UOC
Universitat Oberta de Catalunya
www.uoc.edu

Treball de Fi de Grau

Estudi eines de traça: ltrace, strace i DTrace

Alumne: Pep Rincón Qués
Consultor: Francesc Guim Bernat
Grau d'Enginyeria Informàtica

Aquesta obra està subjecte a la llicència de Reconeixement-NoComercial-CompartirIgual 3.0 Espanya de Creative Commons. Si voleu veure una còpia d'aquesta llicència accediu a https://creativecommons.org/licenses/by-nc-sa/3.0/es/deed.ca o envieu una carta sol·licitant-la a Creative Commons, PO Box 1866, Mountain View, CA 94042, USA

Resum

Aquest treball tracta sobre el estudi de les eines de traça, s'explicarà la seva utilitat i la seva importància per l'optimització i depuració de programes per millorar el rendiment.

Donat la quantitat d'eines existents es delimitarà el estudi a tres eines. Les eines triades son strace, ltrace i DTrace. S'explicarà el seu funcionament, el seu ús i com s'empren mostrant-ho amb exemples. També s'anomenaran avantatges i desavantatges que tenen.

A més s'anomenaran altres eines de traça i es comentaran molt per damunt per veure la situació actual.

Paraules clau: Eines de trace, strace, ltrace, DTrace.

Abstract

This paper deals with the study of trace tools (tracers), Explain their usefulness and their importance for optimizing and debugging software to get better performance.

As there are many tools the study will focus on three tools. The chosen tools are strace, ltrace and DTrace. Their operation, use and how they run will be explained with examples.

Moreover other tracers will be named and explained briefly to know ttheir present situation.

Keyword:

Tracer tools, tracers, strace, ltrace, Dtrace.

Índex

0. Agraïments....................3
1. Introducció....................4
2. Motivació....................5
3. Descripció del projecte....................6
3.1 Objectius i Resultats....................6
3.2 Anàlisi de riscos....................6
4. Abast de la proposta....................7
5. Organització del projecte....................7
6. Pla de treball....................8
6.1 Relació d'activitats....................8
6.2 Fites principals....................9
6.3 Calendari de treball....................9
7. Valoració Econòmica....................11
8. Conceptes previs:....................12
9 Strace....................13
9.1 Què fa?....................13
9.2 Com ho fa?....................13
9.3 Ús....................14
9.4 Avantatges....................14
9.5 Inconvenients....................14
9.6 Exemples d'us de strace....................15
Exemple 1....................15
Exemple 2....................17
Exemple 3....................19
Exemple 4....................19
Exemple 5....................19
Exemple 6....................22
10. Ltrace....................23
10.1 Què fa?....................23
10.2 Com ho fa?....................23
10.3 Ús....................25
10.4 Avantatges....................25
10.5 Inconvenients....................26
10.6 Exemples d'ús de Itrace....................26
Exemple 7....................26
Exemple 8....................27
Exemple 9....................28
Exemple 10....................29
Exemple 11....................31
Exemple 12....................32
Exemple 13....................34
Exemple 14....................34
Exemple 15....................35
11. DTrace....................36
11.1 Què fa?....................36
11.2 Com ho fa?....................36

11.3 Ús....39
11.4 Avantatges....39
11.5 Inconvenients....40
11.6 Exemples d'us de DTrace....40
Exemple 16....41
Exemple 17....41
Exemple 18....42
Exemple 19....43
Exemple 20....44
Exemple 21....45
Exemple 22....45
Exemple 23....46
Exemple 24....46
Exemple 25....47
Exemple 26....48
11.6.1 Scripts existents (DTrace Toolkit)....50
Exemple 27....50
Exemple 28....50
Exemple 29....51
Exemple 30....51
12. Alternatives....52
13. Conclusions....53
13.1 Comparativa....53
13.2 Opinió Personal....54
14. Bibliografia....55
14.1 Strace:....55
14.2 Itrace:....55
14.3 Dtrace:....56
14.4 Altres:....56

0. Agraïments

A la meva dona, per animar-me a estudiar, suportar-me i estimar-me. T'estim.

Als meus sogres per animar-me, al meu sogre per fer-me la competència estudiant.

A la meva mare, per donar-me suport i finançar-me encara que no entengui que es pot treure una carrera a distància.

Al meu pare, al cel sia, per inculcar-me la curiositat per com funciona els que ens envolta i en especial, tot el que tengui electricitat.

A na Henry, en Jack i na Buttercup per donar-me confort i llepar-me quan estava atabalat

1. Introducció

Dins l'àmbit de l'enginyeria informàtica la optimització dels recursos sempre es molt important i valuosa. Ja que una gran potència de computació ha de ser ben emprada per evitar el malgast de recursos.

Aquesta activitat requereix de coneixements profunds del sistema operatiu que es tracta d'optimitzar, a causa d' això es conta amb aplicacions per saber que està fent exactament el ordenador durant un cert procés, quines llibreries obri, quins senyals envia, quin fitxers obri i el temps que tarda en realitzar aquestes operacions entre altres.

Aquest tipus d'aplicacions serveixen també per ajudar a depurar errors de programació, a aquestes eines es diuen eines de traça.

Aquest treball tracta de fer l'estudi comparatiu de diferents aplicacions que existeixen als sistemes operatius, al haver moltes possibilitats ens limitarem a les aplicacions strace, ltrace, i Dtrace.

- Dtrace és una eina de monitoratge desenvolupada per Sun Microsystems per a Solaris que permet un gran control sobre tot el que succeeix en el sistema, amb un llenguatje propi per realitzar les consultes i anàlisis.

- Strace es una comanda de terminal util per diagnosticar errors. El que fa és interceptar i enregistrar les criddes de sistema (system calls) que son cridades per un procés i els senyals rebuts per aquests.

- ltrace es una comanda semblant a strace però en aquest cas intercepta i registre les crides a funcions situades a llibreries dinàmiques realitzades per un procés, així com els retorns rebuts per aquest.

2. Motivació

La motivació d'aquest projecte es analitzar varies aplicacions emprades per el seguiment i monitorització dels processos en execució i veure el seu estat actual. Es nombraran també algunes alternatives ja que n'hi ha diverses.

En aquest treball de caire acadèmic es tracten principalment tres aplicacions on es veu com han anat evolucionant les eines de traça, es limita a tres per l'extensió del temps disponible i s'han triat strace, ltrace i Dtrace per ser eines de traça remarcables.

S'ha decidit per aquest tema, donada la importància que té actualment la optimització de recursos, especialment a entorns de servidors situats al núvol. Els quals poden estar destinats a donar i vendre serveis distribuïts com computació distribuïda con EC2 de Amazon, o serveis de contenidors com Docker o servidors privats virtuals.

Aquest tipus de serveis permeten que les empreses que els donen els facturin al client segons la potencia de comput, el espai de disc o el ampla de banda emprat de manera que el client pagui el que empra. Per això, l'optimització i la monitorització de recursos es molt important en aquests moment tant per l'empresa que dona el servei com per el client.

A demés, per l'escala de tractament de dades que hi ha actualment, ja sigui per la quantitat de clients que hi hagi per un servei de video baix demanda com Netflix o Youtube, per la quantitat que suposen les dades científiques a un sistema de alta computació distribuïda (HPC) o pels milions de cerques que es fan a Google esbrinar on perd temps una aplicació suposa un estalvi important.

L'imparable migració cap aquests serveis es veu reflectida a la industria, on s'estan implementant centres de dades on en contes de tenir una maquina completa per a cada client, es va passar a maquines virtuals dins una sola maquina i ara es veuen dissenys com el Intel Scale Design. En dit servei es gestionen maquines virtuals escalables a partir de mòduls que poden ser només un parell de discos, o un parell de GPUs o una grapada de CPUs.

Un altra de les motivacions, és veure l'estat actual de aquest tipus d'aplicacions a més de possibles diferències en diferents implementacions, ja que son aplicacions que poden arribar a ser molt dependents del sistema operatiu a causa de la seva interacció amb el nucli de sistema.

3. Descripció del projecte.

Amb aquest projecte es vol mostrar i analitzar les tres aplicacions esmentades anteriorment.

Per tant es farà un anàlisis de com s'empren, com funcionen i quins tipus resultats s'obtenen, així com la utilitat d'aquests.

També es dirà quins avantatges i desavantatges te cada una de les aplicacions analitzades.

3.1 Objectius i Resultats

A continuació s'enumeraran els principals objectius d'aquest projecte:

- Introduir que són les eines de traça i la seva utilitat.
- Analitzar el funcionament l'arquitectura de les eines de traça strace, ltrace i DTrace.
- Mostrar avantatges i desavantatges de les eines de traça estudiades.
- Mostrar exemples de com s'empren aquestes eines de traça.
- Estudiar les diferències que hi ha entre les diferents implementacions de aquestes eines.
- Estudiar el estat actual d'aquests tipus d' eines.

3.2 Anàlisi de riscos

Els riscos que es poden trobar principalment poden ser:

Temporals i per tant econòmics pel que s'anirà comprovant durant el temps si s'està complint la planificació.

Tècnics a causa de les implementacions que es puguin trobar de les diferents eines, especialment a versions diferents del nuclis de sistema operatiu amb el qual estan molt lligats per tant es disposarà de maquines virtuals de cara instal-lar sistemes operatius alternatius si és necessari.

4. Abast de la proposta

Es limitarà al estudi de les eines

- strace, ltrace i DTrace

Especialment pel temps del que es disposa per a realitzar el treball.

De strace i ltrace s'explicarà per damunt el seu funcionament intern. Als seus exemples s'explicaran tant les opcions emprades com la sortida obtinguda.

De DTrace s'explicarà els components que la formen i el necessari per poder empar-ho ja que es un entorn de treball i és diferent a les dues eines anteriors.

5. Organització del projecte

Aquí es mostren els recursos que cal assignar a aquest treball:

Recursos Humans:

Tota la feina la fa el mateix autor de totes maneres es defineixen varis rols:

Tècnic de sistemes [TecSys} , Enginyer de Maquinari [Prgm] , Gestor de projecte [GesPro] , Gestor de qualitat [GesQ]. Encara que cada tasca només la fa amb un rol.

Recursos temporals:

Segons la UOC el Treball de fi Grau són 12 crèdits ECTS i un 1 crèdit equival 25 hores, per tant el temps total destinat a aquest treball són 300 hores teòriques.

Recursos materials:

Documentals: la documentació consultada serà adjuntada a la bibliografia incloent llibres i planes web

Tecnològics: S'emprarà l'ordinador del estudiant ja que es suficient per la feina, tots els recursos emprats com ara aplicacions o imatges de màquines virtuals estan disponibles a la xarxa de manera gratuïta aquests inclouen:

- La distribució Archlinux que empra l'autor normalment, i els paquets strace i ltrace son codi obert dins llicència GPL disponible al gestor de paquets d'aquest sistema.

- Les imatges de màquina virtual per Virtualbox Kubuntu per proves ja instal·lada i les imatges de FreeBSD de la seva plana oficial i Oracle Solaris amb el DTrace també de la plana de Oracle.

6. Pla de treball

6.1 Relació d'activitats

Aquestes son les tasques que formen part del projecte.

1. Cerca d'Informació d'eines de traça. 4 dies

2. Documentar els tipus d'eines de traça existents 2 dies

3. Strace

3.1. Cerca i estudi d'informació d'eina strace 4 dies

3.2. Documentar funcionament d'eina strace 2 dia

3.3. Preparar sistema perquè funcioni eina strace 1 dia

3.4. Dur a terme exemples d'execució d'eina strace 4 dies

3.5. Documentar proves 3 dies

4. ltrace

4.1. Cerca i estudi d'informació d'eina ltrace. 4 dies

4.2. Documentar funcionament d'eina ltrace. 2 dies

4.3. Preparar sistema perquè funcioni eina ltrace. 1 dia

4.4. Dur a terme exemples d'execució d'eina ltrace. 4 dies

4.5. Documentar proves. 3 dies

5. DTrace

5.1. Cerca i estudi d'informació d'eina DTrace. 7 dies

5.2. Documentar funcionament d'eina DTrace. 3 dies

5.3. Preparar sistema perquè funcioni eina DTrace. 2 dies

5.4. Dur a terme exemples d'execució d'eina Dtrace. 10 dies

5.5. Documentar proves. 4 dies

6. Realitzar memòria del projecte. 10 dies

7. Repas i adequació. 1 dia

8. Vist i plau. 1 dia

6.2 Fites principals

Aquestes són les fites principals:

- F1: Fita Strace al acabar de fer feina amb strace
- F2: Fita ltrace al acabar de fer feina amb ltrace
- F3: Fita Dtrace al acabar de fer feina amb DTrace

