

Introdução às ferramentas de observação do sistema de computação

Traçado dinâmico recorrendo a DTrace em Solaris 11

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

11 de Junho 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	O_RDONLY ←
tee	22656	29220	5000	/lib/libc.so.1	O_RDONLY ←
tee	22656	29220	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY ←
tee	22656	29220	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY ←
tee	22656	29220	5000	/tmp/test1	O_WRONLY ←
cat	22655	29220	5000	/var/ld/ld.config	O_RDONLY ←
cat	22655	29220	5000	/lib/libc.so.1	O_RDONLY ←
cat	22655	29220	5000	/usr/lib/locale/en_US.UTF-8/en_US.UTF-8.so.3	O_RDONLY ←
cat	22655	29220	5000	/usr/lib/locale/en_US.UTF-8/methods_unicode.so.3	O_RDONLY ←
cat	22655	29220	5000	/etc/inittab	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`:

<code>#!/usr/sbin/dtrace -s</code>	1
<code>/*</code>	2
<code>*****</code>	3
<code>* Copyright(C) 2016 Filipe Oliveira</code>	4
<code>* HPC Group, Computer Science Dpt.</code>	5
<code>* University of Minho</code>	6
<code>* All Rights Reserved.</code>	7
<code>*****</code>	8
<code>* Content : simple openat and openat64 system calls tracer</code>	9
<code>* with /etc/ on its pathname</code>	10
<code>*</code>	11
<code>*****/</code>	12
<code>#pragma D option quiet</code>	13
<code>dtrace::BEGIN {</code>	14
<code>printf("%-10s%-8s%-8s%-30s%-27s%-5s\n", "EXEC", "PID", "UID", "GID", "ABS PATH",</code>	15
<code>"", "FLAGS", "RETURNED VALUE");</code>	16
<code>}</code>	17
<code>/* will catch openat and openat64 */</code>	18
<code>syscall::openat*:entry</code>	19
<code>{</code>	20
<code>self->pathname = copyinstr(arg1);</code>	21
<code>self->flags = arg2;</code>	22
<code>}</code>	23
<code>/* will catch openat and openat64 */</code>	24
<code>syscall::openat*:return</code>	25
<code>/strstr(self->pathname, "/etc") != NULL/</code>	26
<code>{</code>	27
<code>printf("%-10s%-8d%-8d%-8d%-30s", execname, pid, uid, gid, self->pathname);</code>	28
<code>/*if its not O_WRONLY or ORDWR then its implicitly O_RDONLY */</code>	29
<code>printf(" %-9s", self->flags & O_WRONLY ? " O_WRONLY " : self->flags & O_RDWR ? " ←</code>	30
<code>O_RDWR " : " O_RDONLY ");</code>	31
<code>printf(" %-9s", self->flags & O_APPEND ? " O_APPEND " : ");</code>	32
<code>printf(" %-9s", self->flags & O_CREAT ? " O_CREAT " : ");</code>	33
<code>printf("%5i\n", arg1);</code>	34
<code>}</code>	35

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 **walltimestamp** 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
```

[illegible]

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.

A Uma análise à política de escalonamento de zonas paralelas em OpenMP recorrendo à ferramenta Dtrace

A.1 Contextualização

Retomando o trabalho prático 2, e respectiva análise de paralelismo em ambiente de memória partilhada, via zonas paralelas OpenMP será interessante, como extra, analisar a influência dos diversos tipos de escalonamento OpenMP:

- **static** – a maioria dos compiladores dividem o trabalho dos loops em $\frac{N \text{ iteracoes}}{p \text{ threads}}$ por default, sendo o número de iterações distribuído uniformemente por thread OpenMP.

A título ilustrativo, suponha que existem 1000 iterações a serem distribuídas de forma estática por 4 threads OpenMP. O loop será repartido, em caso standard, da seguinte forma:

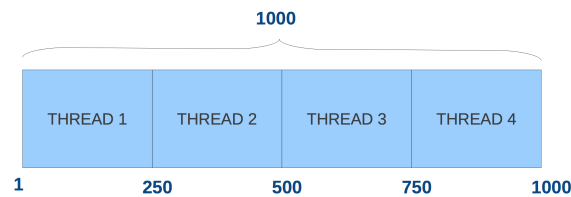


Figura 1: Ilustração do escalonamento estático:

Ora, esta poderá não ser a melhor forma de escalonamento, muito devido à potencial irregularidade de tempos de conclusão de iteração por cada thread. Em caso de trabalhos regulares esta opção representa o menor overhead do paralelismo. Para casos de um trabalho por iteração por thread com tempos irregulares de conclusão, teremos casos de **load imbalance**, casos para os quais as opções de escalonamento **dynamic** e **guided** deverão representar uma melhor solução.

- **dynamic** – O standard OpenMP providência duas formas de escalonamento dinâmico - **dynamic** e **guided**, sendo que o último será falado no ponto seguinte. Com escalonamento dinâmico, novas porções de iterações dos ciclos serão atribuídas às threads conforme o trabalho for sendo concluído, sendo essas porções de iterações fixas.
- **guided** – com escalonamento guided, novas porções de iterações dos ciclos serão atribuídas às threads conforme o trabalho for sendo concluído, sendo essas porções de iterações dependentes relativas ao número de iterações restantes.

A.2 Uma primeira análise ao caso de estudo

Atente no seguinte excerto de código:

