Πανεπιστήμιο Δυτικής Αττικής
Σχολή Μηχανικών
Τμήμα Μηχανικών Πληροφορικής και Υπολογιστών

Διπλωματική Εργασία

Μελέτη και ανάπτυξη τεχνικών για την παρακολούθηση και την αποσφαλμάτωση της εκτέλεσης εντολών σε υπολογιστικά συστήματα

Χρήστος Μαργιώλης
Α.Μ. 19390133

Εισηγητής: Παναγιώτης Καρκαζής

Διπλωματική Εργασία

Μελέτη και ανάπτυξη τεχνικών για την παρακολούθηση και την αποσφαλμάτωση της εκτέλεσης εντολών σε υπολογιστικά συστήματα

Χρήστος Μαργιώλης
Α.Μ. 19390133

Εισηγητής:

Παναγιώτης Καρκαζής, ¡++¿

Εξεταστική επιτροπή:

¡++¿

Ημερομηνία εξέτασης: ¡++¿

Δήλωση συγγραφέα διπλωματικής εργασίας

¡++¿

Ο Δηλών
¡++¿

Ευχαριστίες

¡++¿

Περίληψη

Η εργασία αποσκοπεί στην μελέτη τεχνικών που χρησιμοποιούνται στην ανάλυση και αποσφαλμάτωση λογισμικού μέσω της καταγραφής και παρακολούθησης των εντολών που εκτελούνται σε ένα επεξεργαστή. Στο πλαίσιο της διπλωματικής θα σχεδιαστεί και θα αναπτυχθεί επέκταση του εργαλείου DTrace η οποία θα παρέχει την δυνατότητα παρακολούθησης οποιασδήποτε μεμονωμένης εντολής assembly εντός μιας δεδομένης συνάρτησης του πυρήνα του λειτουργικού συστήματος FreeBSD.

¡++¿

Abstract

¡++¿

Περιεχόμενα

1 Συντομογραφίες **6**

2 Εισαγωγή στην παρακολούθηση (tracing) **6**

3 Εισαγωγή στο DTrace **6**
3.1 Providers . . . 6
3.1.1 FBT . . . 6
3.2 Probes . . . 6
3.3 Γλώσσα D . . . 6
3.3.1 Δομή εντολών DTrace . . . 6
3.3.2 Global μεταβλητές . . . 6
3.3.3 CTF . . . 6
3.3.4 DIF . . . 7
3.4 Επικοινωνία δεδομένων μεταξύ kernel και userspace . . . 7
3.5 Scripts υψηλότερου επιπέδου . . . 7
3.6 Παραδείγματα . . . 7

4 Inline tracing **7**
4.1 Τι είναι inline συναρτήσεις . . . 7
4.2 Γιατί είναι δύσκολη η παρακολούθηση inline συναρτήσεων . . . 7

5 kinst **7**
5.1 Χρήση . . . 8
5.2 Ενορχήστρωση εντολών assembly . . . 8
5.3 «Τραμπολίνο» . . . 8
5.3.1 Διάταξη . . . 9
5.3.2 Σημειώσεις υλοποίησης για x86-64 . . . 9
5.3.3 Σημειώσεις υλοποίησης για ARM64 και RISC-V . . . 9
5.4 Παρακολούθηση inline συναρτήσεων . . . 9
5.4.1 Πρότυπο DWARF . . . 9
5.4.2 Πρότυπο ELF . . . 9
5.4.3 Υπολογισμός ορίων κλήσεων . . . 9
5.4.4 Εύρεση καλούσας συνάρτησης . . . 9
5.4.5 Υπολογισμός των offsets `entry` και `return` . . . 9

6 Πειράματα **9**

7 Συμπεράσματα **10**

8 Βιβλιογραφία **11**

9 Παράρτημα **12**

1 Συντομογραφίες

¡++¿

2 Εισαγωγή στην παρακολούθηση (tracing)

Με τον όρο tracing εννοούμε την παρακολούθηση της ροής εκτέλεσης ενός προγράμματος σε πραγματικό χρόνο (real-time), με σκοπό την εξαγωγή δεδομένων, που είναι χρήσιμα κυρίως στην αποσφαλμάτωση (debugging) του προγράμματος, στην καλύτερη κατανόση της ροής εκτέλεσης, καθώς και στην εύρεση τυχόν σημείων που προκαλούν καθυστερήσεις (bottleneck). Κατά μία έννοια το tracing μπορεί να θεωρηθεί ως μία καλύτερη και πιο δυναμική μορφή καταγραφής πληροφοριών (logging), διότι δίνεται η δυνατότητα στον χρήστη να εξάγει αφηρημένες πληροφορίες από ένα πρόγραμμα, χωρίς να χρειάζεται αναγκαστικά η συγγραφή περαιτέρω πηγαίου κώδικα μέσα στο πρόγραμμα και η επαναμεταγλώττισή του, κάτι το οποίο, στην περίπτωση μεγάλων προγραμμάτων όπως ο πυρήνας ενός λετουργικού συστήματος, τείνει να είναι μία χρονοβόρα διαδικασία.