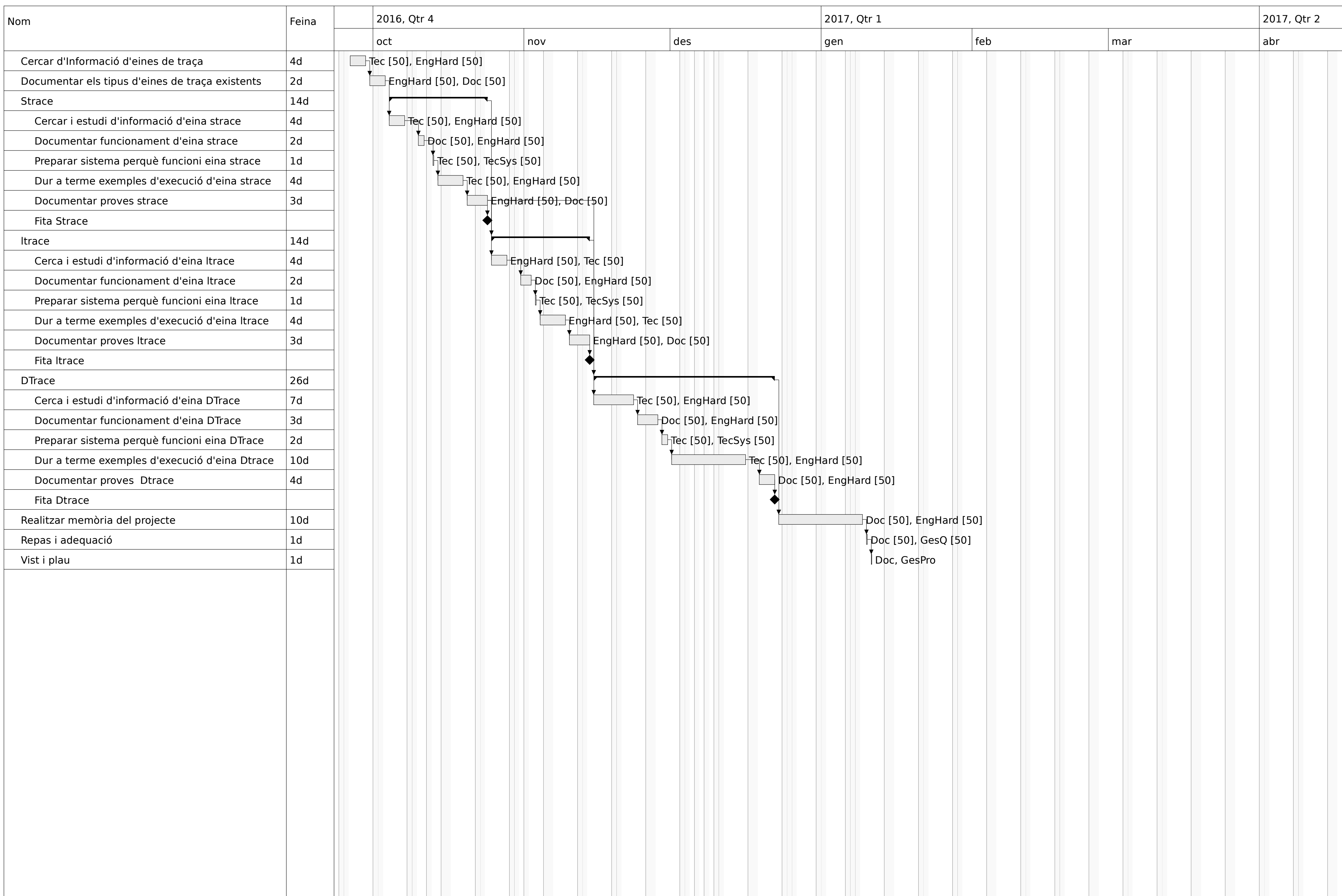
6.3 Calendari de treball

Suposarem inici de semestre 26/09/2016 (dilluns) i final de semestre com 21/01/2017 darrer dia d'examens) 17 setmanes.

Si es lleven festius (12/10, 01/11, 06/12, 08/12, 26/12 de 2016, dissabtes,diumenges, 06/01 de 2017 queden uns 79 dies. 79 dies per 4h = 316h però es respectaran les 300h.

Sobre el projecte s'ha calculat unes 296 hores que son 74 dies.

A la pàgina següent s'inclou el diagrama de Gantt.


Nom
Feina
2016, Qtr 4
2017, Qtr 1
2017, Qtr 2
oct
nov
des
gen
feb
mar
abr
Cercar d'Informació d'eines de traça
4d
Tec [50], EngHard [50]
Documentar els tipus d'eines de traça existents
2d
EngHard [50], Doc [50]
Strace
14d
Cercar i estudi d'informació d'eina strace
4d
Tec [50], EngHard [50]
Documentar funcionament d'eina strace
2d
Doc [50], EngHard [50]
Preparar sistema perquè funcioni eina strace
1d
Tec [50], TecSys [50]
Dur a terme exemples d'execució d'eina strace
4d
Tec [50], EngHard [50]
Documentar proves strace
3d
EngHard [50], Doc [50]
Fita Strace
Itrace
14d
Cerca i estudi d'informació d'eina Itrace
4d
EngHard [50], Tec [50]
Documentar funcionament d'eina Itrace
2d
Doc [50], EngHard [50]
Preparar sistema perquè funcioni eina Itrace
1d
Tec [50], TecSys [50]
Dur a terme exemples d'execució d'eina Itrace
4d
EngHard [50], Tec [50]
Documentar proves Itrace
3d
EngHard [50], Doc [50]
Fita Itrace
DTrace
26d
Cerca i estudi d'informació d'eina DTrace
7d
Tec [50], EngHard [50]
Documentar funcionament d'eina DTrace
3d
Doc [50], EngHard [50]
Preparar sistema perquè funcioni eina DTrace
2d
Tec [50], TecSys [50]
Dur a terme exemples d'execució d'eina Dtrace
10d
Tec [50], EngHard [50]
Documentar proves Dtrace
4d
Doc [50], EngHard [50]
Fita Dtrace
Realitzar memòria del projecte
10d
Doc [50], EngHard [50]
Repas i adequació
1d
Doc [50], GesQ [50]
Vist i plau
1d
Doc, GesPro

7. Valoració Econòmica

A continuació s'adjunta la taula de costos dels recursos:

Taula 1 costos dels recursos:

NOM RECURS	TIPUS	CODI	TASA (€/Hora)	HORES DEDICADES
Enginyer de Maquinari	Feina	EngHard	50,00 €	264
Gerent Projecte	Feina	GesPro	50,00 €	4
Gestor Qualitat	Feina	GesQ	40,00 €	4
Tècnic de sistemes	Feina	TecSys	20,00 €	16
Documental	Material	Doc	2,00 €	124
Tecnològics	Material	Tec	1,00 €	164

Taula 2. Costos de les activitats:

ID	Tasca	Dies	Hores	Cost inc Materials
1	Cercar d'Informació d'eines de traça	4	16	816,00 €
2	Documentar els tipus d'eines de traça existents	2	8	416,00 €
3	Strace			
3.1	Cercar i estudi d'informació d'eina strace	4	16	816,00 €
	Documentar funcionament d'eina strace	2	8	416,00 €
3.3	Preparar sistema perquè funcioni eina strace	1	4	85,00 €
3.4	Dur a terme exemples d'execució d'eina strace	4	16	816,00 €
3.5	Documentar proves strace	3	12	624,00 €
4	Itrace			
4.1	Cerca i estudi d'informació d'eina Itrace	4	16	816,00 €
4.2	Documentar funcionament d'eina Itrace	2	8	416,00 €
4.3	Preparar sistema perquè funcioni eina Itrace	1	4	85,00 €
4.4	Dur a terme exemples d'execució d'eina Itrace	4	16	816,00 €
4.5	Documentar proves Itrace	3	12	624,00 €
5	DTrace			
5.1	Cerca i estudi d'informació d'eina DTrace	7	28	1.428,00 €
5.2	Documentar funcionament d'eina DTrace	3	12	624,00 €
5.3	Preparar sistema perquè funcioni eina DTrace	2	8	168,00 €
5.4	Dur a terme exemples d'execució d'eina Dtrace	10	40	2.040,00 €
5.5	Documentar proves Dtrace	4	16	832,00 €
6	Realitzar memòria del projecte	10	40	2.080,00 €
7	Repas i adequació	1	4	168,00 €
8	Vist i plau	1	4	208,00 €
			TOTAL:	**14.294,00 €**

8. Conceptes previs:

Alguns conceptes previs per a que quedin explicats:

- **Depurador (Debugger)**: Programa que ajuda a detectar els possibles errors en un programa, mentre s'està executant, perquè permet de veure contínuament el contingut de totes les seves estructures de dades. Son més efectius quan tenim el codi font del procés perquè ens poden indicar a quina línia de codi esta fallant, el pot emprar per posar punts de ruptura(breakpoints) on e interessi començar a executar pas a pas un procés per trobar el seu error o mirar quin resultat dona, també es pot modificar dades.

- Eines de traça (tracers): Son eines que permeten fer un seguiment del que està fent un procés en funcionament mitjançant l'observació d'aquest. Perusar-les no fa falta tenir el codi font del procés observar, els resultats poden ser mostrats en pantalla o enregistrats a arxius de text o diaris(logs). Son el tipus d'eines que emprarem en aquest estudi. Es solen emprar a més de per depurar per optimitzar i saber on un procés es queda penjat i on es perd temps per estudiar el rendiment.

- Espai d'usuari: espai de la memòria on fan feina les aplicacions d'usuari, quan es fa feina en aquest espai s'està en mode usuari.

- Espai de nucli: espai de la memòria reservat només per a que hi funcioni el nucli. extensions i alguns controladors, quan es fa feina em aquest espai s'esta en mode d'execució protegit o privilegiat.

- Canvi de context: En aquest treball en el canvi que es fa quan es pasa de mode privilegiat a mode usuari o a la inversa.

- Crida de sistema (System call): Una crida a sistema no és més que una rutina amb una sèrie de paràmetres d'entrada, una sèrie de paràmetres de sortida i una semàntica associada, permeten fer peticions directes de recursos al sistema operatiu. Aquestes crides ofereixen les funcions bàsiques per a poder utilitzar tots els recursos del sistema de manera correcta i controlada. El repertori de crides al sistema constitueix una API del sistema operatiu. s'executen en mode d'execució privilegiat.

- **Instrumentació**: Dins el context de la informàtica la instrumentació es el fet de monitorizar un punt concret de codi de cara a obtenir dades per monitoritzar i mesura el rendiment d'un sistema.

  - Dinàmica: el punt d'instrumentació es pot insertar en temps d'execució per monitoritzar, no es necessari tenir el codi font ni canviar-ho.

  - Estàtica: el punt d'instrumentació s'ha de trobar en forma de codi dins el codi font abans de compilar

9 Strace

9.1 Què fa?

Strace es una comanda que permet observar les crides de sistema realitzades per una aplicació / procés i els senyals que aquestes crides retornen. Permet diagnosticar errors i és pot emprar per monitoritzar processos.

Els administradors de sistemes i testers el trobaran molt útil per solventar problemes a programes dels quals no tenguin el codi font, ja que no és necessari re-compilar-lo per emprar-lo. És suficient amb passar-li el nom del programa o el PID (identificador del procés) per fer-ne el seguiment dels senyals.

Els estudiants, hackers i curiosos el poden emprar per aprendre com funcionen les crides de sistema fent el seguiment de programes de ús quotidià.

Els programadors es trobaran que atès que les crides de sistema i els senyals son activitats que es donen a la interfície entre el mode usuari i el mode privilegiat, poden examinar aquesta frontera per l'aïllament de errors (bugs), comprovar estats i capturar condicions de carrera.

9.2 Com ho fa?

Strace funciona graciés a la crida de sistema *ptrace* la qual permet que un procés observi i controli l'execució d'un altre procés, es una crida específica per depurar. També permet canviar i examinar la memòria i els registres del procés del qual s'està fent el seguiment. A més, pot manipula el senyal del procés que es segueix.

Strace es el procés pare que fa la crida a *ptrace*, al qual se li passa el PID com argument junt amb la bandera PTRACE_ATTACH. Aquest PID és del procés al que volem fer el seguiment, la bandera PTRACE_ATTACH es per convertir de forma temporal el procés a seguir en fill del procés que realitza el seguiment.

Després es fa un altra cridada a *ptrace* però amb arguments bandera PTRACE_SYSCALL i el PID del procés a seguir.

El procés a seguir s'executarà fins que realitzi una crida de sistema, en aquest moment serà aturat per el nucli de Linux que farà una interrupció, crec que arquitectures modernes hi ha una crida de sistema aposta.

El procés seguidor (strace) rebrà un senyal SIGTRAP (aquest senyal es aposta per a depurar) i podrà obtenir la informació que necessiti i imprimir-la o tractar-la.

Després el procés por tornar a fer la crida *ptrace* amb la bandera PTRACE_SYSCALL i el proces a seguir seguirà executant-se normalment fins que torni a passar el mateix.

D'aquesta manera el procés strace pot fer el seguiment d'un altra proces i obtenir informació.

9.3 Ús

El seu ús és com qualsevol altra comanda de Linux, es a dir el nom de la comanda amb una serie d'arguments, un d'ells pot ser la comanda la qual volem fer el seguiment.

```
strace ls
```

Altres arguments poden emprar-se per saber el temps de execució, o fer el seguiment d'un procés que ja està en execució o per saber quantes crides de sistema i de quin tipus ha fet un procés.

9.4 Avantatges

- strace es simple, només tracta crides de sistema i es una comanda POSIX

- La sortida que dona per a cada crida de de sistema es pot entendre fàcilment, no es necessari tractar-la.

- Molt empleat i madur

- Sol funcionar bé amb programes de múltiples fils (threads).

9.5 Inconvenients

- Pot provocar una gran increment del temps de sistema respecte a si no el fem servir, s'ha de recordar que quan es rep el senyal SIGTRAP tot el temps que s'esta tractant la sortida es temps incrementat a l'execució del procés.

