Solaris – Dtrace (mit is rejt a motorház)

Erik Fischer
Principal Engineer
erik.fischer@sun.com

Hibakeresés – in vitro

- Végzetes, nem reprodukálható hibák
  - Core dump
  - Postmortem mdb(1), dbx(1)
- Tranziens hibák
  - Programozási hibák vagy percepcionális teljesítmény problémák
    - Ad hoc technikák vagy szinte semmi
    - truss(1), mdb(1), *stat(1M), sar(1), prstat(1)
    - Sun Studio Performance Analyzer

Hibakeresés – in vivo

- Invaziv technikák
  - Bináris instrumentálás
  - Forrás szintű instrumentálás
  - Interposer library-k
  - Debug library-k
  - Debug kernel
  - Általában durva beavatkozások
  - Általában lassú
  - Általában nagy az additív hibák injekciójának esélye

Elvárások

- Az ideális dinamikus hibakereső rendszer
  - Képes információ gyűjtésére
  - Mindezt éles rendszereken is
  - Gyakorlatilag teljesen biztonságosan
  - Gyakotrlatilag teljesítmény veszteség nélkül

DTrace

- Interpretált probe alapú nyelv, prédikátumokkal és akciókkal
- Dinamikus, függvény be- és kilépési pontokat instrumentáló rendszer
- Kernel modul
- Jellemzők:
  - User és Kernel rétegben is működik
  - 40000+ probe egy átlagos Solaris 10 rendszeren
  - Alapállapotban csak root-ként működik
  - 3 privilégiumot igényel: dtrace_kernel, dtrace_proc és dtrace_user
  - 410 oldalas dokumentáció

Probe

- Az instrumentáció pontos helye egy hierarchiában (leírója egy 'n'-es)

  <provider, module, function, probe_name>
- Egy provider bocsájtja a rendelkezésünkre
- Minden provider modulokra és függvényekre tagolódik
- A probe-nak van neve (általában entry és return)
- dtrace -l

Provider-ek

- Szép számban akadnak
  - fbt – szinte minden entry és return a kernelben (~39000 db)
  - syscall – syscall tábla (~450 db)
  - profile
  - lockstat
  - proc
  - sysinfo
  - vminfo
  - sched
  - io
  - fpuinfo
  - sdt (~190 db)
  - mib – TCP/IP-hez kapcsolódó függvények (~430 db)
  - pid

DTrace fogyasztók

- Gyakorlatilag nincs felső korlátja, a DTrace kernel modul multiplexel
- dtrace(1M) a command line fogyasztó
- Vannak és lehetnek programozott fogyasztók is

A DTrace nyelve: D

- C, awk és perl keveréke
- Teljes hozzáférés a kernel C típusaihoz header fájlokon keresztül
- String támogatás
- Számos különleges nyelvi megoldás

Szkriptek

- dtrace(1M) támogatja a szkripteket

#! /usr/sbin/dtrace –s

- Szkript szintaktika

```
[
provider:module:function:name[,
provider:module:function:name]*
[/predicate/]
{
 action;
 [action;]*
}
]+
```

Változók

- Skalár
  - Szokásos típusok (char, short, int, long, long long, float, double, long double)
  - **NINCS FLOAT ARITHMETIKA!**
- Asszociatív array-ek
  - Továbbfejlesztett Perl hash
  - A kulcs egy 'n'-es
  - Pl. array[execname, timestamp]

Beépített változók

- pid – az aktuális processz ID
- tid – az aktuális thread ID
- execname – az aktuális programnév
- *curthread – az aktuális thread kernel struktúrája
- *curpsinfo – az aktuális processz állapotleíró struktúrája
- *curlwpsinfo – az aktuális lwp állapotleíró struktúrája
- timestamp – a bootolás óta eltelt idő [ns]
- probprov, probemod, probefunc, probename – a probe adatai

...

Aggregációk

- Natív DTrace típus
- Egy aggregáció olyan f(x) függvény, ahol x egy tetszőleges n elemű adathalmaz és:

$$f(f(x_0) \cup f(x_1) \cup \ldots \cup f(x_n)) = f(x_0 \cup x_1 \cup \ldots \cup x_n)$$

- Ilyenek a count, a sum, az avg, a max és a min (pl. a median nem az, éppen ezért nincs is ilyen)
- Speciális aggregáció a quantize és az lquantize

Thread lokális változók

- Azonos név, de per thread adat
- Jelölése: self->
- A nem definiált változóknak nulla az értékük
- Ha nullázunk egy thread lokális változót, avval deallokáljuk (!!)