```
/*  
 * APina - 2016  
 * */  
/*  
 * compilar com: g++-mp-5 --std=c++11 -Wall -O2 -fopenmp -o ex2_v2 ex2_v2.cxx  
 * usar modos de escalonamento : export OMP_SCHEDULE="guided", OMP_SCHEDULE="dynamic", ←  
 * OMP_SCHEDULE="static"  
 * */  
/*  
 * Execucao:  
 * ./ex2_v2 <n.threads> <opcional- intervalo>  
 * */  
#include <cstdlib>  
#include <iostream>  
#include <random>  
using namespace std;  
#include <omp.h>  
  
#define S 1024*1024  
//#define S 100  
  
int main(int argc, char *argv[])  
{  
    int i, r, a[S], np, nr;
```

```

double T1,T2;

np = atoi(argv[1]);
if (argc == 2) nr= 99; else nr= atoi(argv[2]);

std::random_device d;
std::default_random_engine e1(d());
// a distribution that takes randomness and produces values in specified range
std::uniform_int_distribution<> dist(1,nr);

omp_set_num_threads(np);
T1 = omp_get_wtime();
#pragma omp parallel for private (r) schedule (runtime)
for (i=0 ; i < S ; i++) {
    a[i] = 0.;
    for (r = dist(e1) ; r > 0 ; r -= 20) {
        a[i] += r;
    }
}
T2 = omp_get_wtime();
cout << "Threads: " << np << " Range: " << nr << "\nTime: " << (T2-T1)*1e6 << " usecs\n";
}

```

Que iremos executar para as 3 versões de escalonamento dinâmico especificadas anteriormente, por forma a recolhermos dados estatísticos que nos permitam concluir qual das 3 versões a melhor para o nosso kernel específico.

Relativamente às estatísticas teremos interesse em ter conhecimento do momento de criação e término das threads OpenMP, tempos on e off CPU, respectivo número de CPU onde a thread está alocada, assim como as interrupções voluntárias e forçadas e seus tipos.

Assim, o ficheiro **threaded.d**, disponibilizado no contexto da disciplina, apresenta o seguinte formato:

```

#!/usr/sbin/dtrace -s
#pragma D option quiet
BEGIN
{
    baseline = walltimestamp;
    scale = 1000000;
}
sched::on-cpu
/pid == $target && !self->stamp /
{
    self->stamp = walltimestamp;
    self->lastcpu = curcpu->cpu_id;
    self->lastlgrp = curcpu->cpu_lgrp;
    self->stamp = (walltimestamp - baseline) / scale;
    printf("%9d:%-9d TID %3d CPU %3d(%d) created\n",
        self->stamp, 0, tid, curcpu->cpu_id, curcpu->cpu_lgrp);
    /*ustack(); */
}
sched::on-cpu
/pid == $target && self->stamp && self->lastcpu\
    != curcpu->cpu_id/
{
    self->delta = (walltimestamp - self->stamp) / scale;
    self->stamp = walltimestamp;
    self->stamp = (walltimestamp - baseline) / scale;
    printf("%9d:%-9d TID %3d from-CPU %d(%d) ",self->stamp,
        self->delta, tid, self->lastcpu, self->lastlgrp);
    printf("to-cpu %d(%d) CPU migration\n",
        curcpu->cpu_id, curcpu->cpu_lgrp);
    self->lastcpu = curcpu->cpu_id;
    self->latgrp = curcpu->cpu_lgrp;
}
sched::on-cpu
/pid == $target && self->stamp && self->lastcpu\
    == curcpu->cpu_id/
{
    self->delta = (walltimestamp - self->stamp) / scale;
    self->stamp = walltimestamp;
    self->stamp = (walltimestamp - baseline) / scale;
    printf("%9d:%-9d TID %3d CPU %3d(%d) ",self->stamp,
        self->delta, tid, curcpu->cpu_id, curcpu->cpu_lgrp);
    printf("restarted on the same CPU\n");
}
sched::off-cpu
/pid == $target && self->stamp /
{
    self->delta = (walltimestamp - self->stamp) / scale;
    self->stamp = walltimestamp;
    self->stamp = (walltimestamp - baseline) / scale;
}

```

