



## DTrace - Informationsgewinnung leicht gemacht

Thomas Nau, kiz (Thomas.Nau@uni-ulm.de)

# Kommunikations- und Informationszentrum (kiz)

- Die Aufgaben der "Abteilung Infrastruktur" umfassen
  - Cluster basierende universitätsweite Mail-, LDAP-, Portal-, Datenbank- und File-Services
  - Betreuung von ca. 600 Desktop und Laptop Arbeitsplätzen
    - 25% Linux, 75% Windows
  - Backup Service für die Universitäten Konstanz und Ulm
  - HPC Cluster für die Universitäten Konstanz, Stuttgart und Ulm
  - 4 lokale Netzwerke plus flächendeckendes Campus WLAN und MAN im Ulmer Stadtbereich
  - Telefonanlage mit ca. 14.000 Anschlüssen unter Einsatz von VoIP und 2-Draht Technik
  - Azubi Ausbildung

## Tools: *vmstat(1m)*

- liefert statistische Daten über Kernel Threads, Platten IO, CPU Last, virtuellen Speicher und traps
- eines der nützlichsten Tools da es Anhaltspunkte liefert wo Engpässe zu finden sind
  - hohe Anzahl von system-calls oder context-switches
  - viele page-in oder page-out Ereignisse

```
jedi# vmstat 5
kthr      memory          page        disk        faults        cpu
 r  b  w    swap  free  re  mf pi po fr de sr m0 m1 m3 m1   in   sy   cs  us  sy  id
 0  0  0 8620144 4617688 67 69 1  1  1  0  0  0  1  0  0 6147 1254 5529  0  1 99
 0  0  0 8702320 4730536  6  5  0  0  0  0  0  0  0  0  0 8010  532 7370  0  2 98
 0  0  0 8788856 4816776  6  1  0  0  0  0  0  0  0  0  0 7990  534 7428  0  2 98
^C
```

## Tools: *prstat(1m)*

- liefert Informationen über laufende Prozesse und deren Threads im Hinblick auf CPU Zeiten, Latenz Zeiten, context-switches, ...
  - falls Anwendungsprobleme vermutet werden ist *prstat(1m)* ein sehr guter Ausgangspunkt für weitere Untersuchungen

```
jedi# prstat -Lm
  PID USERNAME  USR  SYS  TRP  TFL  DFL  LCK  SLP  LAT  VCX  ICX  SCL  SIG  PROCESS/LWPID
16480 xvm          6.9  9.8  0.0  0.0  0.0   27   54  1.8  30K  114  .2M  0  qemu-dm/3
   363 xvm          0.1  0.2  0.0  0.0  0.0   0.0  100  0.0   4    1   2K  0  xenstored/1
16374 root         0.0  0.1  0.0  0.0  0.0   100  0.0  0.0   10    0   1K  0  dtrace/1
   1644 xvm          0.1  0.1  0.0  0.0  0.0   33   66  0.0  569    7  835  0  qemu-dm/3
   2399 root         0.0  0.1  0.0  0.0  0.0   0.0  100  0.0   49    0  388  0  sshd/1
16376 root         0.0  0.1  0.0  0.0  0.0   0.0  100  0.0   38    0  297  0  prstat/1
11705 xvm          0.0  0.1  0.0  0.0  0.0   50   50  0.0  576   15  858  0  qemu-dm/4
16536 root         0.0  0.1  0.0  0.0  0.0   0.0  100  0.0   48    0  286  0  vncviewer/1
```