- Només es pot fer el seguiment dels processos fills que s'estan seguint

- La visibilitat està limitada a la interfície de crides de sistema

9.6 Exemples d'us de strace

Exemple 1

Com es pot veure si s'executa un strace -h hi ha una bona quantitat de opcions que es poden emprar, si es mira el manual en man strace hi ha moltes més explicacions.

```
buba@Bubarch ~]$ strace -h
usage: strace [-CdffhiqrtttTvVwxxy] [-I n] [-e expr]...
              [-a column] [-o file] [-s strsize] [-P path]...
              -p pid... / [-D] [-E var=val]... [-u username] PROG [ARGS]
   or: strace -c[dfw] [-I n] [-e expr]... [-O overhead] [-S sortby]
              -p pid... / [-D] [-E var=val]... [-u username] PROG [ARGS]

Output format:
  -a column      alignment COLUMN for printing syscall results (default 40)
  -i             print instruction pointer at time of syscall
  -o file        send trace output to FILE instead of stderr
  -q             suppress messages about attaching, detaching, etc.
  -r             print relative timestamp
  -s strsize     limit length of print strings to STRSIZE chars (default 32)
  -t             print absolute timestamp
  -tt            print absolute timestamp with usecs
  -T             print time spent in each syscall
  -x             print non-ascii strings in hex
  -xx            print all strings in hex
  -y             print paths associated with file descriptor arguments
  -yy           print protocol specific information associated with socket file
descriptors

Statistics:
  -c             count time, calls, and errors for each syscall and report
summary
  -C             like -c but also print regular output
  -O overhead    set overhead for tracing syscalls to OVERHEAD usecs
  -S sortby      sort syscall counts by: time, calls, name, nothing (default
time)
  -w             summarise syscall latency (default is system time)

Filtering:
```

```
    -e  expr                  a  qualifying  expression:  option=[!]all  or
option=[!]val1[,val2]...
     options:    trace, abbrev, verbose, raw, signal, read, write, fault
  -P path        trace accesses to path

Tracing:
  -b execve      detach on execve syscall
  -D             run tracer process as a detached grandchild, not as parent
  -f             follow forks
  -ff            follow forks with output into separate files
  -I interruptible
     1:          no signals are blocked
     2:          fatal signals are blocked while decoding syscall (default)
     3:          fatal signals are always blocked (default if '-o FILE PROG')
     4:          fatal signals and SIGTSTP (^Z) are always blocked

                 (useful to make 'strace -o FILE PROG' not stop on ^Z)

Startup:
  -E var         remove var from the environment for command
  -E var=val     put var=val in the environment for command
  -p pid         trace process with process id PID, may be repeated
  -u username    run command as username handling setuid and/or setgid

Miscellaneous:
  -d             enable debug output to stderr
  -v               verbose mode: print unabbreviated argv, stat, termios, etc.
args
  -h             print help message
  -V             print version
  -k             obtain stack trace between each syscall (experimental)
[buba@Bubarch ~]$
       -
```

Aquests parametres serviran per poder obtenir la informació que desitjam i donar-li un cert format, s'explicaran alguns segons es vagin emprant en els exemples que es mostren seguidament.

Exemple 2

Es mostra un exemple simple com la execució de la comanda echo que donarà un text per pantalla, en aquest cas el text «hola».

```
[buba@Bubarch /]$ echo hola
hola
```

Ara es prova amb la comanda strace per fer-ne el seguiment

```
[buba@Bubarch /]$ strace echo "hola"
execve("/usr/bin/echo", ["echo", "hola"], [/* 35 vars */]) = 0
brk(NULL)                               = 0x1829000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=271601, ...}) = 0
mmap(NULL, 271601, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7efd91e1a000
close(3)                                = 0
open("/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\3\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=1951744, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7efd91e18000
mmap(NULL, 3791152, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7efd9189d000
mprotect(0x7efd91a32000, 2093056, PROT_NONE) = 0
mmap(0x7efd91c31000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x194000) = 0x7efd91c31000
mmap(0x7efd91c37000, 14640, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7efd91c37000
close(3)                                = 0
arch_prctl(ARCH_SET_FS, 0x7efd91e19480) = 0
mprotect(0x7efd91c31000, 16384, PROT_READ) = 0
mprotect(0x606000, 4096, PROT_READ)     = 0
mprotect(0x7efd91e5d000, 4096, PROT_READ) = 0
munmap(0x7efd91e1a000, 271601)          = 0
brk(NULL)                               = 0x1829000
brk(0x184a000)                          = 0x184a000
open("/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=2919952, ...}) = 0
mmap(NULL, 2919952, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7efd915d4000
close(3)                                = 0
```

```
fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
write(1, "hola\n", 5hola
)                   = 5
close(1)                                = 0
close(2)                                = 0
exit_group(0)                           = ?
+++ exited with 0 +++
```

La sortida resulta una mica llarga i pareix tediosa, però basta fixar-se una mica per entendre-la, a cada línia apareix el nom de la funció de la crida de sistema que s'ha realitzada amb els arguments que s'hi passen i després un igual amb el resultat retornat.

Per exemple de les linees anteriors a:

```
write(1, "hola\n", 5hola
)                   = 5
```

Es veu que es fa una crida a funció write amb una serie d'arguments i retorna un 5,

La manera per sabre per a que serveix cada argument es mirant el man de la funció cridant man 2 nom de la funció. Si executam `man 2 write`,

```
 ssize_t write(int fd, const void *buf, size_t count);
```

Podem veure que els paràmetres son:

- fd que és file descriptor (descriptor de fitxer, que es un identificador que es dona a un fitxer un pic a memòria), en aquest cas 1 que indica la finestra de sortida (el stdout), en l'opció y a la comanda strace donara el nom del fitxer que identifica.
- El text a escriure, es a dir "hola\n" incloent el retorn de carro
- La mida d'aquest (5). I quan aquesta funció s'executa correctament torna el nombre de caràcters escrits (5).

D'aquesta manera es pot monitoritzar el que va fent un procés.

Es pot apreciar que surten moltes funcions que es possible en aquest moment no ens interessin, com quan s'obrin llibreries, o fitxers que indiquen configuracions regionals, o les càrregues en memòria d'aquest fitxers i la protecció de memòria.

Exemple 3

Per això strace conta amb una opció per que només es mostri informació d'algunes crides de memòria, això es fa amb l'opció -e, per exemple:

```
[buba@Bubarch UOC]$ strace -e trace=execve,write echo hola
execve("/usr/bin/echo", ["echo", "hola"], [/* 35 vars */]) = 0
write(1, "hola\n", 5hola
)                   = 5
+++ exited with 0 +++
```

En aquest cas només es mostra les crides execve que serveix per executar un programa i la de write abans descrita.

Exemple 4

També hi ha l'opció per in incloure segells de temps tant per saber a quina hora comença (t per segons, tt o tt per millorar l'escala), com el temps que tarda en fer la crida (T)

```
[buba@Bubarch /]$ strace -Ttte trace=execve,write echo hola
04:32:55.400287 execve("/usr/bin/echo", ["echo", "hola"], [/* 35 vars */]) = 0
<0.000410>
04:32:55.402024 write(1, "hola\n", 5hola
)   = 5 <0.000032>
04:32:55.402297 +++ exited with 0 +++
[buba@Bubarch /]$
```

Encara que el temps no sigui molt reals per la sobrecarrega que aporta el seguiment, si que pot ajudar a trobar temps anormalment llargs.

Exemple 5

Una de les millors accions que pot dur a terme strace és que es pot adjuntar a un procés ja que ja està en marxa, per això basta proporcionat-li el pid al parèametre p, això ens permet comprobar per què un procés tarda molt a dur-se a terme o obtenir informació d'aquest.

Per exemple es pot saber quantes cridades s'han dut a terme, en aquest cas s'ha emprat el argument -p 1255 per adjuntar-se a aquest PID que correspon al navegador *chromium*. L'opció c es perquè mostri estadístiques i la F per que strace es vagi adjuntant a tots els fills de PID que li hem donat

```
buba@Bubarch ~]$ sudo strace -cqF -p 1255
^C% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 83.89  172.731241        1430    120807     16037 futex
  6.16   12.673687         303     41817           epoll_wait
  5.68   11.703330      688431        17         9 restart_syscall
  1.84    3.798170          82     46207           poll
  0.99    2.036666       19773       103           fdatasync
  0.54    1.106717        1037      1067           pread64
  0.18    0.380078         630       603           unlink
  0.17    0.357745           5     67372      3848 open
  0.17    0.340000        4789        71        57 lstat
  0.15    0.306291          43      7054           pwrite64
  0.11    0.216666      108333         2           wait4
  0.04    0.079999        4706        17           rename
  0.02    0.048894           0    198889       393 read
  0.02    0.046809          69       675       105 stat
  0.01    0.019614           0    145826    131530 recvmsg
  0.01    0.013219           2      8015           getdents
  0.00    0.007977           1     15945        19 sendto
  0.00    0.006788           0     24199           write
  0.00    0.006639           0     60945           madvise
  0.00    0.003347           4       954           ftruncate
  0.00    0.001678           0     61971           fstat
  0.00    0.001549           0     65466           close
  0.00    0.001416           0     33843           gettid
  0.00    0.001289           0      3511           getdents64
  0.00    0.001157           0     23640           ioctl
  0.00    0.000931           0     10113           fcntl
  0.00    0.000250           0      2624           select
  0.00    0.000225           0      2174           writev
  0.00    0.000185           0       448           munmap
  0.00    0.000154           0      5244           semop
  0.00    0.000077           0       203           getsockname
  0.00    0.000065           0      1419       828 recvfrom
  0.00    0.000030           0       825           sendmsg
  0.00    0.000026           1        51           lseek
```

```
  0.00    0.000024           0       213           socket
  0.00    0.000000           0       455           mmap
  0.00    0.000000           0         6           mprotect
  0.00    0.000000           0         1           brk
  0.00    0.000000           0       129         1 rt_sigaction
  0.00    0.000000           0       164           rt_sigprocmask
  0.00    0.000000           0         1         1 rt_sigreturn
  0.00    0.000000           0       116         8 access
  0.00    0.000000           0      1148           dup
  0.00    0.000000           0         1           dup2
  0.00    0.000000           0       261        76 connect
  0.00    0.000000           0        40           bind
  0.00    0.000000           0         4           socketpair
  0.00    0.000000           0       409        80 setsockopt
  0.00    0.000000           0       424           getsockopt
  0.00    0.000000           0         4           clone
  0.00    0.000000           0         1           execve
  0.00    0.000000           0       164           uname
  0.00    0.000000           0         2           getrlimit
  0.00    0.000000           0         1           getuid
  0.00    0.000000           0        52           geteuid
  0.00    0.000000           0         6           statfs
  0.00    0.000000           0         1           getpriority
  0.00    0.000000           0         6         2 setpriority
  0.00    0.000000           0         2           prctl
  0.00    0.000000           0         1           arch_prctl
  0.00    0.000000           0         1           set_tid_address
  0.00    0.000000           0       928           epoll_ctl
  0.00    0.000000           0         1           openat
  0.00    0.000000           0        73           ppoll
  0.00    0.000000           0         5           set_robust_list
------ ----------- ----------- --------- --------- ----------------
100.00  205.892933                956737    152994 total
```

Als resultats es pot observar el percentatge del temps emprat per cada crida de sistema, el temps emprat, els microsegons per crida, el nombre de crides, el nombre d'errors i nom de la crida.

Exemple 6

Altra informació útil que dona son les indicacions d'errors quan no troba un fitxer o no te permisos per tractar-lo.

Per exemple si s'intenta llistar un fitxer que no existeix

```
[buba@Bubarch UOC]$ strace -e trace=stat  ls  tt
stat("tt", 0xcfe150)                    = -1 ENOENT (No such file or directory)
ls: no se puede acceder a 'tt': No existe el fichero o el directorio
+++ exited with 2 +++
```

La crida de de sistema stat intenta obtenir informació del fitxer tt però dona un error de resultat -1 i indica el tipus ENOENT (No existeix el fitxer). Això pot ser útil si un programa o un *script* llarg no funciona perquè no troba un fitxer, especialment si no informa perquè no funciona.

```
[buba@Bubarch /]$ strace -e open  touch cc
open("/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
open("/usr/lib/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
open("/usr/lib/locale/locale-archive", O_RDONLY|O_CLOEXEC) = 3
open("cc", O_WRONLY|O_CREAT|O_NOCTTY|O_NONBLOCK, 0666) = -1 EACCES (Permission denied)
.....
: Permiso denegado
+++ exited with 1 +++
```

Un altra cas parescut es quan no es pot tenir accés al fitxer si ens fixem la crida open quan intentem crear un fitxer al directori arrel sense ser *root*, ens dona l'error de que no hi tenim permisos.

10. Ltrace

10.1 Què fa?

Ltrace es una aplicació molt semblant a strace, però en aquest cas intercepta i enregistra les crides a funcions de llibreries dinàmiques realitzades per un procés i els seus valors de retorn. A més, com strace, també pot interceptar i imprimir les crides a sistema fetes.