```

    printf("%9d:%-9d TID %3d CPU %3d(%d) preempted\n",
        self->stamp, self->delta, tid, curcpu->cpu_id,
        curcpu->cpu_lgrp);
}
sched:::sleep
/pid == $target /
{
    self->sobj = (curlwpsinfo->pr_stype == SOBJ_MUTEX ?
        "kernel mutex" : curlwpsinfo->pr_stype == SOBJ_RWLOCK ?
        "kernel RW lock" : curlwpsinfo->pr_stype == SOBJ_CV ?
        "cond var" : curlwpsinfo->pr_stype == SOBJ_SEMA ?
        "kernel semaphore" : curlwpsinfo->pr_stype == SOBJ_USER ?
        "user-level lock" : curlwpsinfo->pr_stype == SOBJ_USER_PI ?
        "user-level PI lock" : curlwpsinfo->pr_stype == SOBJ_SHUTTLE ?
        "shuttle" : "unknown");
    self->delta = (walltimestamp - self->stamp) / scale;
    self->stamp = walltimestamp;
    self->stamp = (walltimestamp - baseline) / scale;
    printf("%9d:%-9d TID %3d sleeping on '%s'\n",
        self->stamp, self->delta, tid, self->sobj);
    /* @sleep[curlwpsinfo->pr_stype, curlwpsinfo->pr_state, ustack()]=count(); */
}
sched:::sleep
/ pid == $target && ( curlwpsinfo->pr_stype == SOBJ_CV ||
    curlwpsinfo->pr_stype == SOBJ_USER ||
    curlwpsinfo->pr_stype == SOBJ_USER_PI) /
{
    /*ustack()*/
}
sched:::sleep
/pid!=$pid && 0/
{
    @sleep[execname, curlwpsinfo->pr_stype, curlwpsinfo->pr_state, ustack()]=count();
}

```

Foram ainda realizadas algumas alterações ao mesmo, por forma a facilitar o posterior tratamento da informação, sendo ainda adicionada mais uma probe **proc:::lwp-exit** por forma a podermos contabilizar o tempo da última operação da thread.

Assim, o ficheiro final utilizado na realização dos testes apresenta o seguinte formato:

```

#!/usr/sbin/dtrace -s
#pragma D option quiet
BEGIN
{
    scale = 1000;
    baseline = timestamp;
    printf("%-20s, %-20s, %-8s, %-8s, %-8s, %-8s, %-8s, %-25s, %-15s\n",
        "TIMESTAMP (us)", "DELTA (us)", "TID", "CURR CPU", "CURR GRP", "LAST CPU", "LAST GRP", "←
        TYPE", "DESCRIPTION"
    );

    printf("%-20d, %-20s, %-8d, %-8d, %-8d, %-8d, %-8s, %-25s, %-15s\n",
        0, "",
        tid,
        curcpu->cpu_id,
        curcpu->cpu_lgrp,
        "", "", "BEGIN", ""
    );
}

sched:::on-cpu
/pid == $target && !self->stamp /
{
    self->lastcpu = curcpu->cpu_id;
    self->lastlgrp = curcpu->cpu_lgrp;
    self->stamp = timestamp - baseline;
    self->delta = timestamp - baseline;

    printf("%-20d, %-20d, %-8d, %-8d, %-8d, %-8d, %-8d, %-25s, %-15s\n",
        self->stamp / scale, self->delta / scale,
        tid,
        curcpu->cpu_id,
        curcpu->cpu_lgrp,
        self->lastcpu, self->lastlgrp,
        "CREATE", ""
    );
}

sched:::dequeue
/pid == $target
    && cpu != args[2]->cpu_id &&
(curlwpsinfo->pr_flag & PR_IDLE)/
{

```



```

self->delta = (timestamp-baseline) - self->stamp;
self->stamp = timestamp - baseline;

printf("%-20d , %-20d, %-8d , %-8d , %-8d, %-8d , %-8d, %-25s, %-15d\n",
    self->stamp / scale, self->delta / scale,
    tid,
    curcpu->cpu_id,
    curcpu->cpu_lgrp,
    self->lastcpu, self->lastlgrp,
    "THREAD STOLEN BY", args[2]->cpu_id
);
}

sched:::on-cpu
/pid == $target && self->stamp && self->lastcpu\
!= curcpu->cpu_id/
{
    self->delta = (timestamp-baseline) - self->stamp;
    self->stamp = timestamp - baseline;

