

# DTrace per Sysadmins & Devs

---

開  
放  
的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
开放的  
açık  
open  
nyílt  
••••  
πικρ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
வெளிப்படை

**Roman Valls**

# Perquè Dtrace ?

- Quina pinta té el software mentre funciona ?
- Fins que no el modifiquem, no veiem què fa :/
  - if (DEBUG)...
  - #ifdef DEBUG...
- El problema és que això té un cost !
  - Estem parlant de soft diferent
  - Per veure'l l'alentim !
- I si canviem dinàmicament el text d'un programa en producció ?
  - Ja està inventat, pero no tant ordenat

開  
放  
的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
开放的  
açık  
open  
nyílt  
:::  
πικρ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
ఁవఱిపఁపఱఱ

# DTrace Features

- No teniem truss (l'strace de solaris)?
  - DTrace és asíncron respecte les syscalls
  - <http://www.brendangregg.com/DTrace/dtracevstruss.html>
- Instrumentació unificada
  - Permet instrumentar kernel i aplicacions
  - Pot seguir el flux del programa entre Ker i Apps
- Instrumentació en el Kernel
  - Permet instrumentar subsistemes del kernel com memòria virtual, sincronització, scheduler

開放的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
開放的  
açık  
open  
nyilt  
•••••  
πικρο  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
ఁవఱిపఁపఱఱ

# Terminologia DTrace

---

開放的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
開放的  
açık  
open  
nyílt  
:::~:::  
πικρ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
ఁవఱిపఁపఱఱ

- Una *probe* es un punt de instrumentació
- Un *proveïdor* proporciona *probes*
- Cada *probe* consta del nom, *mòdul* i *funció* que instrumenta
- Les *accions* es prenen quan les *probes* salten
- Els *predicats* permeten que accions es portin a terme només quan es compleixen certes condicions

## syscall

- one of the most important provider
- holds the entire communication from userland to kernel space
- every system call on the system

## proc

- handles: process, LWP creation and termination, signaling

## sched

- CPU scheduling: why threads are sleeping, running
- used usually to compute the CPU time, which threads are run by which CPU and for how long

## io

- a better look on iostat, regarding the I/O system
- disk input and output requests
- I/O by device, process, size, filename

## mib

- counters for management information bases
- IP, IPv6, ICMP, IPSec

開放的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
开放的  
açık  
open  
nyílt  
открыт  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
வெளிப்படை

## **fbt**

Function Boundary Tracing

entry and return points of Solaris kernel function

## **vminfo**

probes specific to VM kernel system

VM kernel statistics used in vmstat(1)

## **lockstat**

locks statistic

a better understanding of locking condition and behavior

## Data Recording Actions

### – **trace(expression)**

records the result of trace to the directed buffer

`trace(pid)` traces the current process id

`trace(execname)` traces the current application name

### **printf()**

traces a D expression

allows output style formatting

```
printf("execname is %s", execname);
```

### **printa(aggregation)**

used to display and format aggregations

```
printa(@agg1)
```

# Actions, cont.

## Data Recording Actions

### **stack()**

records a kernel stack trace

```
dtrace -n 'syscall::open:entry{stack();}'
```

### **ustack()**

records a user process stack trace

allows to inspect userland stack processes

```
dtrace -n 'syscall::open:entry{ustack();}' -c ls
```

### **jstack()**

similar with ustack(), used for Java

The stack depth frames is different than in ustack

開放的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
開放的  
açık  
open  
nyílt  
••••  
открыт  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
வெளிப்படை

# Estructura script D(Trace)

---

```
probe descriptions
/ predicate /
{
    action statements
}
```

開放的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
開放的  
açık  
open  
nyílt  
:::  
πικρ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
ОТКРЫТЫЙ  
ఁవఱిపఁదఱ

# El llenguatge D

- Un petit exemple:

```
syscall::read:entry
```

```
{  
    self -> t = timestamp;  
}
```

```
syscall::read:return
```

```
/self -> t != 0/
```

```
{  
    printf("%d/%d spent %d secs in read\n", pid, tid, timestamp -  
        t);  
}
```

開放的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
開放的  
açık  
open  
nyílt  
:::  
πικρ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
வெளிப்படை

# sysadmins: DTrace top

- A sysadmin once told me he was called to a performance problem on a production server, where all the managers were logged in checking it out.
- They couldn't see what was wrong, but he could - they were all running top. Once he convinced them not to the problem disappeared.
- One sysadmin said they had renamed top at work to "top\_is\_a\_cpu\_hog" to draw attention to the problem. Admins who continued to run top under the new name were constantly reminded of the problem in the process listing.

So is top really a hog, or is this another "admin myth" like "sync; sync; sync; reboot"

開放的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
开放的  
açık  
open  
nyilt  
:::~  
ΠΙΠΦ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
வெளிப்படை

# Top vs Prstat

Partial results

Interesting findings: *top* seems to open for each update every psinfo file from /proc. This happens 10 times (remember we started 1 sec x 10 times)

*prstat* uses a smarter method, pread. No need to open 10 times each psinfo file, the file is opened once

Does the difference matter ? So what if *top* uses more syscalls or opens more frequent the psinfo files ?