10.2 Com ho fa?

S'han d'explicar alguns conceptes abans d'entrar un poc més en detall en com funciona ltrace:

- **Llibreries dinàmiques:** Són llibreries de funcions les quals poden ser emprades per qualsevol programa una vegada carregades en memòria, de manera que si dos processos diferents necessiten alguna funció continguda en aquesta llibreria, no és necessari tornar a carregar la mateixa funció en memòria perquè ja hi és

- **ELF (Executable and Linkable Format)**: És un format d'arxiu executable amb una estructura concreta que s'empra a Linux i altres sistemes operatius. Té una estructura concreta definida per un estàndard per tal de tenir una estructura compartida per d'un fitxer executable. Té varies seccions

- **.got (Global Offset Table)**: És una secció del fitxer executable (ELF) que conté un punter per a cada una de les funcions de llibreria dinàmica que empra el programa.

- **.plt: (Procedure Linkage Table):** És una secció del executable que conté codi que permet junt amb la secció *.got* obtenir l'adreça absoluta del codi de la funció que es troba a una llibreria dinàmica.

- **Enllaçador dinàmic:** Es la llibreria que conté la lògica necessària per a la carrega dinàmica de llibreries. Llegeix la informació de capçalera del executable en format ELF que està compilat dinàmicament. Amb aquesta informació determina quines funcions de llibreries és necessari carrega en memòria(aquesta informació es troba la secció *.got*) . Aleshores realitza els enllaços dinàmics i manipula els punters de les adreces dins el executable(a la secció *.plt*)

Com Strace també empra la crida a sistema *ptrace* sigui amb la bandera PTRACE_TRACEME o PTRACE_ATTACH per fer que el procés seguidor sigui pare del procés al que volem fer el seguiment.

Després localitza el *.plt* del programa del que feim al seguiment

Emprant *ptrace* amb la bandera PTRACE_POKETEXT i PTRACE_PEEKTEXT sobreescriu a la secció *.plt* per posar els punts de ruptura (*breakpoints*) quan es fa una crida a una funció d'una llibreria dinàmica.

Quan es crea un punt de ruptura es guarda a una estructura de dades on s'inclou la instrucció original per poder tornar-la a posar si s'atura de fer el seguiment.

Es posen punts de ruptura tant quan s'entra a la funció per obtenir-ne els arguments com a la sortida per obtenir-ne el resultat retornat, aquests punts de ruptura poden ser activats i desactivats.

Una vegada fet això, es segueix executant el programa emprant la bandera PTRACE_SYSCALL i quan aquest faci una crida a una funció s'arribarà a un punt de ruptura, també pot passar que s'hagi arribat al principi o al final de una crida de sistema.

Per esperar s'empra la instrucció *wait()*. Quan s'hi arriba, intervé el nucli de Linux i envia el senyal SIGTRAP a ltrace

Aleshores ltrace examina el programa per sabre per que ha rebut el SIGTRAP. Si prové d'una crida de sistema (si l'opció adequada s'ha activat) o si és un dels punts de ruptura. Així podrà imprimir per pantalla informació com: quina funció de llibreria dinàmica ha cridat, la seva adreça de memòria, els arguments passats o altres dades demanades per l'usuari. Empra *ptrace*(PTRACE_PEEKUSER) per determinar l'origen.

Si és una crida al sistema de manera símil-lar al strace, es poden obtenir les dades i continuar executant la crida amb *ptrace*(PTRACE_SYSCALL) per que es torni atura al acabar la crida i retorna el resultat.

Si el senyal SIGTRAP rebut ve d'un punt de ruptura es un poc més complicat. Ja que s'ha d'obtenir l'adreça de memòria on s'ha produït aquest amb la funció *ptrace*(PTRACE_PEEKUSER) i mira a la nostra llista a quina funció correspon, d'on també podem sabre quants d'arguments tendrà i de quin tipus seran.

Amb aquesta informació, s'han d'obtenir els arguments de la funció amb ptrace(PTRACE_PEEKUSER) , i ja es poden mostrar per pantalla. Ara s'ha de posar un altra punt de ruptura a la adreça de memòria a on retorni la funció per a poder obtenir el resultat de la funció.

Finalment ltrace ha de tornar a posar la secció *.plt* del ELF amb el codi original de manera que el programa pot continuar executant-se correctament, però deixant actiu el punt de

ruptura de manera que es torni aturar quan tornem a cridar a la funció de la llibreria dinàmica.

Per fer això, s'elimina el punt de ruptura de forma temporal posant la instrucció original abans de continuar. Si fa falta, es decrementa el contador de programa per tornar-nos a situar a la crida de la funció. S'executa un sol pas amb *ptrace*(SINGLE_STEP) i tornam a posar el punt de ruptura.

Així ara es pot continuar executant el procés a seguir com estava previst originalment amb *ptrace*(PTRACE_SYSCALL)

Al acabar de executar la funció, s'arribarà al punt de ruptura situat a l'adreça de retorn i on podem obtenir el valor de retorn de la funció per mostrar-lo després s'ha de desactivar aquest punt de ruptura i continuar l'execució.

10.3 Ús

El seu ús es com qualsevol altra comanda de Linux, és a dir, el nom de la comanda amb una serie d'arguments, un d'ells pot ser la comanda la qual volem fer el seguiment.

```
ltrace ls
```

Altres arguments poden emprar-se per saber el temps de execució, o fer el seguiment d'un procés que ja està en execució o per saber quantes crides de sistema i de quin tipus ha fet un procés.

10.4 Avantatges

- ltrace es simple, el seu ús és molt semblant a strace, tracta crides de sistema i llibreries dinàmiques, es una comanda POSIX.

- La sortida que dona es pot entendre fàcilment i es exportable a un fitxer per facilitar el seu anàlisi, no es necessari tractar-la.

- Molt empleat i madur

10.5 Inconvenients

- Pot provocar una gran increment del temps de sistema respecte a si no el fem servir, s'ha de recordar que quan es rep el senyal SIGTRAP tot el temps que s'esta tractant la sortida es temps incrementat en l'execució del procés, al igual que el strace.

- Només es pot fer el seguiment dels processos fills que s'estan seguint

- La seva portabilitat es una mica complexa, ja que segons l'arquitectura pot variar la manera en que s'obtenen arguments de funcions o la manera en que avança el contador de programa.

- No es recomanable el seu ús en processos de molts de fils.

10.6 Exemples d'ús de ltrace

Exemple 7

Amb la comanda -h obtenim d'informació d'ajuda.

```
[buba@Bubarch UOC]$ ltrace -h
Usage: ltrace [option ...] [command [arg ...]]
Trace library calls of a given program.

  -a, --align=COLUMN  align return values in a secific column.
  -A MAXELTS          maximum number of array elements to print.
  -b, --no-signals    don't print signals.
  -c                  count time and calls, and report a summary on exit.
  -C, --demangle      decode low-level symbol names into user-level names.
  -D, --debug=MASK    enable debugging (see -Dh or --debug=help).
  -Dh, --debug=help   show help on debugging.
  -e FILTER           modify which library calls to trace.
  -f                  trace children (fork() and clone()).
  -F, --config=FILE   load alternate configuration file (may be repeated).
  -h, --help          display this help and exit.
  -i                  print instruction pointer at time of library call.
  -l, --library=LIBRARY_PATTERN only trace symbols implemented by this library.
  -L                  do NOT display library calls.
  -n, --indent=NR     indent output by NR spaces for each call level nesting.
  -o, --output=FILENAME write the trace output to file with given name.
```

```
  -p PID              attach to the process with the process ID pid.
  -r                  print relative timestamps.
  -s STRSIZE          specify the maximum string size to print.
  -S                  trace system calls as well as library calls.
  -t, -tt, -ttt       print absolute timestamps.
  -T                  show the time spent inside each call.
  -u USERNAME         run command with the userid, groupid of username.
  -V, --version       output version information and exit.
  -x FILTER           modify which static functions to trace.

Report bugs to ltrace-devel@lists.alioth.debian.org
```

Es veu que hi ha bastantes opcions, tal vegada les més destacables són les que ens permeten adjuntar-nos a un procés que esta en marxa(-p)comprovat les crides de sistema (-S) com fa strace,triar quines crides volem mostrar (e),limitar-nos a una llibreria(-l),imprimir segells de temps (-t,-tt,-ttt) i fer estadistiques (-c). Tambè n'hi ha que ajuden a entendre el fluxe del programa.

Exemple 8

Aquesta és la sortida si es fa un ltrace echo hola:

```
[buba@Bubarch UOC]$ ltrace echo hola
getenv("POSIXLY_CORRECT")                         = nil
strrchr("echo", '/')                              = nil
setlocale(LC_ALL, "")                             = "es_ES.UTF-8"
bindtextdomain("coreutils", "/usr/share/locale") = "/usr/share/locale"
textdomain("coreutils")                           = "coreutils"
__cxa_atexit(0x401de0, 0, 0, 0x736c6974756572)   = 0
strcmp("hola", "--help")                          = 59
strcmp("hola", "--version")                       = 59
fputs_unlocked(0x7ffdabbe8c0e, 0x7f82bead95e0, 45, 32) = 1
__overflow(0x7f82bead95e0, 10, 0x1b70033, 0xfbad2a84hola
) = 10
__fpending(0x7f82bead95e0, 0, 0x401de0, 0x7f82bead9c30) = 0
fileno(0x7f82bead95e0)                            = 1
__freading(0x7f82bead95e0, 0, 0x401de0, 0x7f82bead9c30) = 0
__freading(0x7f82bead95e0, 0, 2052, 0x7f82bead9c30) = 0
```

```
fflush(0x7f82bead95e0)                              = 0
fclose(0x7f82bead95e0)                              = 0
__fpending(0x7f82bead9500, 0, 0x7f82bead48c0, 2880) = 0
fileno(0x7f82bead9500)                              = 2
__freading(0x7f82bead9500, 0, 0x7f82bead48c0, 2880) = 0
__freading(0x7f82bead9500, 0, 4, 2880)              = 0
fflush(0x7f82bead9500)                              = 0
fclose(0x7f82bead9500)                              = 0
+++ exited (status 0) +++
[buba@Bubarch UOC]$
```

Es pot observar com es mostren les funcions cridades, amb els paràmetres que s'hi passen i el retorn que en fan les funcions, així com quan el procés acaba.

Exemple 9

Es pot limitar el nom de funció que es vol observar amb el paràmetre -e, es pot usar una expressió per limitar-ho un parell de funcions (al exemple +setlocale). Inclús emprar caràcters comodí per només haver de posar-hi una part de la funció (com es fa amb '*str*')

```
[buba@Bubarch UOC]$ ltrace -e '*str*'+setlocale  echo hola
echo->strrchr("echo", '/')                          = nil
echo->setlocale(LC_ALL, "")                         = "es_ES.UTF-8"
echo->strcmp("hola", "--help")                      = 59
echo->strcmp("hola", "--version")                   = 59
hola
+++ exited (status 0) +++
[buba@Bubarch UOC]$
```

Exemple 10

També es pot limitar la sortida a una llibreria concreta, en teoria es fa amb el paràmetre -l, però al propi manual recomanen el -x '*@patrollibreria' el patrò és per poder agafar varies llibreries, amb això em a passa algo curiós, a la versió que es té la distribució que s'ha emprat (0.7.3), amb el paràmetre -l no va bè:

```
[buba@Bubarch proves]$ ltrace -l /usr/lib/libbz2.so.1.0 bzip2 test
+++ exited (status 0) +++
```

Amb el suggerit al manual tampoc, donava tota la sortida, es posen parts ja que sinó serà masa llarga

```
buba@Bubarch proves]$ ltrace  -x '*@libbz2.so.1.0' bzip2 test
__libc_start_main(0x401a40, 2, 0x7ffd8643bf38, 0x404eb0 <unfinished ...>
signal(SIGSEGV, 0x403450)                        = 0
signal(SIGBUS, 0x403450)                         = 0
strlen("(none)")                                 = 6
strncpy(0x608420, "(none)", 1024)                = 0x608420
...
ferror(0x804280)                                 = 0
BZ2_bzWriteOpen(0x7ffd8643a868, 0x804280, 9, 0 <unfinished ...>
BZ2_bzWriteOpen@libbz2.so.1.0(0x7ffd8643a868, 0x804280, 9, 0 <unfinished ...>
BZ2_bzCompressInit@libbz2.so.1.0(0x805848, 9, 0, 30) = 0
...
_fini@libbz2.so.1.0(0x7fbb3392a920, 0, 0xffffffff, 0) = 0x7fbb33728340
+++ exited (status 0) +++
```

S'observen crides a funcions que comencen per BZ2 que son clarament de la llibreria i altres que no ho son

Només va funcionar emprat l'opció -e

```
[buba@Bubarch proves]$ ltrace  -e '*@libbz2.so.1.0' bzip2 test
libbz2.so.1.0->ferror(0x20c7280)                 = 0
libbz2.so.1.0->malloc(5104)                      = 0x20c74b0
libbz2.so.1.0->BZ2_bzCompressInit(0x20c8848, 9, 0, 30 <unfinished ...>
libbz2.so.1.0->malloc(55768)                     = 0x20c88b0
...
libbz2.so.1.0->BZ2_bzCompress(0x20c8848, 0, 0x7ffdfc239660, 5) = 1
libbz2.so.1.0->ferror(0x20c7280)
```

```
...
libbz2.so.1.0->free(0x20c74b0)                       = <void>
libbz2.so.1.0->__cxa_finalize(0x7f5fae32dd00, 0, 0, 0) = 0x7f5fae11b010
+++ exited (status 0) +++
```

Aquí es pot veure com aparentment totes les crides mostrades son a la llibreria demanada.