    printf("%-20d , %-20d, %-8d , %-8d , %-8d, %-8d , %-8d, %-25s, %-15s\n",
        self->stamp / scale, self->delta / scale,
        tid,
        curcpu->cpu_id,
        curcpu->cpu_lgrp,
        self->lastcpu, self->lastlgrp,
        "CPU MIGRATION", ""
    );

    self->lastcpu = curcpu->cpu_id;
    self->lastlgrp = curcpu->cpu_lgrp;
}

sched:::on-cpu
/pid == $target && self->stamp && self->lastcpu\
== curcpu->cpu_id/
{
    self->delta = (timestamp -baseline)- self->stamp;
    self->stamp = timestamp - baseline;

    printf("%-20d , %-20d, %-8d , %-8d , %-8d, %-8d , %-8d, %-25s, %-15s\n",
        self->stamp / scale, self->delta / scale,
        tid,
        curcpu->cpu_id,
        curcpu->cpu_lgrp,
        self->lastcpu, self->lastlgrp,
        "RESTART ON SAME CPU", ""
    );
}

sched:::off-cpu
/pid == $target && self->stamp /
{
    self->delta = (timestamp - baseline)- self->stamp;
    self->stamp = ( timestamp - baseline );

    printf("%-20d , %-20d, %-8d , %-8d , %-8d, %-8d , %-8d, %-25s, %-15s\n",
        self->stamp / scale, self->delta / scale,
        tid,
        curcpu->cpu_id,
        curcpu->cpu_lgrp,
        self->lastcpu, self->lastlgrp,
        "PREEMPTED", ""
    );
}

sched:::sleep
/pid == $target /
{
    self->subj = (curlwpsinfo->pr_stype == SOBJ_MUTEX ?
        "kernel mutex" : curlwpsinfo->pr_stype == SOBJ_RWLOCK ?
        "kernel RW lock" : curlwpsinfo->pr_stype == SOBJ_CV ?
        "cond var" : curlwpsinfo->pr_stype == SOBJ_SEMA ?
        "kernel semaphore" : curlwpsinfo->pr_stype == SOBJ_USER ?
        "user-level lock" : curlwpsinfo->pr_stype == SOBJ_USER_PI ?
        "user-level PI lock" : curlwpsinfo->pr_stype == SOBJ_SHUTTLE ?
        "shuttle" : "unknown");

    self->delta = ( timestamp - baseline ) - self->stamp;
    self->stamp = timestamp - baseline;

    printf("%-20d , %-20d, %-8d , %-8d , %-8d, %-8d , %-8d, %-25s, %-15s\n",
        self->stamp / scale, self->delta / scale,

```

```

    tid,
    curcpu->cpu_id,
    curcpu->cpu_lgrp,
    self->lastcpu, self->lastlgrp,
    "SLEEPING ON", self->sobj
);
}

proc:::lwp-exit
/pid == $target /
{
    self->delta = (timestamp-baseline) - self->stamp;
    self->stamp = timestamp - baseline;

    printf("%-20d , %-20d, %-8d , %-8d , %-8d, %-8d , %-8d, %-25s, %-15s\n",
        self->stamp / scale, self->delta / scale,
        tid,
        curcpu->cpu_id,
        curcpu->cpu_lgrp,
        self->lastcpu, self->lastlgrp,
        "EXIT", ""
    );
}

