

DTrace

...er awesome!

Carl Martin Rosenberg

MAPS

25.02.2015



Først litt reklame...

- MAPS arrangerer programmeringskonkurranse 19. mars.
- Formatet er inspirert av *Project Euler*. Finn riktig tekststreng!
- Hvordan du kommer frem til svaret (programmeringsspråk o.l.) er opp til deg.
- Det blir oppgaver for enhver smak.
- Bli med!



Velkommen!

- Dette er altså et foredrag om DTrace
- Merk: Jeg er ikke DTrace-ekspert, men jeg tror jeg vet nok til å overbevise deg om at DTrace er verdt å sjekke ut!



Velkommen!

- DTrace er en skikkelig heftig teknologi (bare vent!), men vanskelig å forstå uten å først forstå hvilke problemer det er laget for å løse.
- La oss derfor først ta en titt på hvordan software-verdenen ser ut i dag...



Moderne software-løsninger er *systemer*

- Moderne software-løsninger er *systemer*, ikke enkeltprogrammer.
- Fantastisk! Man slipper å finne på hjulet på nytt!
- Likevel: Din egen kode er bare en bitteliten del av helheten...
- ...og sannsynligvis den minst komplekse av dem alle.



Når vi utvikler komplekse systemer, trenger vi...

- Vi må kunne analysere hvordan komponentene interagerer med hverandre: *Er det databasen eller webserveren som gjør at ting går treigt?*
- Vi må kunne få en viss innsikt i programmer vi enten
 - 1 ikke selv har kildekoden til, eller
 - 2 ikke selv har tid og ressurser til å analysere fullstendig.



Når vi utvikler komplekse systemer, trenger vi...

- ...men hva med *kjørende* systemer som det er *svært kostbart* å ta ned?
- Det er ikke nok å kunne få innsikt i ting når vi sitter på våre egne utviklingsmaskiner.



Vi *må* kunne få innsikt i det kjørende systemet

- Vi må raskt kunne få innsikt i det kjørende systemet. . .
- . . . uten at selve *observasjonen av systemet* påvirker **sikkerhet** eller **ytelse**.



DTrace to the rescue!

- DTrace er laget fra bunn av for å løse disse problemene!



- DTrace ble lansert av Sun Microsystems i 2005.
- Beviste for verden at dynamisk tracing kan gjøres på en sikker måte.
- Finnes på Mac OS X, FreeBSD, NetBSD og de fleste OpenSolaris-forks (Illumos etc.).
- DTrace er *åpen kildekode* men anses dessverre ikke for å være GPL-kompatibel. Dette betyr dessverre at Linux-støtten er begrenset.
- Linux har likevel verktøy som kan gi deg det aller meste av DTrace's funksjonalitet.



Nøkkelideen: Dynamisk Tracing

- Tidligere teknologier baserte seg utelukkende på *statisk* tracing.
- Statisk tracing kunne bare brukes til å analysere de problemene man antok at var viktige når man lagde programmet.
- Dette gjorde det ofte fristende å skape et skille mellom kode-under-utvikling og kode-i-produksjon



Nøkkelideen: Dynamisk Tracing

- DTrace introduserer *dynamisk* tracing: DTrace kan observere systemet *fortløende* ved å endre cpu-instruksjonene som ligger i minnet!



Nøkkelideen: Dynamisk Tracing

- Dette betyr blant annet:

- 1 Programmer må ikke lages observerbare for å kunne observeres: “Alt” som kjører på systemet kan observeres (bare vi er systematiske nok. . .)
- 2 Programmer trenger ikke stoppes eller startes på nytt for å bli instrumentert (Cantrill 04)
- 3 *Zero overhead when not in use*: De ekstra instruksjonene som brukes for å observere systemet legges bare til når observasjonen gjøres. Systemet er ellers uendret.