# prstat(1m) Erklärungsnot

PID	USERNAME	SIZE	RSS	STATE	PRI	NICE	TIME	CPU	PROCESS/NLWP
937	nau	8644K	2568K	sleep	10	0	0:00:25	13%	loop.sh/1
445	root	11M	3652K	sleep	59	0	0:00:04	2.4%	nscd/44
698	nau	67M	35M	sleep	59	0	0:00:07	0.3%	Xorg/3
912	nau	76M	18M	sleep	59	0	0:00:00	0.3%	gnome-terminal/2
870	nau	27M	16M	sleep	59	0	0:00:00	0.1%	metacity/1
6919	nau	8988K	3148K	cpu0	59	0	0:00:00	0.0%	prstat/1
873	nau	12M	4976K	sleep	59	0	0:00:00	0.0%	xscreensaver/1
872	nau	90M	30M	sleep	49	0	0:00:00	0.0%	nautilus/1
884	nau	28M	16M	sleep	59	0	0:00:00	0.0%	wnck-applet/1
865	nau	29M	17M	sleep	59	0	0:00:00	0.0%	gnome-settings-/1
871	nau	80M	21M	sleep	59	0	0:00:00	0.0%	gnome-panel/1
11614	nau	7432K	1148K	sleep	59	0	0:00:00	0.0%	sleep/1
897	nau	78M	18M	sleep	59	0	0:00:00	0.0%	mixer_applet2/2
510	root	10M	1756K	sleep	59	0	0:00:00	0.0%	VBoxService/7
5	root	0K	0K	sleep	99	-20	0:00:00	0.0%	zpool-rpool/136
3833	nau	8680K	2808K	sleep	59	0	0:00:00	0.0%	bash/1
520	root	4608K	3304K	sleep	59	0	0:00:00	0.0%	console-kit-dae/2
693	root	9944K	3352K	sleep	59	0	0:00:00	0.0%	gdm-binary/2
197	root	2132K	1480K	sleep	59	0	0:00:00	0.0%	pfexecd/3
248	root	2548K	1432K	sleep	60	-20	0:00:00	0.0%	zonestatd/5
345	root	7288K	952K	sleep	59	0	0:00:00	0.0%	iscsid/2
163	daemon	7352K	1244K	sleep	59	0	0:00:00	0.0%	kcfd/2
552	root	8152K	1468K	sleep	59	0	0:00:00	0.0%	automountd/2
Total: 85 processes, 420 lwps, load averages:							1.07, 0.61, 0.29		

## Tools: *truss(1)*

- *truss(1)* ist nicht dynamisch was eine Analyse vorübergehender oder kurzlebiger Probleme erschwert oder unmöglich macht
  - stoppt die Anwendung um die notwendigen Daten zu sammeln bevor der Prozess fortgesetzt wird
  - die Anwendung, besonders das Timing, wird beeinflusst
- die Auswertung zusammenhängender Prozesse wie etwa einer Login-Session ist sehr aufwändig, wenn überhaupt möglich
- die Kernel Grenze kann nicht überschritten werden