END {
    self->stamp = timestamp - baseline;
    self->delta = timestamp - baseline;
    printf("%-20d , %-20s, %-8d , %-8d , %-8d, %-8s , %-8s, %-25s, %-15s",
        self->stamp / scale, "",
        tid,
        curcpu->cpu_id,
        curcpu->cpu_lgrp,
        "", "",
        "END", "",
    );
}

```

A.3 Relação entre o tipo de escalonamento OpenMP, número de Threads OpenMP, e tempo total para a solução, para uma gama de valores aleatórios do ciclo mais interior inalterada ([1:99]).

Com base nos valores impressos pela execução do script dtrace poderemos tratar posteriormente os dados por forma a obtermos uma representação visual, à semelhança do realizado no trabalho prático 2.

Ora, sabendo que o número de Cores físicos de cada CPU é 8, sendo que a máquina em estudo é multiprocessador, totalizando 16 Cores físicos e 32 Cores em HyperThread, devemos realizar os testes que englobem números de threads OpenMP entre 1 e 64. Desta forma, iremos realizar testes para 7 números de threads OpenMP distintos para cada opção de escalonamento, sendo estes 1, 2, 4, 8, 16, 32 e 64 threads.

Inicialmente iremos manter a gama de valores aleatórios do ciclo mais interno do código apresentado na seção A.2. Desta forma garantimos que a distribuição de carga entre threads, qualquer que seja o tipo de escalonamento não irá ser muito distinta. Analisemos portanto essa opção.

Foram realizadas 10 repetições para cada número de Thread OpenMP em teste, e para cada forma de escalonamento. Dessas 10 repetições foi escolhida a que registou um menor tempo total para a solução, produzindo a figura 2, vulgo **heatmap**, que associa o número de Threads OpenMP, tipo de escalonamento e Tempo Total para a solução. Denote que cores mais aproximadas do verde escuro representam a melhor solução alcançada:

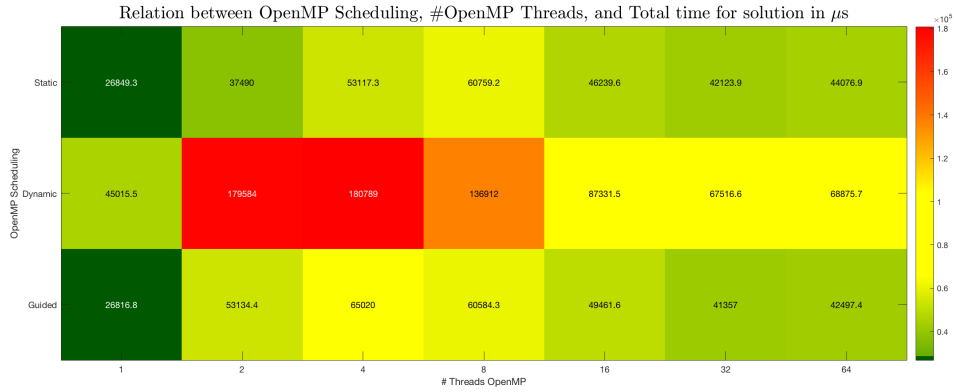


Figura 2: Relação entre o tipo de escalonamento OpenMP, número de Threads OpenMP, e tempo total para a solução, para uma gama de valores aleatórios do ciclo mais interior inalterada ([1:99]).

A.3.1 Análise gráfica para o "best-case-scenario" e "worst-case-scenario" dos tempos totais obtidos no estudo da secção A.3

Voltemos novamente aos valores de tempos totais obtidos e apresentados graficamente na figura 2. Analisemos portanto os dois outputs produzidos pela ferramenta Dtrace já devidamente ordenados pelo valor da coluna timestamp. Dessa análise resultaram as tabelas 1 e 2, que apresentaremos de seguida.