En canvi es va fer una prova a una màquina virtual amb una ubuntu antiga de l'assignatura de seguretat, amb una versió anterior de ltrace, la 0.5.3, el bzip també es anterior

```
uocseg@vm-mail:~/test$ ltrace -l /lib/libbz2.so.1.0 /bin/bzip2  test
BZ2_bzWriteOpen(0xbfc42830, 0x9ba9190, 9, 0, 30) = 0x9ba92f8
BZ2_bzWrite(0xbfc42830, 0x9ba92f8, 0xbfc42844, 5, 30) = 5000
BZ2_bzWriteClose64(0xbfc42830, 0x9ba92f8, 0, 0xbfc42840, 0xbfc4283c) = 134416
+++ exited (status 0) +++
uocseg@vm-mail:~/test$
```

S'ha capturat la imatge per destacar que es una màquina diferent però ja es veu aquí si fa cas al paràmetre -l.

Amb al paràmetre -e també es pot comprovar on es troba la funció que es crida, però nomes si es fa la cerca i més com accepta expressions es cercar les que comencen per una cadena concreta com s'ha fet a continuació.

```
buba@Bubarch proves]$ ltrace  -e 'BZ2*' bzip2 test
bzip2->BZ2_bzWriteOpen(0x7ffca55d15c8, 0x2111280, 9, 0 <unfinished ...>
libbz2.so.1.0->BZ2_bzCompressInit(0x2112848, 9, 0, 30) = 0
...
libbz2.so.1.0->BZ2_bzCompress(0x2112848, 0, 0x7ffca55d1630, 5) = 1
...
<... BZ2_bzWriteClose64 resumed> )                   = 1
+++ exited (status 0) +++
```

Es pot observar que només es mostren funcions que comencen per BZ2 i si provenen del executable bzip2 o de llibreria. Seria una millora que no fos necessari aquest paràmetre.

Exemple 11

El programa ltrace permet observar les crides de processos ja existents amb el parametre -p PID i als seus fills amb -f i se li pot dir que la sortida l'envii a un text amb -o.

Això junt amb la possibilitat de filtrar pot donar molt de joc, però sembla que la darrera versió al sistema emprat no acaba d'anar com toca, potser sigui alguna protecció inclosa o el mètode com s'ha compilat, així i tot es demostrarà amb alguns exemples.

Per això s'ha emprat el thunar que es el gestor d'arxius i carpetes al entorn gràfic que s'empra (XFCE), es cerquen funcions que formin part de pango que es una llibreria per mostrar i col·locar text per GTK. Aquesta es la comanda emprada:

```
[buba@Bubarch proves]$ sudo ltrace -e 'pango*' -f -p 856 -o thunarltrace.log
```

Després s'ha anat al finestra oberta de *thunar* i s'ha mogut el punter i seleccionat algun fitxer i mostrat algun menú. En pocs segons es tenia un fitxer de registre de 221 Kb.

Obrint el text i agafant una petita part

```
856 exe->pango_layout_set_attributes(0xbfd020, 0, 3, 3) = 0xc4fdc0
856 exe->pango_layout_set_width(0xbfd020, 0xffffffff, 0xffffffff, 0xc4fdc0) = 0xffffffff
856 exe->pango_layout_set_wrap(0xbfd020, 1, 0xffffffff, 0xc4fdc0) = 40
856 exe->pango_layout_set_text(0xbfd020, 0xc3e900, 0xffffffff, 0xc4fdc0) = 0
856 exe->pango_layout_set_alignment(0xbfd020, 0, 0xffffffff, 0xc5a2a0) = 40
856 exe->pango_layout_get_pixel_extents(0xbfd020, 0, 0x7fff4db14490, 0xc5a2a0) = 16
856 exe->pango_layout_set_attributes(0xbfd020, 0, 3, 3) = 0xc5a2a0
856 exe->pango_layout_set_width(0xbfd020, 0xffffffff, 0xffffffff, 0xc5a2a0) = 0x
```

Ja es veu al PID al que s'ha adjuntat el ltrace (856) i es mostra com totes les crides es fan des de un exe i no cap llibreria.

En canvi executant el *thunar* des del principi amb la comanda:

```
ltrace -e 'pango*' -f -o thunarltrace2.log thunar
...
2972 thunar->pango_layout_get_pixel_extents(0x21848c0, 0, 0x7ffc72f69c60, 0x21dfde0 <unfinished ...>
2972 <... pango_layout_get_pixel_extents resumed> ) = 15
2972 libgdk-x11-2.0.so.0->pango_layout_get_type(0x20f6c20, 0x2127870, 1, 1) = 0x20807b0
2972 libgdk-x11-2.0.so.0->pango_layout_get_type(0x20f6c20, 0x2127870, 1, 1) = 0x20807b0
```

Es pot observar com ara si es veuen funcions que es criden des de llibreries i allà on es troben. A la versió provada a la màquina virtual només captura el missatge de sortida exit. Com he dit no s'ha aconseguit esbrinar perquè passa això però es digne d'estudi.

Exemple 12

Altra activitat que es pot dur a terme em ltrace es incloure a més de les crides a funcions, les crides a sistema

```
[buba@Bubarch proves]$ ltrace -S echo 'hola'
SYS_brk(0)                                          = 0x1675000
SYS_access("/etc/ld.so.preload", 04)                = -2
SYS_open("/etc/ld.so.cache", 524288, 01)            = 3
SYS_fstat(3, 0x7fff798c4420)                        = 0
SYS_mmap(0, 0x424f1, 1, 2)                          = 0x7f528ed13000
SYS_close(3)                                        = 0
SYS_open("/usr/lib/libc.so.6", 524288, 021665300420) = 3
SYS_read(3, "\177ELF\002\001\001\003", 832)         = 832
SYS_fstat(3, 0x7fff798c4460)                        = 0
SYS_mmap(0, 8192, 3, 34)                            = 0x7f528ed11000
SYS_mmap(0, 0x39d930, 5, 2050)                      = 0x7f528e796000
SYS_mprotect(0x7f528e92b000, 2093056, 0)            = 0
SYS_mmap(0x7f528eb2a000, 0x6000, 3, 2066)           = 0x7f528eb2a000
SYS_mmap(0x7f528eb30000, 0x3930, 3, 50)             = 0x7f528eb30000
SYS_close(3)                                        = 0
SYS_arch_prctl(4098, 0x7f528ed12480, 0x7f528ed12d90, 0x7f528ed11978) = 0
SYS_mprotect(0x7f528eb2a000, 16384, 1)              = 0
SYS_mprotect(0x606000, 4096, 1)                     = 0
SYS_mprotect(0x7f528ed56000, 4096, 1)               = 0
SYS_munmap(0x7f528ed13000, 271601)                  = 0
getenv("POSIXLY_CORRECT")                           = nil
strrchr("echo", '/')                                = nil
setlocale(LC_ALL, "" <unfinished ...>
SYS_brk(0)                                          = 0x1675000
SYS_brk(0x1696000)                                  = 0x1696000
SYS_open("/usr/lib/locale/locale-archive", 524288, 037777777777) = 3
SYS_fstat(3, 0x7f528eb2f940)                        = 0
SYS_mmap(0, 0x2c8e10, 1, 2)                         = 0x7f528e4cd000
SYS_close(3)                                        = 0
```

```
<... setlocale resumed> )                        = "es_ES.UTF-8"
bindtextdomain("coreutils", "/usr/share/locale") = "/usr/share/locale"
textdomain("coreutils")                          = "coreutils"
__cxa_atexit(0x401de0, 0, 0, 0x736c6974756572)   = 0
strcmp("hola", "--help")                         = 59
strcmp("hola", "--version")                      = 59
fputs_unlocked(0x7fff798c5c07, 0x7f528eb2f5e0, 45, 32 <unfinished ...>
SYS_fstat(1, 0x7fff798c4ba0)                     = 0
<... fputs_unlocked resumed> )                   = 1
__overflow(0x7f528eb2f5e0, 10, 0x1676033, 0xfbad2a84 <unfinished ...>
SYS_write(1, "hola\n", 5hola
)                        = 5
<... __overflow resumed> )                       = 10
__fpending(0x7f528eb2f5e0, 0, 0x401de0, 0x7f528eb2fc30) = 0
fileno(0x7f528eb2f5e0)                           = 1
__freading(0x7f528eb2f5e0, 0, 0x401de0, 0x7f528eb2fc30) = 0
__freading(0x7f528eb2f5e0, 0, 2052, 0x7f528eb2fc30) = 0
fflush(0x7f528eb2f5e0)                           = 0
fclose(0x7f528eb2f5e0 <unfinished ...>
SYS_close(1)                                     = 0
<... fclose resumed> )                           = 0
__fpending(0x7f528eb2f500, 0, 0x7f528eb2a8c0, 2880) = 0
fileno(0x7f528eb2f500)                           = 2
__freading(0x7f528eb2f500, 0, 0x7f528eb2a8c0, 2880) = 0
__freading(0x7f528eb2f500, 0, 4, 2880)           = 0
fflush(0x7f528eb2f500)                           = 0
fclose(0x7f528eb2f500 <unfinished ...>
SYS_close(2)                                     = 0
<... fclose resumed> )                           = 0
SYS_exit_group(0 <no return ...>
+++ exited (status 0) +++
```

Aqui es mostra de la sortida completa del echo hola, es pot observar que si s'agafen només les crides a sistema, obtenim el mateix resultat que amb el strace.

Exemple 13

A dir veritat executat amb paràmetres -SL s'obtenen només les crides de sistema com es pot veure a continuació parcialment.

```
[buba@Bubarch proves]$ ltrace -SL echo 'hola'
SYS_brk(0)                                         = 0x1d5e000
SYS_access("/etc/ld.so.preload", 04)               = -2
SYS_open("/etc/ld.so.cache", 524288, 01)           = 3
SYS_fstat(3, 0x7ffc033b2450)                       = 0
SYS_mmap(0, 0x424f1, 1, 2)                         = 0x7ff993d29000
SYS_close(3)                                       = 0
SYS_open("/usr/lib/libc.so.6", 524288, 022365560420) = 3
SYS_read(3, "\177ELF\002\001\001\003", 832)      = 832
...
SYS_write(1, "hola\n", 5hola
)                        = 5
SYS_close(1)                                       = 0
SYS_close(2)                                       = 0
SYS_exit_group(0 <no return ...>
+++ exited (status 0) +++
[buba@Bubarch proves]$
```

Exemple 14

Més informació que es pot obtenir es el segell de temps com a strace (-tt) i el punter d'instrucció (i) i al final el temps de cada crida i funció (-T),com es mostra continuacio de manera parcial.

```
[buba@Bubarch proves]$ ltrace -ttTiS echo 'hola'
05:07:43.422370 [0x7f8aca26d199] SYS_brk(0)      = 0x1add000 <0.000108>
05:07:43.422561  [0x7f8aca26dec7]  SYS_access("/etc/ld.so.preload",  04)  =  -2
<0.000190>
...
05:10:36.736376 [0x7f252df98999] SYS_brk(0)      = 0x2232000 <0.000042>
05:10:36.736455 [0x7f252df98999] SYS_brk(0x2253000) = 0x2253000 <0.000038>
05:10:36.736531   [0x7f252dee2f0f]   SYS_open("/usr/lib/locale/locale-archive",
524288, 037777777777) = 3 <0.000227>
...
05:10:36.738146 [0x401421] strcmp("hola", "--version") = 59 <0.000154>
05:10:36.738341  [0x401596]  fputs_unlocked(0x7ffd6ac67c07,  0x7f252e2515e0,  45,
32 <unfinished ...>
```

```
05:10:36.738475 [0x7f252df92ea4] SYS_fstat(1, 0x7ffd6ac65db0) = 0 <0.000050>
05:10:36.738558 [0x401596] <... fputs_unlocked resumed> ) = 1 <0.000209>
05:10:36.738607 [0x401860] __overflow(0x7f252e2515e0, 10, 0x2233033, 0xfbad2a84
<unfinished ...>
05:10:36.738736 [0x7f252df93530] SYS_write(1, "hola\n", 5hola
) = 5 <0.000076>
...
5:10:36.741418 [0x7f252df70448] SYS_exit_group(0 <no return ...>
05:10:36.741572 [0xffffffffffffffff] +++ exited (status 0) +++
[buba@Bubarch proves]$
```

Exemple 15

A més també es pot obtenir una estadística de les crides a funció fetes, parescut a la que es pot fer amb strace amb les senyals, amb el mateix paràmetre -c.

```
[buba@Bubarch proves]$ ltrace -c echo 'hola'
hola
% time     seconds  usecs/call     calls      function
------ ----------- ----------- --------- --------------------
 15.47    0.000483         120         4 __freading
 13.90    0.000434         434         1 setlocale
  9.67    0.000302         151         2 fclose
  7.91    0.000247         123         2 __fpending
  7.82    0.000244         122         2 fileno
  7.75    0.000242         121         2 fflush
  7.11    0.000222         111         2 strcmp
  5.38    0.000168         168         1 __overflow
  5.29    0.000165         165         1 fputs_unlocked
  4.68    0.000146         146         1 getenv
  3.94    0.000123         123         1 bindtextdomain
  3.72    0.000116         116         1 textdomain
  3.68    0.000115         115         1 strrchr
  3.68    0.000115         115         1 __cxa_atexit
------ ----------- ----------- --------- --------------------
100.00    0.003122                    22 total
```

Així encara que ha hagut alguns problemes, aquesta aplicació dona molt informació la pot ajudar comprobar el rendiment, optimitzar i depurar aplicacions, encara que hi cap alguna millora.