開  
放  
的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
开放的  
açık  
open  
nyílt  
•••••  
открыт  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
வெளிப்படை

# Top vs Prstat

## Measuring the CPU overhead between *prstat* and *top*

```
#!/usr/sbin/dtrace -s
syscall:::entry
/execname == $$1/
{
    self->start = vtimestamp;
}
syscall:::return
/self->start/
{
    this->time = vtimestamp - self->start;
    @Time[probefunc] = sum(this->time);
    @Time["TOTAL:"] = sum(this->time);
    self->start = 0;
}
```

開放的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
開放的  
açık  
open  
nyílt  
•••••  
открыт  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
வெளிப்படை

# Top vs Prstat, cont.

The CPU overhead for top, ~ 60ms

open64	37270
stat64	40315
memcntl	76578
mmap	89981
pollsys	146723
write	332862
ioctl	591361
getdents64	2856879
close	2983845
read	21667295
open	29061813
TOTAL :	58105511

The CPU overhead for prstat, ~ 30ms

memcntl	201860
exece	321309
close	355196
write	708567
open	883755
doorfs	1085825
getdents	2870547
pread	21986669
TOTAL :	28924120

# Top vs Prstat, cont.

## Conclusions

Using DTrace we were able to understand how *prstat* and *top* works

This was not intended to be a benchmark exercise !

For each screen update *top* opens, reads and closes a *psinfo* file for every process. *prstat* only does a read

On very fast machines the difference is small, however it very much depends how many processes are running. Try to experiment with different number of processes on different hardware

Try to discover other similar monitoring applications like *top*

開  
放  
的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
开放的  
açık  
open  
nyílt  
:::~  
ΠΙΠΦ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
வெளிப்படை

# DTrace oneliners

## System Calls Count by Application

```
$ dtrace -n 'syscall::entry@[execname] = count();}'
```

## System Calls Count by Application and Process

```
$ dtrace -n 'syscall::entry@[execname,pid] = count();}'
```

## How many times a file has been opened

```
$ dtrace -n 'syscall::open:entry@[copyinstr(arg0)] = count();}'
```

## Files Opened by process

```
$ dtrace -qn 'syscall::open*:entry{ printf("%s,\n%s\n",execname,copyinstr(arg0)); }'
```

## Read Bytes by process

```
$ dtrace -n 'sysinfo::readch{ @[execname] = sum(arg0);}'
```

## Write Bytes by process

```
$ dtrace -n 'sysinfo::writech{ @[execname] = sum(arg0);}'
```

## How big a read is

```
$ dtrace -n 'syscall::read:entry@[execname] = quantize(arg2);}'
```

## How big a write is

```
$ dtrace -n 'syscall::write:entry@[execname] = quantize(arg2);}'
```

## Disk size by process

```
$ dtrace -qn 'io:::start{printf("%d %s %d\n",pid,execname,args[0]->b_bcount); }'
```

## High system time

```
$ dtrace -n profile-501' {@[stack()] = count()}END{trunc(@, 25)}'
```

## What processes are using fork

```
$ dtrace -n 'syscall::fork*:entry{printf("%s %d",execname,pid);}'
```

## My application is doing nothing

```
$ dtrace -n sched:::off-cpu' {@[ustack()] = count()}' -p pid
```

開放的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
開放的  
açık  
open  
nyílt  
:::  
ΠΙΠΠ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
ОТКРЫТЫЙ  
வெளிப்படை

# Projecte D-Light

---

- Un GUI per Dtrace amigable
- Corre sobre Sun Studio 12
- Unifica el profiling d'aplicació i sistema
- Mostra interactivament com respon l'aplicació
- Es pot identificar la línia de codi on ets en qualsevol moment, molt útil per identificar ràpidament punts a millorar

開放的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
开放的  
açık  
open  
nyílt  
•••••  
πικρο  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
открытый  
வெளிப்படை

# Perl & DTrace

From: Andy Armstrong andy hexten net

I now have a dtraced bleadperl which runs no slower than the clean version:

orig		dtrace		ratio
=====				
2.51239896		2.47904992		0.98672622
2.50696611		2.46995783		0.98523782
2.52996778		2.47554016		0.97848683
2.50457382		2.47788405		0.98934359
2.50622296		2.46893001		0.98511986

In fact in those tests the dtraced version is running marginally faster. I'm putting that down to a happy code alignment or somesuch.

開放的  
열린  
مفتوح  
libre  
मुक्त  
ಮುಕ್ತ  
livre  
libero  
ముక్త  
开放的  
açık  
open  
nyílt  
•••••  
ΠΙΠΦ  
オープン  
livre  
ανοικτό  
offen  
otevřený  
öppen  
ОТКРЫТЫЙ  
வெளிப்படை

# Benvolguts sysadmins... :D

---

開放的  
열린  
مفتوح

libre

मुक्त

ಮುಕ್ತ

livre

libero

ముక్త

開放的

açık

open

nyílt

⋯⋯⋯

πληρ

オープン

livre

ανοικτό

offen

otevřený

öppen

открытый

வெளிப்படை

...voldria privilegis per DTrace a les màquines

```
usermod -K defaultpriv=basic,dtrace_proc,dtrace_user,dtrace_kernel $USER
```

...i potser també el DtraceToolkit :-)~~~

## Preguntes ?