A.3.2 Análise do output da ferramenta Dtrace para o "best-case-scenario" dos tempos totais obtidos no estudo da secção A.3

Tabela 1: Análise do output da ferramenta Dtrace para o "best-case-scenario" dos tempos totais obtidos no estudo da secção A.3

Times-tamp (μs)	Delta (μs)	TID	Current CPU	Current Group	Last CPU	Last Group	Type	Description
0		1	7	1			BEGIN	
895	895	1	24	2	24	2	CREATE	
906	11	1	24	2	24	2	RESTART ON SAME CPU	
940	33	1	24	2	24	2	PREEMPTED	
977	36	1	24	2	24	2	RESTART ON SAME CPU	
1423	446	1	24	2	24	2	PREEMPTED	
2253	829	1	24	2	24	2	RESTART ON SAME CPU	
2267	14	1	24	2	24	2	PREEMPTED	
2295	27	1	24	2	24	2	RESTART ON SAME CPU	
2535	239	1	24	2	24	2	PREEMPTED	
2560	24	1	24	2	24	2	RESTART ON SAME CPU	
2569	9	1	24	2	24	2	PREEMPTED	
2604	35	1	24	2	24	2	RESTART ON SAME CPU	
32506	29901	1	24	2	24	2	SLEEPING ON	cond var
32524	18	1	24	2	24	2	PREEMPTED	
34663	2138	1	24	2	24	2	RESTART ON SAME CPU	
34718	55	1	24	2	24	2	SLEEPING ON	cond var
34724	5	1	24	2	24	2	PREEMPTED	
34888	164	1	24	2	24	2	RESTART ON SAME CPU	
34931	42	1	24	2	24	2	SLEEPING ON	shuttle
34936	4	1	24	2	24	2	PREEMPTED	
34991	54	1	24	2	24	2	RESTART ON SAME CPU	
35001	10	1	24	2	24	2	SLEEPING ON	shuttle
35005	4	1	24	2	24	2	PREEMPTED	
35031	25	1	24	2	24	2	RESTART ON SAME CPU	
35174	143	1	24	2	24	2	EXIT	
35329		1	7	1			END	

A.3.3 Análise do output da ferramenta Dtrace para o "worst-case-scenario" dos tempos totais obtidos no estudo da secção A.3

Tabela 2: Análise do output da ferramenta Dtrace para o "worst-case-scenario" dos tempos totais obtidos no estudo da secção A.3

Times-tamp (μ s)	Delta (μ s)	TID	Current CPU	Current Group	Last CPU	Last Group	Type	Description
0		1	2	1			BEGIN	
803	803	1	14	2	14	2	CREATE	
831	28	1	14	2	14	2	RESTART ON SAME CPU	
867	35	1	14	2	14	2	PREEMPTED	
936	68	1	14	2	14	2	RESTART ON SAME CPU	
1411	475	1	14	2	14	2	PREEMPTED	
2072	661	1	14	2	14	2	RESTART ON SAME CPU	
2132	59	1	14	2	14	2	PREEMPTED	
2158	25	1	14	2	14	2	RESTART ON SAME CPU	
2398	239	1	14	2	14	2	PREEMPTED	
2444	46	1	14	2	14	2	RESTART ON SAME CPU	
2470	26	1	14	2	14	2	PREEMPTED	
2513	43	1	14	2	14	2	RESTART ON SAME CPU	
6192	6192	3	13	2	13	2	CREATE	
6197	6197	2	7	1	7	1	CREATE	
6198	6	3	13	2	13	2	RESTART ON SAME CPU	
6208	3694	1	14	2	14	2	SLEEPING ON	cond var