11. DTrace

11.1 Què fa?

Aixi com les eines de traça anteriors són semblants entre elles per l'ús que fan de la crida de sistema *ptrace* del nucli de Linux i estan fetes de manera que siguin una petita utilitat, aquesta és molt diferent.

Dtrace es un entorn de treball (framework) originari del sistema operatiu Solaris 10 però que ha estat portat en part a Linuxi a FreeBSD, i en la seva totalitat a macOS. Permet observar, estudiar i obtenir informació del comportament de processos que es trobin tant al espai d'usuari com a l'espai del nucli. Ho fa amb instrumentació dinàmica tant dels processos de sistema com del usuari.

11.2 Com ho fa?

No s'explicarà exactament com funciona com s'ha fet anteriorment per la complexitat que té, però si que explicaré com ho fa per damunt, la seva arquitectura i el que es necessari saber per poder-ho emprar.

Components

Primer s'explicaran els components que formen part d'aquest *framework*.

Llenguatge D (D language): No s'ha de confondre amb el llenguatge de programació D, es un llenguatge d'*scripting* , una mescla de C i *awk* que permet fer *scripts* per emprar-los amb Dtrace.

Consumidor (Consumer): És el programa d'espai d'usuari que empra el framework, en el nostre cas serà el programa *dtrace* però es podria emprar per fer interfícies gràfiques com *dlight* a Solaris o Instruments a *macOS*. Al sistema operatiu Solaris per exemple hi ha comandes com *lockstat* que l'empren.

Sonda (Probe): Aquest és un punt de instrumentació activat per l'usuari. S'entén com punt de instrumentació una localització especifica dins el flux del programa, es poden llistar amb l'opció -l. En pot haver de dinàmiques i d'estàtiques. Se'n poden programar de noves per espai d'usuari per emprar-les a aplicacions seran estàtiques.

Proveïdor (Provider): Els proveïdors gestionen les sondes associades a un subsistema del nucli especific. Son llibreries de sondes, donen sondes amb noms comprensibles i informació relativa a aquestes sondes. Fa possible la instrumentació de programari sense tenir que ser un expert en aquella àrea del sistema. Per exemple *syscall* gestiona sondes als punts d'entrada i retorn de les crides de sistema. N'hi ha d'especials com la *dtrace* on

hi ha les sondes BEGIN on es pot inicialitzar variables i imprimir capçaleres i END on es podrien imprimir informes finals.

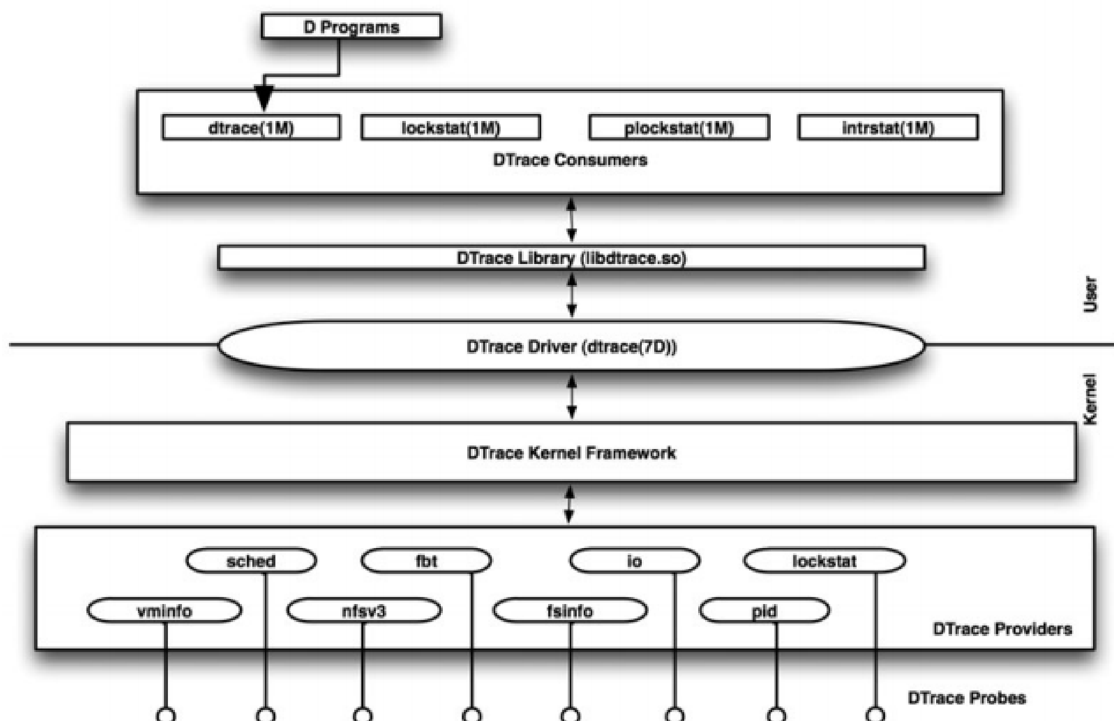
Arquitectura general

Ara que ja s'han descrit les parts principals de DTrace es pot donar una ullada a la seva arquitectura.

Els consumidors de DTrace s'executen en espai d'usuari, el que s'emprarà a aquest treball es únicament la comanda *dtrace*. empren la llibreria *libtrace.so*, aquesta no té una interfície pública sinó que s'empra a través de la comanda dtrace.

Quan un *script* fet en D s'executa es compila en *bytecode*. El codi es validat i executat en el nucli en un entorn semblant a una maquina virtual, per això DTrace inclou un processador emulat que empra el conjunt d'instruccions RISC, es parescut a com funciona Java en això.

Hi ha interfícies entre el nucli de DTrace i els proveïdors. El entorn de treball activa les sondes sol·licitades al programa en D compilat demanat-ho als proveïdors. Durant l'execució del programa en D les dades demanades son recollides, emmagatzemades i retornades al consumidor que els ha demanat. Quan el programa s'acaba d'executar, els proveïdors desactiven les sondes.



Arquitectura del llenguatge D

Components del llenguatge D

Descripció de les sondes

La descripció de les sondes es la següent:

provider:module:function:name

on

provider (proveïdor): Són llibreries de sondes que instrumenten una àrea especifica del sistema, (per exemple shed per *scheduller*) o un mode de seguiment (per exemple fbt). Segons es treuen noves versions, es treuen noves sondes i nous proveïdors (tcp, ip, etc)

module (mòdul): El mòdul de nucli (*kernel*) on les sondes es troben. Per sondes que es troben a l'espai d'usuari es mostra la llibreria compartida que conté la sonda.

function (funció) es la funció que conte la sonda

name (nom): Es un nom descriptiu de la sonda, per exemple com *entry* (entrada) o *return* (retorn) son sondes que s'engeguen a l'entrada i al retorn de la corresponent funció.

Clàusula (Clause): Aquestes són les accions definides per l'usuari que s'executaran quan una sonda s'engegui. Les accions poden ser tals com recol·lecció de dades, captura segells de temps entre altres. Van precedides per una descripció de la sonda, son opcionals i si existeixen estan entre claus ({ i })

Predicat (Predicate): És una instrucció condicional definida per l'usuari i que s'avalua quan les sondes s'engeguen. Activa la captura de dades només si una o un grup de condicions son veritat. És opcional i s'escriu entre barres (/).

Variable (Variable): Com a altres llenguatges de programació, una variable a DTrace pot emmagatzemar un objecte d'un cert tipus de dades. DTrace suporta variables definides per l'usuari, així com un conjunt de variables integrades.

Agregació (Aggregation): És un tipus de variable especial i el seu conjunt de funcions per a col·locació i representació de dades. Dona una manera simple per representar i agrupar les dades

Funció (Function): Es qualsevol de les moltes funcions que poden ser cridades com part de les accions definides per l'usuari al llenguatge D

Estructura de un programa en D

Els programes en D es poden cridar tant des de la pròpia línia de comanda amb el paràmetre -n o es poden posar en forma d'*script*.

A la linea de comanda tendran la forma:

```
dtrace -n 'sonda /predicat/ {clàusula}
```

Un script en D tendrà la forma:

```
#!/usr/sbin/dtrace -s
sonda
/predicat/
{
 clàusula
}
```

11.3 Ús

El seu ús es molt ampli pot anar des d'una simple comanda curta, a una línia més llarga o a executar un script fet en D que pot tenir una longitud variable.

Ja que empra un llenguatge propi per els scripts té molts de paràmetres, i a més depenent de la quantitat de sondes disponibles pot variar la quantitat de accions que es pot dur a terme, hi ha sistemes operatius que tenen més sondes disponibles que d'altres.

Amb la comanda dtrace -l es poden veure les sondes disponibles.

11.4 Avantatges

- Es molt flexible ja que conta amb un llenguatge de scripting propi

- Segons quantitat de sondes disponibles ens permet monitoritzar la major part del nucli: crides de sistemes, crides a funcions del nucli, trafic de xarxa, entrada/sortida de discs

- La sortida que dona es molt flexible i disposa de funcions especifiques destinades a generar una sortida fàcil d'entendre.

- Es ampliable ja que es poden fer noves sondes i nous scripts en D, es molt empleat al sistemes on està disponible (Solaris i derivats, MacOS i FreeBSD

- Per la seva capacitat es molt recomanble per trobar problemes de rendiment

- Permet l'estudi de varis processos sense perdre gaire rendiment segons la quantitat de sondes activades

11.5 Inconvenients

- Es necessària l'existència de les sondes adequades per monitoritzar el que volem

- Si s'activen moltes sondes a la vegada es pot notar al rendiment de l'aplicació, han de ser moltes.

- La seva portabilitat es molt complexa, especialment per que el important es disposar de moltes sondes adequades, es per això que la versió de Linux no funciona adequadament mentre a macOS es la eina de traça que s'empra a través de Instruments i a Solaris està àmpliament suportada ja que és on es va originar, a FreeBSD també una mica més de suport que a Linux

11.6 Exemples d'us de DTrace

Donada la flexibilitat en que conta l'eina DTrace gracies al llenguatge D i la quantitat de punts de prova (sondes) disponibles es posaran alguns exemples que donaran una idea de la capacitat que té, també s'explicara un script per saber com són.

Aquest entorn de treball es molt més complet que les eines anteriors ja que com s'ha dit abans conta en moltes sondes per tot el kernel.

En principi es volia empra el FreeBSD per aquest treball però s'ha trobat que encara te algunes mancances i alguns scripts provats no funcionaven i hi ha moltes menys sondes implementades.

```
root@:/usr/local/share/dtrace-toolkit # dtrace -l ¦ wc -l
   59487
```

Al final al s'ha optat per emprar la implementació de Oracle Solaris 11.3 a una màquina virtual on no s'ha trobat cap problema de compatibilitat i es conta amb més sondes.

```
root@bubuSOL:/usr/dtrace/DTT# dtrace -l | wc -l
   81432
root@bubuSOL:/usr/dtrace/DTT#
```

Exemple 16

Una de les primeres coses que es pot fer es llistar les probes amb dstrace -l, 81.432, son moltes si s'agafen només les de del proveïdor syscall (crides de sistema son moltes menys, es pot filtrar el llistat per proveidor -l -p, per mòdul -l -m, per funció -l -f, per nom ,per exemple si es fa per mòdul syscall,n'hi ha 431 es posaren només una part.

```
root@bubuSOL:/usr/dtrace/DTT# dtrace -l -P syscall
   ID   PROVIDER            MODULE                          FUNCTION NAME
10167    syscall                                       so_socketpair return
10168    syscall                                                bind entry
10169    syscall                                                bind return
10170    syscall                                              listen entry
...
14155    syscall                                           so_socket entry
14156    syscall                                           so_socket return
14157    syscall                                       so_socketpair entry
root@bubuSOL:/usr/dtrace/DTT#
```

Surt per columnes amb un identificador únic, proveïdor, modul, funció i nom de la sonda.

Exemple 17

També es poden llistar limitant per varis camps emprant tuples separant els camps amb ':' i si es deixa en blanc un camp es com posa un asterisc, es dir tots. A més accepta expressions. s'empra l'opció -l -n proveïdor:mòdul:funció:nom

```
root@bubuSOL:/usr/dtrace/DTT# dtrace -l  -n syscall::read*:entry
   ID   PROVIDER            MODULE                          FUNCTION NAME
13781    syscall                                                read entry
13803    syscall                                          readlinkat entry
13975    syscall                                               readv entry
root@bubuSOL:/usr/dtrace/DTT#
```

Així en aquest exemple emprant l'asterisc per veure totes les funcions amb una part de la funció read, amb nom entry i pertanyents al proveïdor syscall.

Només amb la comanda amb linea i un parell de paràmetres es poden fer bastantes coses, es com si s'executes un petit script, un ex-enginyer de Sun, ara de Netflix en té una grapada i també n'hi ha a la documentació de Oracle, es posaran alguns exemples.