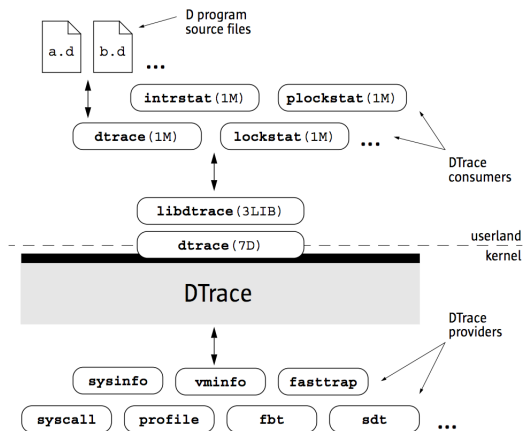


La oss ta en titt!

- DTrace gir oss en observasjonspunkter for hele systemet: Fra alle kjørende userland-prosesser til ting i kjernen (scheduleren, låser mm.).
- Hvert observasjonspunkt representerer en *hendelse*, for eksempel hendelsen “en prosess kaller `malloc`”.
- Vi bruker DTrace ved å knytte handlinger til disse hendelsene for å observere systemet.



Det store bildet 1



Fra *Solaris Dynamic Tracing Guide*, side 33



Observasjonspunktene

- Et observasjonspunkt kalles en *probe*.
- Hvert observasjonspunkt identifiseres ved et unikt fire-tupple:
`<provider:module:function:name>`
- La oss gå dette tuppelet nærmere i sømmene...



<**provider**:module:function:name>

- Providere er kjernemoduler som henter inn data på vegne av DTrace.
- Providere oppretter instrumenteringspunktene dynamisk.
- Providerne iverter sikkerheten: de bestemmer hva som er trygt og hva som er utrygt å instrumentere. De andre komponentene i DTrace er “konsumenter” av dataene fra providerne.
- Systemet har providere delt inn etter tema.



Providers: De vanligste

- 1 **syscall**: Gir innsikt i systemkallene som operativsystemkjernen mottar.
- 2 **fbt** (function boundary tracing): Gir instrumenteringspunkter for funksjoner i operativsystemkjernen.
- 3 **proc**: Holder styr på prosessrelaterte hendelser.
- 4 **sched**: Observerer scheduleren i operativsystemkjernen.
- 5 **io**: Disk I/O.
- 6 **tcp**: Hendelser i TCP-laget.
- 7 **ip**: Hendelser i IP-stacken, f.eks send og receive.
- 8 **pid**: Gir en egen provider for hver prosess som kjører (en slags meta-provider).



Providers i userland

- I tillegg til disse kan applikasjonsutviklere lage “provider” for sine egne programmer, som eksponerer instrumenteringspunkter:
 - 1 De fleste programmeringsspråk har en tilhørende provider
 - 2 Masse inflytelsesrike programmer kan kompileres til å eksponere en DTrace-provider: PostgreSQL, Node.js, Apache mm.
 - 3 Du kan lage dine egne (statiske) instrumenteringspunkter og være en provider for ditt eget program!



<provider:**module**:function:name>

- Hvis instrumenteringspunktet ligger i en kjernemodul, vil navnet på denne kjernemodulen være module-delen av tuppelet
 - På FreeBSD med ZFS-filsystemet kan module være ZFS.



Moduler (forts.)

- Hvis instrumenteringspunktet ligger i userland og er knyttet til et delt bibliotek, vil dette stå i module-feltet.
- *Eksempel:* På mitt system er python-instrumenteringspunktene knyttet til python-modulen.
- De fleste instrumenteringspunkter er ikke knyttet til en bestemt modul. I så fall kan vi sette en * (wildcard) i module-feltet eller bare la det stå tomt.



`<provider:module: function:name>`

- Hvis instrumenteringspunktet er knyttet til et faktisk funksjonssymbol (for eksempel `malloc`), vil dette stå på funksjonsplassen.
- I likhet med module er det ikke nødvendigvis slik at alle instrumenteringspunkter har et funksjonsfelt.
- Dersom et instrumenteringspunkt er knyttet til en modul og en funksjon, kaller vi dette en “anchored probe”. Hvis ikke er den “unanchored” (Solaris Dynamic Tracing Guide).