6212	15	2	7	1	7	1	RESTART ON SAME CPU	
6216	17	3	13	2	13	2	SLEEPING ON	cond var
6220	12	1	14	2	14	2	PREEMPTED	
6223	7	3	13	2	13	2	PREEMPTED	
6251	6251	4	21	1	21	1	CREATE	
6257	5	4	21	1	21	1	RESTART ON SAME CPU	
6262	49	2	7	1	7	1	SLEEPING ON	cond var
6272	9	2	7	1	7	1	PREEMPTED	
6287	29	4	21	1	21	1	SLEEPING ON	cond var
6291	70	1	14	2	14	2	RESTART ON SAME CPU	
6295	8	4	21	1	21	1	PREEMPTED	
6305	82	3	13	2	13	2	RESTART ON SAME CPU	
6345	72	2	7	1	7	1	RESTART ON SAME CPU	
6393	97	4	21	1	21	1	RESTART ON SAME CPU	
186405	180012	4	21	1	21	1	SLEEPING ON	cond var
186410	180065	2	7	1	7	1	SLEEPING ON	cond var
186413	8	4	21	1	21	1	PREEMPTED	
186414	180108	3	13	2	13	2	SLEEPING ON	cond var
186419	180127	1	14	2	14	2	SLEEPING ON	cond var
186419	8	2	7	1	7	1	PREEMPTED	
186421	7	3	13	2	13	2	PREEMPTED	
186424	5	1	14	2	14	2	PREEMPTED	
186425	11	4	21	1	21	1	RESTART ON SAME CPU	
186438	19	2	7	1	7	1	RESTART ON SAME CPU	
186439	14	4	21	1	21	1	SLEEPING ON	cond var
186446	6	4	21	1	21	1	PREEMPTED	
186448	9	2	7	1	7	1	SLEEPING ON	cond var
186461	12	2	7	1	7	1	PREEMPTED	
186469	47	3	13	2	13	2	RESTART ON SAME CPU	
186504	35	3	13	2	13	2	SLEEPING ON	cond var
186506	81	1	14	2	14	2	RESTART ON SAME CPU	
186528	23	3	13	2	13	2	PREEMPTED	
186659	152	1	14	2	14	2	SLEEPING ON	cond var
186666	7	1	14	2	14	2	PREEMPTED	
188662	1995	1	14	2	14	2	RESTART ON SAME CPU	
188717	54	1	14	2	14	2	SLEEPING ON	cond var
188723	5	1	14	2	14	2	PREEMPTED	
188899	176	1	14	2	14	2	RESTART ON SAME CPU	
188934	34	1	14	2	14	2	SLEEPING ON	shuttle
188958	23	1	14	2	14	2	PREEMPTED	
189022	64	1	14	2	14	2	RESTART ON SAME CPU	
189033	10	1	14	2	14	2	SLEEPING ON	shuttle
189037	4	1	14	2	14	2	PREEMPTED	
189057	19	1	14	2	14	2	RESTART ON SAME CPU	
189228	171	1	14	2	14	2	SLEEPING ON	cond var
189228	2700	3	13	2	13	2	RESTART ON SAME CPU	
189229	2767	2	7	1	7	1	RESTART ON SAME CPU	
189231	2784	4	21	1	21	1	RESTART ON SAME CPU	
189235	7	1	14	2	14	2	PREEMPTED	
189238	10	3	13	2	13	2	EXIT	
189264	34	2	7	1	7	1	EXIT	
189265	30	1	14	2	14	2	RESTART ON SAME CPU	
189274	8	1	14	2	14	2	SLEEPING ON	cond var
189280	49	4	21	1	21	1	EXIT	
189280	5	1	14	2	14	2	PREEMPTED	
189293	12	1	14	2	14	2	RESTART ON SAME CPU	
189298	4	1	14	2	214	2	EXIT	
189485		1	2	1			END	

A.3.4 Análise gráfica para o "best-case-scenario" dos tempos totais obtidos no estudo da secção A.3