Exemple 18

En una linea tambè es poden contar les crides de sistema per una comanda concret

```
root@bubuSOL:/tmp/test# dtrace -n 'syscall:::entry /pid == $target/
{@[probefunc]=count();}' -c 'ls'
dtrace: description 'syscall:::entry ' matched 215 probes
tt          tt.tar.bz2
dtrace: pid 2451 has exited

  fcntl                                                        1
  getpid                                                       1
  getrlimit                                                    1
  mmap                                                         1
  rexit                                                        1
  write                                                        1
  getdents                                                     2
  memcntl                                                      2
  mmapobj                                                      2
  resolvepath                                                  2
  setcontext                                                   2
  close                                                        3
  ioctl                                                        3
  openat                                                       3
  brk                                                          5
  fstatat                                                      6
root@bubuSOL:/tmp/test#
```

En aquest cas s'obtenen les crides de sistema que fa la comanda ls que s'executa.

S'empre la descripció de sonda que inclou totes les sondes d'entrada del proveidor syscall (syscall:::entry), al predicat es un condiconal on es mira que l'executable el pid a mirar sigui el de l'objectiu, (pid ==$target). un agregat (@[probefunc]=count() per contar les crides i el paràmetre -c 'ls' que executa el programa

Exemple 19

Pero es poden fer coses més complicades

```
root@bubuSOL:/usr/dtrace/DTT# dtrace -n 'syscall:::entry /execname != "dtrace"/
{ @sc[execname, probefunc] = count(); }'
dtrace: description 'syscall:::entry ' matched 215 probes
^C
  VBoxService                       fcntl                             1
  VBoxService                       lseek                             1
  VBoxService                       so_socket                         1
  dhcpagent                         pollsys                           1
  dlmgmtd                           lwp_park                          1
...
  gnome-terminal                    lseek                           161
  VBoxClient                        nanosleep                       196
  VBoxClient                        recv                            196
  java                              lwp_cond_signal                 198
  Xorg                              clock_gettime                   317
  java                              lwp_cond_wait                   403
root@bubuSOL:/usr/dtrace/DTT#
```

A la sortida es pot veure com surt el recompte de les vegades que es fan crides a sistema per programa i crida.

S'empre la descripció de sonda que inclou totes les sondes d'entrada del proveidor syscall (syscall:::entry), al predicat es un condiconal on es mira que l'executable no sigui dtrace (exename !=dtrace).Finalment un agregat (@sc que conti (count) per crida i programa

Exemple 20

Un altra exemple es mira quin fitxers es llegeixen al llançar el Navegador Firefox, per això es llança primer la comanda i després obrim el *firefoxi* els paràmetres que passam al dtrace són:

- La descripció de la sonda que es la del modul de crida de sistema, de la funció llegir a l'entrada, al predicat (syscall::read:entry.

- El predicat condiciona que només interessen les crides de sistema de *firefox* (execname==firefox)

- La clàusula es una variable d'agregació que inclou el nom del executable i el cami absolut del fitxer que es llegeix, i el recompte de les vegades que hi accedeix @reads[execname, fds[arg0].fi_pathname] = count();

El camí s'obté a partir de la variable interna *fds*, és una matriu que proporciona una traducció dels descriptors de fitxers que s'empren en un procés a una estructura de dades tipus *fileinfo_t* per a cada procés. Els fitxers estan indexats pel descriptor de fitxer que identifica cada fitxer obert. És el mateix descriptor de fitxer ,que és un nombre, que es passaria a una crida de sistema com llegir (read) o escriure (write).

El tipus *fileinfo_t* inclou el nom, el directori, el camí complet del fitxer, el sistema de fitxers entre altres. En aquest cas s'obté el camí complet del fitxer (fi_pathname).

Finalment hi tenim la funció que conta les vegades que s'accedeix (count()).

```
dtrace -n 'syscall::read:entry /execname == "firefox"/ { @reads[execname, fds[arg0].fi_pathname] = count(); }'
^C
 firefox              /etc/default/init                                  1
 firefox              /export/home/bu/.mozilla/firefox/profiles.ini        1
 firefox /export/home/bu/.mozilla/firefox/suqgjazk.default/compatibility.ini 1
 firefox /export/home/bu/.mozilla/firefox/suqgjazk.default/extensions.ini     1
  ...
 firefox     /export/home/bu/.mozilla/firefox/suqgjazk.default/secmod.db    3
 firefox     /usr/lib/firefox/browser/chrome/icons/default/default16.png    3
 firefox     /usr/lib/firefox/browser/chrome/icons/default/default32.png    3
  ...
```

La sortida que s'ha indicat es parcial ja que era molt llarga i s'han editat els espais.

Es pot observar com es mostra una columna el nom del procés (*firefox*) , a una altra els camins dels fitxers que s'obrin i les vegades que ho fa.

Exemple 21

Comprobar aplicacions que es van executant amb exit:

```
root@bubuSOL:/tmp/test#  dtrace  -n  'proc:::exec-success  {  trace(curpsinfo->pr_psargs); }'
dtrace: description 'proc:::exec-success ' matched 1 probe
 CPU     ID                    FUNCTION:NAME
   0  10732        exec_common:exec-success   ls
   0  10732        exec_common:exec-success   /usr/bin/sh /usr/bin/clear
   0  10732        exec_common:exec-success   /usr/bin/tput clear
   0  10732        exec_common:exec-success   gnome-about-me
   0  10732        exec_common:exec-success   /usr/lib/e-addressbook-factory
   0  10732        exec_common:exec-success   nautilus  --no-desktop /export/home/bu
```

On s'empra la sonda amb nom exec-succes del module proc.

I la clàusula trace(cursinfo - > pr_psargs) que indica l'execució de la funció trace que permet imprimir punters. L'argument que es passa (cursinfo - > pr_psargs) es una variable interna que descriu el estat del procés on *curinfo* es una estrucutura dades de tipus *psinfo* i *pr_psargs* correspon al nom del procés.

D'aquesta manera es van mostrant els executables que es van cridant, les 2 línies són el que realment executa el shell *bash* quan execut un clear,

Exemple 22

També poden emprar sondes de xarxa, per exemple per llistar la quantitat de paquets que es reben per ip on es mira la sonda receive (rebre) del mòdul tcp i la ip surt del argument 2 de la propia sonda que es tipus ipinfo_t on hi ha l'adreça d'origen (ip_saddr)

```
root@bubuSOL:/tmp/test#   dtrace   -n   'tcp:::receive   /execname=="firefox"/ {@pkts[args[2]->ip_saddr] = count();}'
dtrace: description 'tcp:::receive ' matched 4 probes
^C
  151.101.192.166                                                1
  178.250.0.80                                                   1
 ...
  54.247.107.88                                                  8
  216.58.211.238                                                11
  213.73.40.242                                                 16
root@bubuSOL:/tmp/test#
```

Exemple 23

Permet comprovar per exemple quines funcions d'usuari empren més temps de CPU.

Per això es pot emprar la sonda profile-997hz al predicat que mostreja la CPU a una freqüència donada (997hz) , arg1 es un argument de la sonda perquè miri només l'espai d'usuari i l'agregació agafa el nom del executable, el nom de la funció obtingut amb la funció *ufunc* a partir del cotador de programa en espai d'usuari i el que conta son events.

```
root@bubuSOL:/tmp/test#  dtrace  -n  'profile-997hz  /arg1/  {  @[execname, ufunc(arg1)] = count(); }'
dtrace: description 'profile-997hz ' matched 1 probe
^C

  gnome-terminal          libc.so.1`mutex_unlock_queue              1
  gnome-terminal          libc.so.1`mutex_lock_impl                 1
  dtrace                  libproc.so.1`byaddr_cmp                   8
  dtrace                  libproc.so.1`byaddr_cmp_common            9
  Xorg                    libpixman-1.so.0`pixman_glyph_cache_lookup    11
  dtrace                  libc.so.1`strcmp                          14
  Xorg                    vboxvideo_drv.so`VBoxHGSMIBufferSubmit    17
root@bubuSOL:/tmp/test#
```

Aquesta sonda es útil per perfilació (profiling) ja que ens permet comprovar en que esta ocupada la CPU.

Exemple 24

Un altra exemple es fer la suma del bytes llegits per procés i fitxer:

```
root@bubuSOL:/tmp/test#   dtrace  -n  'fsinfo:::read  {  @[execname,args[0]->fi_pathname] = sum(arg1);
dtrace: description 'fsinfo:::read ' matched 1 probe
^C

  gnome-terminal          /var/tmp/vteOYJ5TY                 624
  gnome-terminal          /var/tmp/vte2QJ5TY                 2610
  VBoxService             /system/volatile/utmpx             5580
```

En aquest cas s'hja emprat la sonda read del mòdul fsinfo util per obtenir informació del sistema d'arxius. A la clausuala, hi ha un agregat on hi ha el nom del executable (execname), el nom del cami complet de ftixer accedit a travès de args[0]→ fi_pathname i la suma dels bytes llegits.

Exemple 25

Un altre exemple es l'hora de fer distribucions amb quantize. En aquest cas es mostra una distribució de les vegades que un procés concret (PID), em emprat el del *top* que tenia executant-se a una altra finestra, ha demanat via la instrucció malloc() una certa quantitat de memòria per que li sigui assignada.

```
root@bubuSOL:/tmp/test#   dtrace   -n   'pid$target::malloc:entry   {   @   =
quantize(arg0); }' -p 2914
dtrace: description 'pid$target::malloc:entry ' matched 2 probes
^C
           value  ------------- Distribution ------------- count
              16 |                                         0
              32 |@                                        9
              64 |@@@@@@@@@@@                              100
             128 |@@@@@@@@@@@@@@@                          129
             256 |@@@@@                                    40
             512 |@@@@@                                    46
            1024 |@@                                       14
            2048 |@                                        6
            4096 |@                                        6
            8192 |                                         0
           16384 |                                         2
           32768 |                                         0
           65536 |                                         0
          131072 |                                         1
          262144 |                                         0
```

S'ha emprat la sonda entry de la funció malloc del mòdul pid, el target es per referir-se al pid que li passa pel paràmetre -p (2914). A la clàusula hi ha l'argument amb la funció quantize que la que fa la distribució en potencies de 2 emprant l'argument arg0 que es la memòria demanada cada vegada. Es mostra gràficament de manera que es pot veure que la que més vegades s'ha fet (129) es es la de 128.

Hi ha algunes funcions més d'agregació que s'empren:

lquantize que fa una distribució lineal de l'agregació, amb el valor del primer argument que se li passen, després van dos valors que li passen (maxim i mini), i finalment se li passa la mida de cada pasa (les pases son les de la columna esquerra.

La funció d'agregació trunc(x, @a) es per que només tengui en conta els x majors valors.

I es pot emprar la funció normalize() per dividir els valors d'una aggregació per algo, util per passar de bytes a kylobytes.

Exemple 26

Es posa ara un exemple curt d'un *script*.

```
1. #! /usr/sbin/dtrace -s
2. #pragma D option quiet
3. dtrace:::BEGIN
4. {
5.    start = timestamp;
6.    printf("Use PID from running program, firefox for example\n");
7.    printf("Press ctrl+c to stop tracing...\n");
8.
9.}
10.
11.tcp:::receive
12.   /pid==$1/
13.{
14.   printf("IP Packets received for 8%d\n",$1);
15.   printf("%-20s      %8s\n","Source IP Address","Rcvd");
16.   @pkts[args[2]->ip_saddr] = count();
17.   printa("%-20s      %@8d\n",@pkts);
18.}
19.
20.dtrace:::END
21.{
22.   printf("Total time: %d secs", (timestamp - start) / 1000000000);
23.}
```

Línia 1: Ès l'interpret de la línia, que és /usr/sbin/dtrace.

Línia 2: Li diu a D que mostri el que imprimeix normalment (com que ha trobat sondes).

Línia 5: On es troba la sonda especial que indica l'inici.

Línia 5: Posa a una variable un segell de temps.

Línies 6 i 7: Missatge per indicar que s'ha de posar un PID com argument del *script*.

Línia 11: És on hi ha la descripció de la sonda que emprada, com el exemple 22.

Línia 12: És on hi ha el predicat que ens diu que agafi el PID del argument 1 de la línia de comandes.

Línies 14,15,16 i 17: És on hi ha la impressió amb format adequat de missatge i capçalera i on hi ha la variable d'agregació que agafa i suma els paquets.

Línia 20 a 23 es on hi ha la sonda especial que indica el final i imprimeix el temps.

I la sortida:

Abans de pitjar Ctril+c

```
root@bubuSOL:/export/home/bu# ./testTCPcount.d 3039
Use PID from running program, firefox for example
Press ctrl+c to stop tracing...
```

i després de navega un poc

```
IP Packets received for 83039
Source IP Address             Rcvd
108.161.188.192                  1
173.241.240.143                  1
199.59.149.243                   1
216.58.211.226                   1
217.12.15.83                     1
23.111.9.32                      1
23.23.128.175                    1
23.39.100.178                    1
54.208.253.242                   1
93.184.220.29                    1
94.31.29.64                      1
52.22.102.34                     2
52.94.220.16                     2
216.58.214.168                   3
54.228.255.29                    3
74.125.206.157                   3
52.35.19.240                     6
151.101.193.140                  8
109.70.36.240                   28
^C
Total time: 87 secs
root@bubuSOL:/export/home/bu#
```

La sortida surten les adreces IP amb el recompte de paquets i el temps que ha passat, a vegades es repeteix la llista de paquets però això només es un exemple.

11.6.1 Scripts existents (DTrace Toolkit).

Al directori /usr/dtrace/DTT hi ha un gran nombre de scripts com exemple i útils tant per monitoritzar com de cara optimitzar o depurar el sistema, es comentaran alguns perquè tenen molt d'interès, formen part del Dtrace Toolkit de Brendan Gregg.