Intelligens megjelentés

- printf() – trace() a printf(3C) jótulajdonságaival
- printa() – a printf(3C), de aggregációkra

Akciók

- trace() – az argumentum értékét a trace pufferbe helyezi
- tracemem() – az argumentumban megadott memóriacímtől adott hosszig ment értékét a trace pufferbe
- stack() – a kernel stack trace-t a trace pufferbe helyezi
- ustack() – a user stack trace-t a trace pufferbe helyezi
- exit() – a DTrace fogyasztót kilépésre kényszeríti

...

Destruktív akciók

- A –w flag-gel engedélyezni kell
- stop() – megállítja az aktuális processzt
- raise() – szignált küld az adott processznek
- panic() – kernel pánik
- chill() – adott időre leállítja a processzt
- copyout() és copyoutstr() – adott változóból adott számú adatot egy memória helyre másol

Pragmák

#pragma D option <name>[=<value>]

- Alapvetően hangolható paraméter beállítások
- Speciális kapcsolók beállítása

#pragma D option destructive

#pragma D option quiet

Következő verziók

- A Solaris 11 (OpenSolaris, Nevada) és a Solaris 10 update-ek számos bugfix-et, új nyelvi elemet és funkciót tartalmaznak már most is

Syscall 1.

```
#! /usr/sbin/dtrace -s
#
# pragma D option quiet

syscall:::entry
{
  @cnt[execname, probefunc]=count();
}
```

Syscall 2.

```
#! /usr/sbin/dtrace -s
#
# pragma D option quiet

syscall:::entry
{
 @cnt[execname, probefunc]=count();
}
END
{
 printa("%s, syscall %s == %@d\n", @cnt);
}
```

Syscall 3.

```
#! /usr/sbin/dtrace -s
#
# pragma D option quiet

syscall:::entry
{
 @cnt[execname, probefunc]=count();
}
profile-5s
{
 printa("%s, syscall %s == %@d\n", @cnt);
 trunc(@cnt, 0);
 printf("\n\n\n");
}
```

Syscall 4.

```
#! /usr/sbin/dtrace -s
#
# pragma D option quiet

syscall:::entry
{
 self->ts=timestamp;
}
syscall:::return
/self->ts/
{
 @cnt[execname, probefunc]=avg((timestamp-self->ts)/1000000);
 self->ts=0;
}
profile-5s
{
 printa("%s, syscall %s == %@d\n", @cnt);
 trunc(@cnt, 0);
 printf("\n\n\n");
}
```

Syscall 5.

```
#! /usr/sbin/dtrace -s
#
# pragma D option quiet

syscall:::entry
{
 self->ts=timestamp;
}
syscall:::return
/self->ts/
{
 @cnt[execname, probefunc]=quantize((timestamp-self->ts)/1000000);
 self->ts=0;
}
profile-5s
{
 printa("%s, syscall %s\n%@d\n", @cnt);
 trunc(@cnt, 0);
 printf("\n\n\n");
}
```

Read – write 1.

```
#! /usr/sbin/dtrace -s
#
# pragma D option quiet

syscall::read:entry
{
 self->tsr=timestamp;
}
syscall::read:return
/self->tsr/
{
 @rtime[execname] =
    quantize(timestamp - self->tsr);
 self->tsr=0;
}
```

```
syscall::write:entry
{
 self->tsw=timestamp;
}
syscall::write:return
/self->tsw/
{
 @wtime[execname] =
    quantize(timestamp - self->tsw);
 self->tsw=0;
}
profile-2s
{
 printa("%s read\n%@d\n", @rtime);
 trunc(@rtime, 0);
 printa("%s write\n%@d\n", @wtime);
 trunc(@wtime, 0);
 printf("\n\n\n");
}
```

Read – write 2.

```
syscall::read:entry
/execname != "dtrace"/
{
 self->tsr=timestamp;
 self->fdescr=arg0;
}
syscall::read:return
/self->tsr/
{
 printf("%s read %d bytes
   from descriptor: %d\n",
   execname, arg0, self->fdescr);
 self->tsr=0;
 self->fdescr=0;
}
```

```
syscall::write:entry
/execname != "dtrace"/
{
 self->tsw=timestamp;
 self->fdescw=arg0;
}
syscall::write:return
/self->tsw/
{
 printf("%s written %d bytes
    to descriptor: %d\n", execname,
    arg0, self->fdescw);
 self->tsw=0;
 self->fdescw=0;
}
```