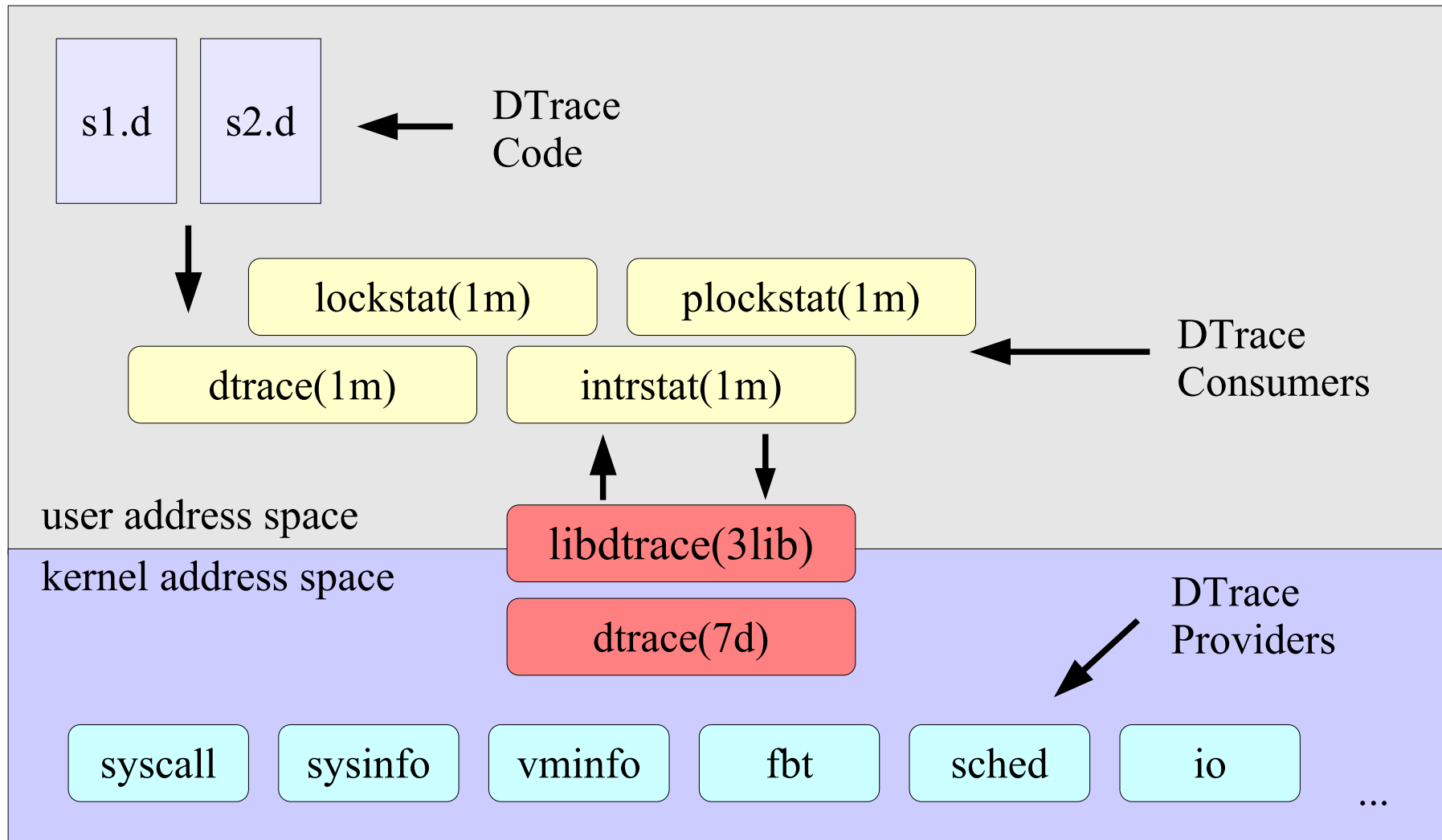
**Wie kommt man weiter,  
wenn die beschriebenen  
Tools nicht gut genug sind?**

# DTrace, die "Dynamic Tracing Facility"

- gibt uns die Möglichkeit dynamisch und effizient Kernel-, Bibliotheks- und Anwendungs-Code zu instrumentieren
- abhängig vom System und den geladenen Modulen stehen derzeit über 90.000 „Probes“ zur Verfügung die sich beliebig und effizient ein- bzw. ausschalten lassen
- DTrace ist durch die "C" ähnliche Sprache "D" skriptfähig
  - wird im Kernel Kontext ausgeführt und beinhaltet daher aus Sicherheitsgründen keine Schleifen-Konstrukte (for, while, ...)
- immer mehr Solaris Tools verwenden DTrace



# DTrace Block Schaubild



# Provider

- stellen sogenannte Probes zur Verfügung
- decken spezifische Bereiche ab und **bereiten Daten auf**
  - proc:  
Liefert Informationen über Prozesse und Threads
  - syscall:  
Probes für alle Ein- und Austrittspunkte der Solaris system-calls
  - io:  
Disk-IO bezogene Probes
  - fbt (function boundary tracing):  
Probes für die meisten Solaris Kernel Funktionen
- neue Provider und Probes werden auch im Rahmen von Updates, etwa Solaris 11, bereitgestellt

# Probes

- sind Triggerpunkte im Kernel oder in Anwendungen die vielfältige Aktionen auslösen können
  - Aufzeichnung oder Ausgabe von Kernel- oder User-Stacks
  - Ausgabe von Datenstrukturen oder Speicherinhalten
  - Manipulation von Daten oder Speicherinhalten
  - ...