Al directori hi trobam més directoris separats en seccions on es troben scripts CPU, disc(disk) , sistemes de fitxers (FS),nucli (kernel), memòria (Mem) processos (Proc)etc. Son sobretot per comprovar el rendiment i perfilar el sistema ja que permet obtenir estadístiques con el de interrupcions dins la CPU, el accés a disc,bytes llegits/escrits per un PID. També hi ha scripts per monitoritzar llenguatges com Java, Python PHP.

Es mostraran alguns exemples

Exemple 27

opensnoop permet veure informació dels fitxers oberts per una aplicació, amb l'opció -h surt l'ajuda,

```
root@bubuSOL:/usr/dtrace/DTT# ./opensnoop firefox


UID    PID COMM          FD PATH
 100   1382 firefox      75 /etc/X11/fontpath.d/dejavu:pri=42/DejaVuSans.ttf
   0    593 nscd         12 /etc/resolv.conf
100     1382 firefox     75 /export/home/bu/.cache/mozilla/firefox/suqgjazk
.default/Cache/1/00/602D1d01
...
```

Aixi es veu quin usuari i procés obrir el fitxer identificat per el FD i el seu camí, pot obtenir més opcions com arguments o errors si n'hi ha.

Exemple 28

iotop es per mirar qui esta llegit o escriguent del disc com si fos el *top* per processos.

```
root@bubuSOL:/usr/dtrace/DTT# ./iotop 1
017 Jan 20 21:39:48,  load: 1.90,  disk_r:    321 KB,  disk_w:   7912 KB


  UID    PID   PPID CMD              DEVICE  MAJ MIN D            BYTES
  100   1413      1 firefox          cmdk0    54   1 R           329216
    0      5      0 zpool-rpool      cmdk0    54   1 W          8102400
```

En aquest cas el argument 1 es perquè mostri resultats cada segon, hi ha altres *scripts* com iosnoop o iopattern que treuen més informació d'entrada/sortida

Exemple 29

Altra script interessant es el dtruss, truss es l'equivalent al strace i algo més que hi ha a Solaris ja que strace no hi és, és a dir, fa el seguiment de les crides de sistema

```
oot@bubuSOL:/usr/dtrace/DTT# ./dtruss echo "hola"
hola
SYSCALL(args)           = return
systeminfo(0x5, 0xF500E6C0, 0x101)       = 6 0
mmap(0x0, 0x8000, 0x3)        = -237830144 0
sysconfig(0x6, 0x0, 0xF1D30000)          = 4096 0
memcntl(0xF1D4A000, 0x8C4C, 0x4)         = 0 0
...
fstatat64(0x1, 0x0, 0xF500E8D0)          = 0 0
write(0x1, "hola\n\0", 0x5)        = 5 0

root@bubuSOL:/usr/dtrace/DTT#
```

Es veu com fa accions com strace on podem veure les crides amb els arguments i el retorn.

Exemple 30

Hi ha una eina interessant que es diu dapptrace que en principi fa el mateix que ltrace

```
root@bubuSOL:/usr/dtrace/DTT/Proc#  ./dapptrace  tar  -cfvj  /usr/tmp/tt.tar.bz2
/usr/tmp/tt
 ALL(args)        = return
Compressing '/usr/tmp/tt.tar.bz2' with '/usr/bin/bzip2'...
a /usr/tmp/tt 1K
-> _start(0xFCEB2BAC, 0xFCEB2BB0, 0xFCEB2BB6)
-> __fsr(0x4, 0xFCEB2AEC, 0xFCEB2B00)
<- __fsr = 122
-> main(0x4, 0xFCEB2AEC, 0xFCEB2B00)
-> init_yes(0x8052BD4, 0xF7400364, 0x1)
<- init_yes = 559
-> assert_string(0xFCEB2BB6, 0x80561E0, 0xFCEB2AB8)
<- assert_string = 41
-> compress_malloc(0xF, 0x8078960, 0x7C4)
<- compress_malloc = 37
```

Aqui es veuen les crides a funcions de llibreries.

12. Alternatives

Ja s'ha dit abans que existeixen bastantes eines de traça se'n anomenaran algunes:

ftrace: És una eina de traça de funcions del kernel de Linux que suporta molts de punts d'instrumentació propis i forma part d'aquest, també permet instrumentació dinàmica. La seva interfície funciona com un sistema de fitxers. S'empra tant per depurar com per comprovar latències de cara a optimitzar i comprovar un sistema. Existeixen frontends com trace-cmd.

perf_events: és una eina de traça emprada sobretot per comprovar el rendiment ja que permet instrumentat els contadors de rendiment de les CPU, a més de punts d'instrumentació estàtics, kprobes (sondes del nucli) i uprobes (sondes d'usuari per instrumentació dinàmica). Està integrada amb el kernel i te un subsistema de events

Systemtap: Vol ser lo que és Dtrace a Solaris i macOS, te un llenguatge de scripting propi i suporta tant punts de instrumentació estàtics com kprobes i uprobes (varen sortir d'aquí) està molt suportat a Red Hat però encara falta que es poleix-qui una mica.

Sysdig: És una eina de traça que le que te d'especial es que suporta contenidor tipo docker, permet empra el llenguatge de scripting Lua

LTTng: És una eina de dades que que es més com una gravadora i desprès es pot mira les dades amb un GUI, permet instrumentació dinàmica.

eBPF: Va sorgir del monitoratge de xarxa, però s'ha anat ampliant. Funciona com una màquina virtual dins el nucli que pot córrer programes de manera molt eficient. Empra quasi qualsevol sonda que hi ha ja que es molt flexible,,es pot programar el necessari dins la maquina virtual perquè ho faci. També empra el subsistema de events de perf i els punt d'instrumentació de ftrace. Encara no està completament suportat al nucli però va avançant. Hi ha un frontend que es diu bcc que es un conjunt d'eines que permet la programació en python i Lua. Sembla bastant prometedor.

13. Conclusions

13.1 Comparativa

Després de provar i examinar les eines de traça strace, ltrace i DTrace puc dir sense dubte que la més completa i flexible es la DTrace, així i tot s'ha de tenir en conta que examinant-les s'ha vist com han anat evolucionant en temps les eines de traça ja que aquesta es la més moderna.

La raó principal que tenc per considerar-la millor es per la seva flexibilitat ja que gracies al llenguatge D i a les sondes permet dur a terme anàlisis molt generals si només volem cercar estadístiques de rendiments o molt específics si estem cercant un error. El pitjor problema que té es que només està forçament suportada als sistema operatius de macOS , OS X i Solaris, mentre que a FreeBSD el suport es parcial i a Linux, apenes cap. Això i tot Oracle intenta portar-la a Linux.

Les eines strace i ltrace tenen en això darrer avantatge, ja que estan suportades a Linux. Aquest és molt més emprat, especialment a entorns de servidors proporcionant informació útil especialment a la hora de trobar i diagnosticar un problema d'una manera simple. Lo dolent és que fa temps que es varen deixar de seguir desenvolupant i per tant segons evoluciona el nucli de Linux hi ha coses que poden no anar bé com ha passat durant els les probes fetes.

En tots els casos per emprar les eines s'ha de tenir un coneixement bastant profund de l'arquitectura de computadors i de sistemes operatius, una vegada es te aquest coneixement la corba d'aprenentatge no és molt alta, especialment a strace i ltrace on només són un parell de comandes amb opcions, en el cas de DTrace és una mica més inclinada però també es poden obtenir molts més resultats.

13.2 Opinió Personal

Acadèmicament, a nivell de coneixements, m'he topat amb un tema que sabia que existia però en el qual no havia entrat en profunditat, la veritat es m'ha semblat força interessant. Sempre m'ha agradat sabre exactament que fa l'ordinador i ara disposo de moltes més eines per saber-ho.

Jo era més de llegir logs i mira el top, i intentar fer anar un depurador per cercar alguns problemes, però ara em serà més fàcil. No m'importaria gens dedicar-me professionalment a diagnosticar errors, o trobar perquè un programa va molt lent i trobar-los de manera tan concreta com amb aquest tipus d'eines.

Un del problemes tècnics que he tengut ha estat a l'hora de provar el ltrace que vaig acabar provant-ho a una altra màquina on vaig obtenir resultats diferents, però bé al manco me'n vaig adonar conta, encara que em va fer perdre temps.

Un altra problema ha estat a l'hora de empra el DTrace que vaig acabar per posar una màquina solaris on sí han anat bé les coses, per què a la FreeBSD no anava com calia. En aprendre una mica de D i com anaven les sondes no ha estat molt difícil però t'hi tenies que fixar, està molt ben documentat.

He confirmat problemes que sabia que tenia com la meva poca traça en documentar. M'ha resultat més difícil el documentar i posar-ho tot en un cert ordre que no entendre la part tècnica, a vegades algunes eines no ajuden.

Un altra problema m'he trobat és que soc molt mal planificador. No crec que arribi ser mai un gran gestor de projectes encara que tampoc m'atreu gaire, preferesc està amb el metall. Treballo millor amb deadlines curts com a les PACs, per això també pens que se'm dona bé arreglar petites coses però no fer algun projecte gran, jo tot sol, si hi una data molt llunyana. Per això vull agrair al meu consultor la flexibilitat que m'ha donat.

A la banda personal és que aquest tema oferit, un poc per casualitat, m'ha resultat molt interessant, ben segur que en miraré més coses especialment del eBPF/bcc ja que pareix que hi ha futur. Com a mínim he aprés més maneres de poder diagnosticar problemes de cara al desenvolupament de controladors i comprovar problemes.

El Treball de fi de Grau m'ha suposat molta feina però m'ha alegrat veure que el tema m'ha interessat encara que a vegades em frustrava, sobretot a l'hora de fer que el LibreOffice o l'aplicació per fer el diagrama de Gantt em fes cas. O no trobar la documentació adequada. Tot i això es un treball que m'ha estat agradable fer-ho.

Pep Rincón i Qués

Palma, Gener de 2017

14. Bibliografia

14.1 Strace:

http://www.thegeekstuff.com/2011/11/strace-examples

http://strace.git.sourceforge.net/git/gitweb.cgi?p=strace/strace;a=tree

https://blog.packagecloud.io/eng/2016/02/29/how-does-strace-work/

https://blogs.oracle.com/ksplice/entry/strace_the_sysadmin_s_microscope

http://www.brendangregg.com/blog/2014-05-11/strace-wow-much-syscall.html

http://www.linuxjournal.com/article/6100

14.2 ltrace:

Cespedes. Juan. Proyecto Fin de Carrera Trazado de Llamadas a Funciones de Biblioteca dinámica. Universidad Politécnica de Madrid, 2003

https://blog.packagecloud.io/eng/2016/03/14/how-does-ltrace-work/

http://techblog.rosedu.org/ltrace.html

https://www.kernel.org/doc/ols/2007/ols2007v1-pages-41-52.pdf

https://blog.packagecloud.io/eng/2016/03/14/how-does-ltrace-work/

https://www.cyberciti.biz/tips/linux-shared-library-management.html

http://techblog.rosedu.org/ltrace.html

http://wm-help.net/lib/b/book/1434608941/256

http://notes.secretsauce.net/notes/2014/06/25_ltrace-filtering-details.html

https://anonscm.debian.org/gitweb/?p=collab-maint/ltrace.git

14.3 Dtrace:

DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD, Brendan Gregg, Jim Mauro; Prenice Hall 2011

http://www.brendangregg.com/dtrace.html

http://dtrace.org/blogs/ahl/2011/10/10/oel-this-is-not-dtrace/

http://docs.oracle.com/cd/E19253-01/819-5488/gbwaz/index.html

https://www.freebsd.org/doc/handbook/dtrace-using.html

http://www.brendangregg.com/DTrace/dtrace_oneliners.txt

http://dtrace.org/guide/chp-intro.html

https://www.joyent.com/blog/bruning-questions-debugging

http://dtrace-discuss.opensolaris.narkive.com/WRvG9x3j/emulating-truss-u-with-dtrace-impossible

http://dtrace.org/guide/bookinfo.html

https://www.bignerdranch.com/blog/hooked-on-dtrace-part-1/

https://blogs.oracle.com/mws/entry/dtrace_inlines_translators_and_file

http://www.solarisinternals.com/wiki/index.php/DTrace_Topics_Intro

http://www.scalingbits.com/dtrace/largescripts

14.4 Altres:

http://www.cs.stevens.edu/~jschauma/810/elf.html

http://reverseengineering.stackexchange.com/questions/1992/what-is-plt-got

https://www.technovelty.org/linux/plt-and-got-the-key-to-code-sharing-and-dynamic-libraries.html

https://www.youtube.com/user/brendangregg/videos

https://en.wikipedia.org/wiki/Unix_signal

http://rainbow.chard.org/2011/10/02/debug-like-a-sysadmin/

https://filippo.io/linux-syscall-table/

http://refspecs.linuxfoundation.org/

https://greek0.net/elf.html

http://www.linuxjournal.com/article/1060

http://www.pango.org/

http://www.airs.com/blog/archives/41

https://www.ibm.com/support/knowledgecenter/SSRTLW_9.1.0/com.ibm.rational.llc.ui.doc/topics/c_dynstat_instrument.html

http://www.oracle.com/technetwork/server-storage/solaris11/downloads/vm-templates-2245495.html

Totes les planes web tenien accés a dia 17/01/2017