Read – write 3.

```
syscall::read:entry
/execname != "dtrace"/
{
 self->tsr=timestamp;
 self->fdescr=arg0;
 self->rbuff=arg1;
}
syscall::read:return
/self->tsr/
{
 self->read =
   stringof(copyin(self->rbuff, arg0));
 printf("%s (%d) read %d bytes
   from descriptor: %d\n", execname,
   pid, arg0, self->fdescr);
 printf("   Read:%S\n", self->read);
 self->tsr=0;
 self->fdescr=0;
 self->rbuff=0;
}
```

```
syscall::write:entry
/execname != "dtrace"/
{
 self->tsw=timestamp;
 self->fdescw=arg0;
 self->written =
   stringof(copyin(arg1, arg2));
}
syscall::write:return
/self->tsw/
{
 printf("%s (%d) written %d bytes
   to descriptor: %d\n", execname,
   pid, arg0, self->fdescw);
 printf("  Written:%S\n",
   self->written);
 self->tsw=0;
 self->fdescw=0;
 self->written=0;
}
```

Intermezzo 1.

- IO provider – start probe, 3 argumentum
  - bufinfo_t – args[0]

```
typedef struct bufinfo {
   int b_flags;              /* flags */
   size_t b_bcount;          /* number of bytes */
} bufinfo_t;
```

  - devinfo_t – args[1]

```
typedef struct devinfo {
   string dev_name;          /* name of device */
} devinfo_t;
```

  - fileinfo_t – args[2]

```
typedef struct fileinfo {
   string fi_pathname;       /* full pathname */
} fileinfo_t;
```

Fizikai I/O

```
#! /usr/sbin/dtrace -s
#
# pragma D option quiet

io:::start
{
 printf("%s (%d) device:%s i/o size:%d
  %s filepath:%s\n", execname, pid,
  args[1]->dev_name, args[0]->b_bcount,
  args[0]->b_flags & B_READ ? "Read" : "Write",
  args[2]->fi_pathname);
}
```

Intermezzo 2.

- Sched provider – on-cpu probe
- Három lényegi pointer
  - **cpuinfo_t *curcpu**
  - **lwpsinfo_t *curlwpsinfo**
  - **psinfo_t *curpsinfo**
- man –s 4 proc, illetve

```
typedef struct cpuinfo {
  processorid_t cpu_id;          /* CPU identifier */
  psetid_t cpu_pset;             /* processor set identifier */
  chipid_t cpu_chip;             /* chip identifier */
  lgrp_id_t cpu_lgrp;            /* locality group identifer */
  processor_info_t cpu_info;     /* CPU information */
} cpuinfo_t;
```

- Ahonnan man –s 2 processor_info

Ütemező 1.

```
#! /usr/sbin/dtrace -s
#
# pragma D option quiet

sched:::on-cpu
/execname != "sched"/
{
 printf("%s (%d) on CPU:%d (%s %dMHz) LWPs:%d LID:%d\n",
   execname, curpsinfo->pr_pid, curcpu->cpu_id,
   curcpu->cpu_info.pi_processor_type,  curcpu->cpu_info.pi_clock,
   curpsinfo->pr_nlwp, curlwpsinfo->pr_lwpid);
}
```

Ütemező 2.

```
#! /usr/sbin/dtrace -s
#
# pragma D option quiet

sched:::on-cpu
/execname == "nscd"/
{
 self->ts=timestamp;
}
sched:::off-cpu
/self->ts/
{
 @tim[cpu, tid]=quantize((timestamp-self->ts)/1000);
 self->ts=0;
}
profile-5s
{
 printa("On CPU %d thread %d running time distribution [us]: %@d\n", @tim);
 trunc(@tim);
}
```

pid

```
#!/usr/sbin/dtrace -s
#
# pragma D option quiet

pid$1:libc.so.1:malloc:entry
{
 self->trace = 1;
 self->size = arg0;
}
pid$1:libc.so.1:malloc:return
/self->trace == 1/
{
 printf("Ptr=0x%p Size=%d", arg1, self->size);
 self->trace = 0;
 self->size = 0;
}
pid$1:libc.so.1:free:entry
{
 printf("Ptr=0x%p ", arg0);
}
```

Destruktív művelet

```
#! /usr/sbin/dtrace -s
#
# pragma D option quiet
# pragma D option destructive

syscall::uname:entry
{
 self->ustr=arg0;
}
syscall::uname:return
/self->ustr/
{
 copyoutstr("5.12", self->ustr+2*257, 257);
 copyoutstr(“SolErix", self->ustr+3*257, 257);
}
```

Erik Fischer
Principal Engineer
erik.fischer@sun.com