- Namens-Syntax für Probes

provider:module:function:name

*sched:unix:resume:off-cpu*

# Beispiel: wer öffnet welche Datei?

```
obi-wan# dtrace -q -n \  
  'syscall::open*:entry {  
    printf("%-20s %s\n", execname, copyinstr(arg0));  
  }'
```

```
httpd          /www/htdocs/guc/bilder/grafik/m_kontakt.gif  
httpd          /www/cms/uploads/pics/rw_logo2_03.png  
mysqld         /www/mysqlldata/cms/fe_users.MYI  
mysqld         ./cms/fe_users.MYD  
mysqld         /www/mysqlldata/cms/fe_groups.MYI  
...
```

## “D” Sprachstruktur

- "D" definiert eine Art Unterprogramm-Sammlung
- einfache Struktur die sequentiell von oben nach unten ausgewertet wird; „Bedingungen“ sind optional

```
Proben-Namen  
/ Bedingung /  
{  
    Aktionen  
}
```

- aus Sicherheitsgründen kennt "D" keine Schleifen, ...
- einfach in eigene Anwendungen integrierbar

# Variable, Typen und Operatoren

- "C" "look and feel" mit geringen Unterschieden
  - int8\_t, uintptr\_t, ...*
- Skalare, Strings, Zeiger, *structs* und *unions* sind verfügbar
  - provider und consumer laufen in unterschiedlichen Adressräumen
- Variable müssen nicht vor Verwendung definiert werden
  - viele schon vordefiniert: *pid, uid, execname, curcpu, ...*
  - lesender Zugriff auf kernel variable möglich: *`kmem\_flags*
- grundlegende Arithmetik-, Logik- und Vergleichs-Operatoren stehen zur Verfügung
- Vergleichs-Operatoren können auf Strings angewendet werden; liefern 1 im Falle von Gleichheit sonst 0



# Ausgabe bzw. Sammeln von Daten

- `printf()` arbeitet wie in "C" mit Format-Strings
  - `printa()` arbeitet analog zu `printf()` mit "aggregations"
  - nützliche Erweiterungen verfügbar
    - 'a'     Ausgabe eines Zeigers als Kernel Symbol
    - 'p'     Hexadezimal Ausgabe eines Zeigers
    - 'S'     Ausgabe von Strings wobei nicht druckbare Zeichen mit "\" dargestellt werden
    - 'Y'     formatiert ein Argument "Nanosekunden seit 1.1.1970" als für Menschen verständlichen String
- `stack()`, `ustack()`
- `tracemem()`

# DTrace's Warp Antrieb: Aggregations

- Aggregations nutzen eine besondere mathematische Eigenschaft bestimmter Funktionen aus
  - die Anwendung der Funktion auf Teilmengen der Daten und anschließend auf die Zwischenergebnisse liefert das identische Ergebnis wie die Anwendung auf die Gesamtmenge
  - `count()`, `min()`, `max()`, `avg()`, `quantize()`, `lquantize()`
- helfen den Speicherbedarf gering zu halten
  - es besteht keine Notwendigkeit alle Daten zwischenzuspeichern
  - Probleme mit der Skalierung werden vermieden
- der Index von Aggregations ist nahezu beliebig, etwa *ustack()* und *stack()* oder *execname*, *pid*, ...

# SUM Aggregation

$$\sum 1, 2, 3, \dots 99, 100 = 5050$$

$$\sum 1, \dots 10 = 55$$

$$\sum 11, \dots 20 = 155$$

...

$$\sum 91, \dots 100 = 955$$

$$\left. \begin{array}{l} \sum 1, \dots 10 = 55 \\ \sum 11, \dots 20 = 155 \\ \dots \\ \sum 91, \dots 100 = 955 \end{array} \right\} \sum = 5050$$