Ora, retomando o exemplo do trabalho prático 2, e da análise gráfica da criação e término de threads, será interessante compreender a influência de cada uma das **probes** adicionadas ao profiling no comportamento da melhor e pior versões da aplicação.

Desta forma, foram produzidas as figuras 4 a 6, ilustrando o descrito anteriormente, permitindo ainda a comparação com a mesma escala de tempos as duas versões da aplicação.

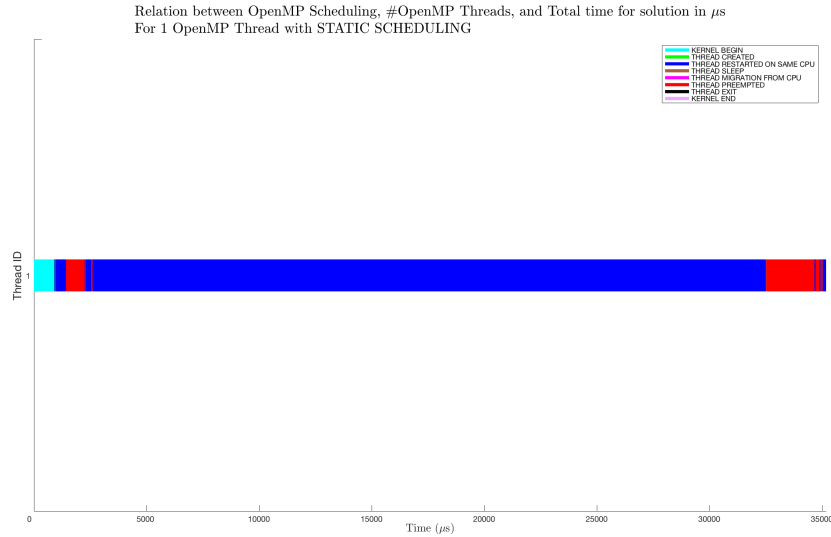


Figura 3: Análise gráfica para o "best-case-scenario" dos tempos totais obtidos no estudo da secção A.3, para uma gama de valores aleatórios do ciclo mais interior inalterada ([1:99]).

A.3.5 Análise gráfica para o "worst-case-scenario" dos tempos totais obtidos no estudo da secção A.3

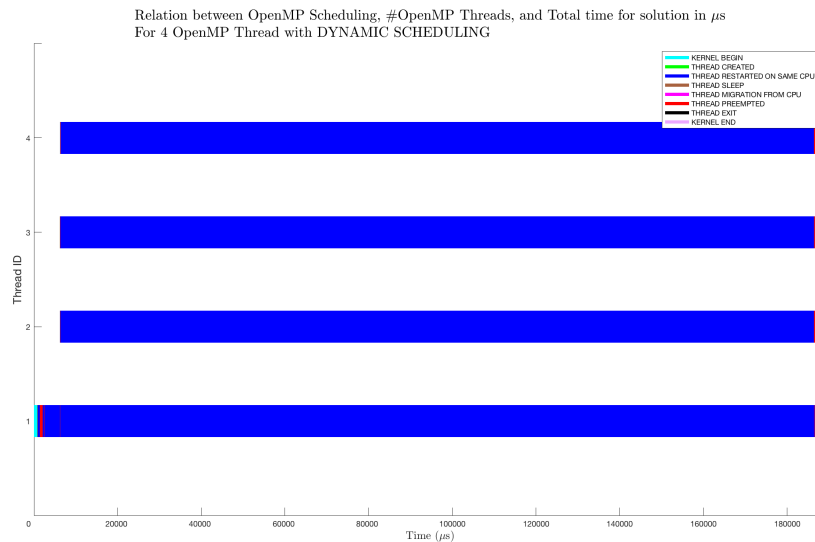


Figura 4: Análise gráfica para o "worst-case-scenario" dos tempos totais obtidos no estudo da secção A.3, para uma gama de valores aleatórios do ciclo mais interior inalterada ([1:99]).

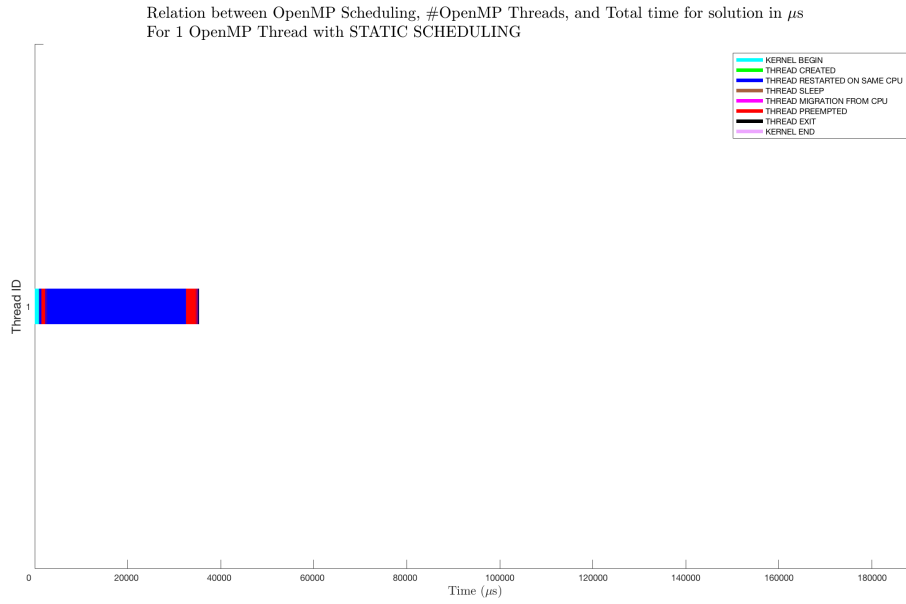


Figura 5: Análise gráfica para o "best-case-scenario" dos tempos totais obtidos no estudo da secção A.3, para uma gama de valores aleatórios do ciclo mais interior inalterada ([1:99]), com a mesma escala de tempos presente para o "worst-case-scenario".

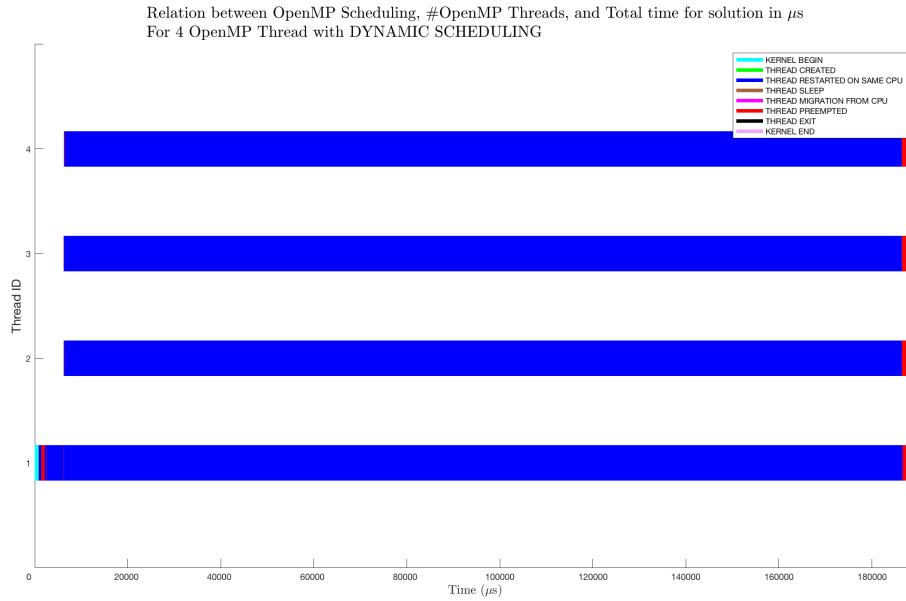


Figura 6: Análise gráfica para o "worst-case-scenario" dos tempos totais obtidos no estudo da secção A.3, para uma gama de valores aleatórios do ciclo mais interior inalterada ([1:99]).

Tendo em conta a dimensão da LLCache da máquina de teste, e a respectiva dimensão do dataset em estudo, conseguimos garantir que durante a computação, não existem bottlenecks "memory related". Ora, pela análise das imagens anteriores compreendemos que o grande diferenciador entre ambas as versões é respectivamente o overhead adicional da zona paralela. Se por uma lado, para escalonamento estático temos um menor overhead, para o caso do escalonamento dinâmico, esse mesmo overhead é completamente limitante em termos de performance.

Ora, esta análise depreende que os valores aleatórios do ciclo interno se encontram num intervalo de valores muito restrito, e é por essa mesma razão que o escalonamento estático não sofre de load imbalance quando comparado com outros tipos de escalonamento.

Testemos agora a situação onde o intervalo de valores poderá estar compreendido entre $[1$ e $262144]$.