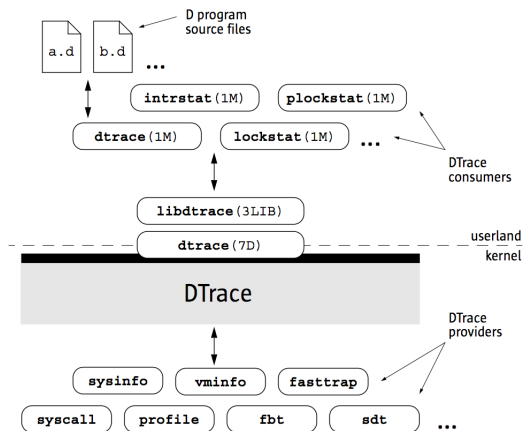


`<provider:module:function:name>`

- Name er et felt som skal formidle hva instrumenteringspunktet er til.
- Name vil ofte være enten BEGIN eller END, korresponderende til henholdsvis når man går inn eller ut av en gitt funksjon.
- Name kan også være mer domenespesifikt, for eksempel acquire, spin og block for en bestemt lås.



Det store bildet 2



Fra *Solaris Dynamic Tracing Guide*, side 33



- 1 Hvis vi bare sier hvilke instrumenteringspunkter vi er interessert i og ikke oppgir en action-block, listes bare de instrumenteringspunktene som fyres.
- 2 Så hva kan vi egentlig gjøre i action-blokken? Ganske mye!



Actions 101

- 1 Det mest åpenbare vi kan gjøre er å printe noe til skjermen når en hendelse inntreffer.
- 2 Dette kan vi gjøre mer sofistikert, med å bruke innebygde variabler, som `execname`, `pid`, `timestamp` osv.
- 3 Med disse enkle byggestenene kan vi til og med inspisere hvilke argumenter funksjoner blir kalt med!



Actions: Innebygde funksjoner

- 1 Vi kan gå enda lenger...
- 2 La oss finne ut filnavn som programmene på systemet forsøker å åpne med systemkallet `open`!
- 3 Kult, men blir det ikke veldig mye data?
- 4 Vi har to verktøy for å filtrere og fortolke datamengden:
aggregeringer og predikater.



Actions: Aggregeringer

- Vi kan bruke aggregeringer for å få mere konsise statistikker.
- Det finnes adskillig mer sofistikerte aggregeringer i `count()`, men det rekker jeg ikke gå inn på.



- Vi kan bruke predikater til å filtrere *før* vi gjør actions.
- Sammenligningene fungerer som i C.



Hvordan det føles



Hvor er groove-spaken?








- Metode er *essensielt*.
- Vi må definere problemene våre på en måte som kan *måles*.
- DTrace er et ekspertverktøy. . .
- . . . men kan også hjelpe oss til å *bli* eksperter!



Hvis du vil vite mer...

- Tips, triks, ting å lese, ting å se:
<https://github.com/cmrosenberg/>



-  Bryan Cantrill. “Hidden in Plain Sight”. In: *Queue* 4.1 (Feb. 2006), pp. 26–36. ISSN: 1542-7730. DOI: 10.1145/1117389.1117401. URL: <http://doi.acm.org/10.1145/1117389.1117401>.
-  Bryan Cantrill, Michael W Shapiro, Adam H Leventhal, et al. “Dynamic Instrumentation of Production Systems.” In: *USENIX Annual Technical Conference, General Track*. 2004.
-  Oracle Corporation. *DTrace Guide for Oracle Solaris 11*. Oracle Corporation. 2012. URL: http://docs.oracle.com/cd/E23824_01/pdf/E22973.pdf.
-  Brendan Gregg. *Systems Performance: Enterprise and the Cloud*. Pearson Education, 2013.
-  Brendan Gregg and Jim Mauro. *DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X, and FreeBSD*. Prentice Hall Professional, 2011.

Spørsmål?

- Spørsmål?



Takk for meg!

- Takk for meg!