# Solaris 11: IP Provider Probes

- send                      Solaris Netzwerk Stack verschickt ein IP Paket
- receive                   Solaris Netzwerk Stack empfängt ein IP Paket
- der nur ab Solaris 11 verfügbare IP Provider verwendet für die Datenübergabe 6 Zeiger auf die folgenden Strukturen:

pktinfo\_t \*  
ifinfo\_t \*

csinfo\_t \*  
ipv4info\_t \*

ipinfo\_t \*  
ipv6info\_t \*

## *ipinfo\_t* und *ipv4info\_t* Definitionen

```
typedef struct ipinfo {
    uint8_t ip_ver;           /* IP version (4, 6)      */
    uint16_t ip_plength;     /* payload length        */
    string ip_saddr;         /* source address        */
    string ip_daddr;         /* destination address   */
} ipinfo_t;

typedef struct ipv4info {
    ...
    uint8_t ipv4_protocol;   /* next level protocol   */
    string ipv4_protostr;    /* same as a string      */
    ...
    ipaddr_t ipv4_src;       /* source address        */
    ipaddr_t ipv4_dst;       /* destination address   */
    string ipv4_saddr;       /* src address, string   */
    string ipv4_daddr;       /* dest address, string  */
    ...
} ipv4info_t;
```

# Beispiel: Verteilung der IP Paket Größe

- basiert auf Beispiel von  
*<https://wikis.oracle.com/display/DTrace/ip+Provider>*
- gut geeignet für IP basierte File- oder Storage Server

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

#pragma D option quiet

ip:::send
{
    @c[args[2]->ip_daddr, execname] =
        quantize(args[2]->ip_plength);
}
2^n Verteilung
```



# Beispiel: Verteilung der IP Paket Größe

...  
192.168.23.23

nfsd

value	----- Distribution -----	count
16		0
32		1
64		0
128	@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@	244
256	@@@@@@@@@@@@@	88
512		0
1024	@@@@	32
2048		0
...		

# Hilfreiche Funktionen für Aggregations

- *trunc( @aggr [, n] )*  
löscht eine Aggregation oder Teile davon
  - $n > 0$  die obersten  $n$  Einträge bleiben erhalten
  - $n < 0$  die untersten  $n$  Einträge bleiben erhalten
- *clear( @aggr )*  
setzt die Werte einer Aggregation auf 0
- *normalize( @aggr, val )*  
dividiert alle Werte durch *val*
- *denormalize()*  
macht die Normierung rückgängig
- **Tipp:** im Zusammenspiel mit der *tick probe* lassen sich einfach top-ten artige Ausgaben realisieren

# Sortierung von Aggregations

- die Sortierung lässt sich über Optionen steuern

```
setopt("aggsortkey", true);
```

- Möglichkeiten

aggsortkey	Sortierung nach Index, nicht Wert
aggsortkeypos	Nummer des Index nach dem sortiert wird
aggsortrev	Umkehrung der Reihenfolge

# Beispiel: Netzverkehr einzelner Clients

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

```
#pragma D option quiet
```

```
dtrace:::BEGIN {  
    ts = timestamp;  
}
```

nur in Solaris 11

```
ip:::send {  
    @[args[2]->ip_daddr, probename] =  
        sum(args[2]->ip_plength);  
}
```

```
ip:::receive {  
    @[args[2]->ip_saddr, probename] =  
        sum(args[2]->ip_plength);  
}
```

## Beispiel: Netzverkehr einzelner Clients

```
/*
 * print top-n
 * normalized to bytes per second
 */
tick-$1 {
    trunc(@, $2);
    normalize(@, (timestamp-ts) / 1000000000);
    printf("\n%Y\n", walltimestamp);
    printa("%-15s %-10s %@15d\n", @);
    ts = timestamp;
    trunc(@);          /* start from scratch */
}
```

# Beispiel: Netzverkehr einzelner Clients

```
obi-wan# ./ip_top.d 2s 5
```

```
2009 Jun 4 16:35:36
```

134.60.84.144	receive	34276
134.60.84.170	receive	49792
134.60.215.64	receive	54448
134.60.1.50	receive	301724
134.60.40.100	receive	1304200

```
2009 Jun 4 16:35:38
```

134.60.84.131	send	101752
134.60.84.170	receive	124928
134.60.84.170	send	303596
134.60.1.50	receive	408576
134.60.2.117	receive	580712

