
[ 2016 Apr 10 03:52:16 * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * *]
[ 2016 Apr 10 03:52:17 * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * *]
[ 2016 Apr 10 03:52:18 * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * *]
[ 2016 Apr 10 03:52:19 * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * *]
[ 2016 Apr 10 03:52:20 * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * *]
[ 2016 Apr 10 03:52:21 * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * *]
[ 2016 Apr 10 03:52:22 * * * * * * * * * *
22487 bash 1 0 1
22714 cat 35 0 30
* * * * * * * * * * * * * * * * * * * *]
[ 2016 Apr 10 03:52:23 * * * * * * * * * *
* * * * * * * * * * * * * * * * * * * *]
^C
***** AGGREGATED RESULTS *****
2016 Apr 10 03:52:24
22487 bash 1 0 1
22713 ex2b.d 2 0 2
259 utmpd 3 1 4
22714 cat 35 0 30
*****

```

3 Conclusão

Tal como mencionado no início do presente caso de estudo a ferramenta DTrace mostra-se bastante útil e única em termos de funcionalidades quando necessitamos de agregar informação de vários processos/threads,etc. Ou seja, no contexto da computação paralela será extremamente interessante recorrer a esta ferramenta de traçado dinâmico na execução de algoritmos paralelos.

Este foi apenas um trabalho introdutório mas permitiu demonstrar a capacidade de recolher e ao mesmo tempo tratar dados de todo um sistema extremamente complexo e vasto com apenas uma ferramenta. O caso de estudo ultrapassa portanto os resultados obtidos pelas scripts geradas, prendendo-se uma vez mais com o desenvolvimento de capacidade prática no uso da ferramenta, e envolvimento com métodos de tratamento de grandes volumes de dados, e análise de métricas de sistemas de computação de alta performance.

Retrata sobretudo a capacidade analisar funcionalidades disponibilizadas e a sua correta aplicação na resolução de problemas de computação tendo sempre em conta o mínimo de alteração possível na performance dos kernels/sistemas a analisar.