```
^C
```



## Der *pid* Provider

- der *pid* Provider erlaubt die Verfolgung von Funktion oder Instruktionen einer Anwendung inkl. gelinkter Bibliotheken
  - Einschränkung: inlining durch den Compiler
- einzelne Befehle werden als Offset in Bytes bezogen auf den Anfang der Funktion angegeben

*pid54321:my-object:my-function:8*

am besten nicht vergessen

*pid54321:libc.so.1:strcpy:entry*

*pid54321:libc.so:strcpy:entry*

*pid54321:libc:strcpy:entry*

- die Probes werden erst bei Bedarf generiert und tauchen daher bei der Ausgabe von "*dtrace -l*" nicht auf
- **Tipp:** Kommandozeilen Optionen "-p" und "-c"

# Beispiel: Aufruf-Stacks

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

```
/* uses the 'pid' Provider to print call stacks */
```

```
#pragma D option quiet
```

```
pid$target:$1:$2:entry
```

```
{
```

```
    printf("%s:%s:%s", probeprov, probemod, probefunc);
```

```
    ustack();
```

```
}
```



\$1, \$2: Kommandozeilen Argumente  
\$target: PID des Prozesses

# Beispiel: Aufruf-Stacks

```
obi-wan# ./lib_call_stack.d -c ls 'libc' 'str*'
```

```
lib_call_stack.d
```

“ls” Ausgabe

```
pid1002:libc.so.1:strcmp
    libc.so.1`strcmp
    libc.so.1`setlocale+0x1378
    ls`main+0x22
    ls`_start+0x7a
pid1002:libc.so.1:strlen
    libc.so.1`strlen
    ls`xstrdup+0x12
    ls`main+0x486
    ls`_start+0x7a
pid1002:libc.so.1:strlen
    libc.so.1`strlen
    ls`xstrdup+0x12
```

# Beispiel: libc Trace

```
obi-wan# ./libc_trace.d -F -c date
```

```
dtrace: script './libc_trace.d' matched 5789 Probes  
Sunday, November 11, 2012 11:50:04 AM CET  
dtrace: pid 16648 has exited
```

```
CPU FUNCTION
```

```
5  -> __tls_static_mods  
5   -> malloc  
5    -> getbucketnum  
5     <- getbucketnum  
5    -> initial_allocation  
5     -> __syscall  
5     <- __syscall  
5     <- initial_allocation  
5    <- malloc  
...
```

“date” Ausgabe

# Aus dem wahren (vergangenen) Leben

## Beispiel: Active Directory Controller

- wir verwendeten xVM (Xen) um auf Basis von Solaris Windows Server 2008 zu virtualisieren
- Platten IO ist auf Grund der para-virtualisierten Treiber im Allgemeinen sehr gut
- Problem: nachdem ein performantes System die Funktion eines AD Controllers übernahm brach die Leistung massiv ein



## Beispiel: Active Directory Controller

- "vmstat 5" liefert Hinweise auf eine hohe system-call Rate
- "prstat -Lm" zeigt welcher Prozess es ist

PID	NAME	USR	SYS	TRP	TFL	DFL	LCK	SLP	LAT	VCX	ICX	SCL	SIG	PROC/NLWP
16480	xvm	6.9	9.8	0.0	0.0	0.0	27	54	1.8	30K	114	.2M	0	qemu-dm/3
363	xvm	0.1	0.2	0.0	0.0	0.0	0.0	100	0.0	4	1	2K	0	xenstored/1
16374	root	0.0	0.1	0.0	0.0	0.0	100	0.0	0.0	10	0	1K	0	dtrace/1
1644	xvm	0.1	0.1	0.0	0.0	0.0	33	66	0.0	569	7	835	0	qemu-dm/3
2399	root	0.0	0.1	0.0	0.0	0.0	0.0	100	0.0	49	0	388	0	sshd/1
16376	root	0.0	0.1	0.0	0.0	0.0	0.0	100	0.0	38	0	297	0	prstat/1
11705	xvm	0.0	0.1	0.0	0.0	0.0	50	50	0.0	576	15	858	0	qemu-dm/4
16536	root	0.0	0.1	0.0	0.0	0.0	0.0	100	0.0	48	0	286	0	vncviewer/1
...														

# Beispiel: Active Directory Controller

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

/* get some statistic about system call rates */

#pragma D option quiet

BEGIN {
    timer = timestamp;          /* nanosecond timestamp */
}
syscall::entry {
    @c[pid, execname, probefunc] = count();
}
tick-5s {
    trunc(@c, 10);
    normalize(@c, (timestamp-timer) /1000000000);
    printa("%5d %-20s %6@d %s\n", @c);
    clear(@c);
    printf("\n");
    timer = timestamp;
}
```

# Beispiel: Active Directory Controller

```
leto# ./count_syscalls.d
```

209	nscd	27	xstat
16376	prstat	35	pread
16480	qemu-dm	117	pollsys
16480	qemu-dm	123	read
16480	qemu-dm	136	ioctl
11705	qemu-dm	145	pollsys
1644	qemu-dm	151	pollsys
16374	dtrace	331	ioctl
16480	qemu-dm	35512	lseek
16480	qemu-dm	35607	write

# Beispiel: Active Directory Controller

```
lcto# ./seek-write-stat.d
```

lseek	fdesc	delta	count
	5	26	28
	5	29	28
	5	0	42
	5	21	42
	5	1	134540

write	fdesc	size	count
	5	21	42
	15	4	54
	16	4	63
	14	4	441
	5	1	134554

## Beispiel: Active Directory Controller

- Überprüfung von file-descriptor #5 mit Hilfe von *pfiles(1)* liefert Hinweise darauf, das es sich um Zugriffe auf die virtuelle Platte handeln muss

```
leto# pfiles 16480
```

```
...  
    5: S_IFREG mode:0600 dev:182,65543 ino:26 uid:60  
gid:0 size:11623923712  
    O_RDWR|O_LARGEFILE  
    /xvm/hermia/disk_c/vdisk.vmdk  
...
```

# Beispiel: Active Directory Controller

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

/* print call stack statistics for "qemu-dm"
 * for the lseek() and write() system calls
 */

#pragma D option quiet

syscall::lseek:entry, syscall::write:entry
/ execname == "qemu-dm" /
{
    @c[probfunc, ustack()] = count();
}

tick-5s {
    trunc(@c, 3);
    printa(@c);
    clear(@c);
}
```

# Beispiel: Active Directory Controller

...

write

```
libc.so.1`__write+0xa  
qemu-dm`RTFileWrite+0x37  
qemu-dm`RTFileWriteAt+0x48  
qemu-dm`vmdkWriteDescriptor+0x1d5  
qemu-dm`vmdkFlushImage+0x23  
qemu-dm`vmdkFlush+0x9  
qemu-dm`VDFlush+0x91  
qemu-dm`vdisk_flush+0x1c  
qemu-dm`bdrv_flush+0x2e  
qemu-dm`ide_write_dma_cb+0x187  
qemu-dm`bdrv_aio_bh_cb+0x16  
qemu-dm`qemu_bh_poll+0x2d  
qemu-dm`main_loop_wait+0x22c  
qemu-dm`main_loop+0x7a  
qemu-dm`main+0x1886  
qemu-dm`_start+0x6c  
28758
```

## Beispiel: Active Directory Controller

- auf Grund des call-stacks liegt die Vermutung nahe, dass
  - Windows den Platten Cache leert
  - dieser deaktiviert ist
- letzteres war der Fall
- eine Aktivierung in einem Windows Start-Skript beseitigt die IO Probleme
  - über eine Quantisierung der IO Größe bei Schreibzugriffen lässt sich dies mit DTrace verifizieren



# Beispiel: Active Directory Controller

```
lcto# ./qemu-stat-quant.d 5
```

```
write
value  ----- Distribution ----- count
  256  |                                           0
  512  | @@@@@@                                  14
 1024  |                                           0
 2048  | @@@@                                    10
 4096  | @@@@@@@@@@@@@@@@@@@@@@@@@@@@          39
 8192  | @@@@@@@@@@@@@@@@@@@@@@@@@@@@          36
16384  |                                           0
```

# Literatur

- hervorragende Informationsquelle zumal auch alle neuen und nur in Solaris 11 verfügbaren Provider dokumentiert sind

*<http://wikis.sun.com/display/DTrace>*

- das DTrace Manual (PDF)

*[http://docs.oracle.com/cd/E18752\\_01/pdf/817-6223.pdf](http://docs.oracle.com/cd/E18752_01/pdf/817-6223.pdf)*

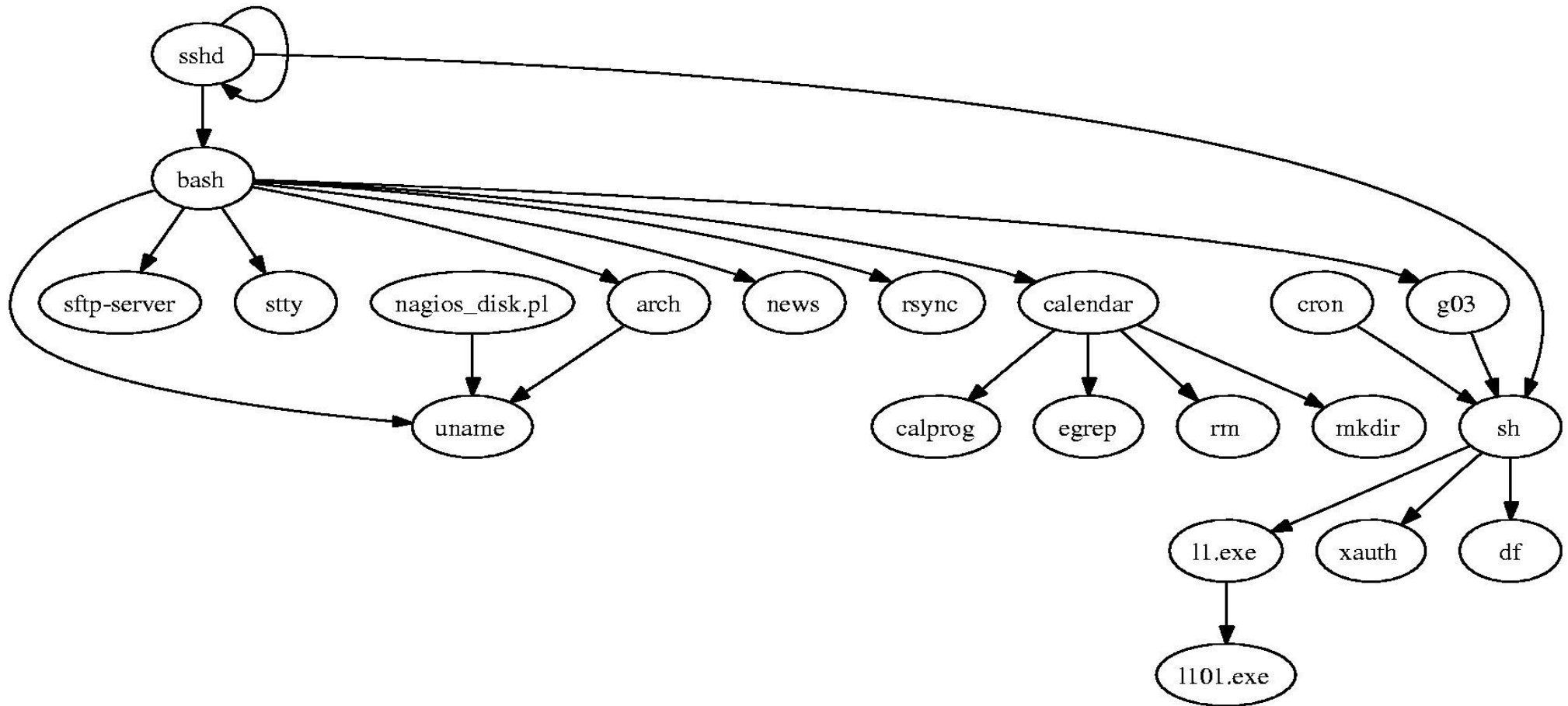
- Beispiele finden sich unter

*<https://wikis.oracle.com/display/DTrace/DTrace>*

*<http://www.brendangregg.com/dtrace.html>*

*</usr/demo/dtrace>*

# Pimp my DTrace: *proc* provider plus Graphviz



# Danke für's Zuhören!