3 Εισαγωγή στο DTrace

¡++¿

3.1 Providers

¡++¿

3.1.1 FBT

¡++¿

3.2 Probes

¡++¿

3.3 Γλώσσα D

¡++¿

3.3.1 Δομή εντολών DTrace

Οι εντολές DTrace ορίζονται ως εξής:

```
<provider>:<module>:<function>:<name> /<predicate>/ {<actions>}
```

¡++¿

3.3.2 Global μεταβλητές

¡++¿

3.3.3 CTF

¡++¿

3.3.4 DIF

Για να εισαχθεί ένα D script στον πυρήνα του λειτουργικού συστήματος, χρησιμοποιείται η κωδικοποίηση DIF (D Intermediary Format). Η κωδικοποίηση αυτή έχει ως στόχο την μεταγλώττιση του D script σε μία ενδιάμεση byte κωδικοποίηση (byte code), η οποία μπορεί να εκτελεστεί από το DTrace εντός του kernel. Αφού το script έχει μεταγλωττιστεί, φορτώνεται στο kernel από το DTrace, και εκτελείται για τα probes που έχουν οριστεί όταν αυτά ενεργοποιηθούν.

Αντίστοιχα στο Linux, χρησιμοποείται η κωδικοποίηση `eBPF`.

¡++¿

3.4 Επικοινωνία δεδομένων μεταξύ kernel και userspace

Η επικονωνία δεδομένων μεταξύ kernel και userspace επιτυχάνεται μέσω ενός ζευγαριού buffers, διαφορετικό για κάθε πυρήνα (per-CPU). Ανά πάσα στιγμή ο ένας buffer είναι ανενεργός και ο άλλος ενεργός, και κάθε ένα δευτερόλεπτο, οι buffers ανταλλάσονται, οπότε αυτός που ήταν ενεργός τώρα είναι ανενεργός και αυτός που ήταν ανενεργός τώρα είναι ενεργός. Στον ενεργό buffer γράφονται τα δεδομένα από το kernel τα οποία οποία με την ανταλλαγή των buffers στέλνονται και γίνονται διαθέσιμα προς διάβασμα από το userspace μέσω του πλέον ανενεργού buffer. Με άλλα λόγια, το kernel γράφει στον ενεργό buffer, και το userspace διαβάζει από τον ανενεργό. Αυτό έχει ως αποτέλεσμα την συνεχή ροή δεδομένων από το kernel προς το userspace, διότι ταυτόχρονα γίνονται διάβασμα και καταγραφή δεδομένων.

¡++¿

3.5 Scripts υψηλότερου επιπέδου

¡++¿

3.6 Παραδείγματα

¡++¿

4 Inline tracing

¡++¿

4.1 Τι είναι inline συναρτήσεις

¡++¿

4.2 Γιατί είναι δύσκολη η παρακολούθηση inline συναρτήσεων

¡++¿

5 kinst

Ο `kinst` είναι ένας νέος DTrace provider, ο οποίος αρχικά δημιουργήθηκε με στόχο να αντιμετωπίσει τους περιορισμούς του FBT (βλέπε ενότητα 3.1.1), δηλαδή την έλλειψη δυνατότητας παρακολούθησης inline συναρτήσεων, καθώς και γενικότερα της πιο εξειδικευμένής παρακολουθήσης πέραν της αρχής ή/και του τέλους μίας συνάρτησης.

Στον πηγαίο κώδικα του FreeBSD, ο κώδικας του `kinst` βρίσκεται στο `sys/cddl/dev/kinst` [2] και υποστηρίζεται στις αρχιτεκτονικές x86-64 (AMD64), ARM64 και RISC-V.

Το όνομα "kinst" είναι εμπνευσμένο από την δημοσίευση των Tamches & Miller [3], στην οποία κάνουν αναφορά στο πειραματικό εργαλείο παρακολούθησης που ανέπτυξαν, ως "KernInst".

¡++¿

5.1 Χρήση

Ο `kinst` δέχεται τις παρακάτω τρεις συντάξεις:

- `kinst::<function>:`
- `kinst::<function>:<instruction>`
- `kinst::<inline_function>:<entry|return>`

Με την πρώτη σύνταξη παρακολουθούνται *όλες* οι εντολές assembly της συνάρτησης `function`. Για παράδειγμα:

¡++¿

Με την δεύτερη σύνταξη παρακολουθείται εντός της συνάρτησης `function` μόνο η εντολή που βρίσκεται στο offset που ορίστηκε στο πεδίο `instruction`. Για παράδειγμα:

¡++¿

Με την τρίτη σύνταξη παρακολουθείται η αρχή (`entry`) ή/και το τέλος (`return`) μίας inline συνάρτησης `inline_function`. Για παράδειγμα:

```
# dtrace -n 'kinst::critical_enter:return'
dtrace: description 'kinst::critical_enter:return' matched 130 probes
CPU     ID                    FUNCTION:NAME
  1  71024                spinlock_enter:53
  0  71024                spinlock_enter:53
  1  70992                uma_zalloc_arg:49
  1  70925    malloc_type_zone_allocated:21
  1  70994                uma_zfree_arg:365
  1  70924             malloc_type_freed:21
  1  71024                spinlock_enter:53
  0  71024                spinlock_enter:53
  0  70947         _epoch_enter_preempt:122
  0  70949           _epoch_exit_preempt:28
  ^C
```

Listing 1: Inline tracing

¡++¿

5.2 Ενορχήστρωση εντολών assembly

¡++¿

5.3 «Τραμπολίνο»

Τραμπολίνο (trampoline) ονομάζεται ένα *εκτελέσιμο* κομμάτι μνήμης, το οποίο χρησιμοποιείται ως περιοχή αναπήδησης (jump) για την εκτέλεση κάποιας εντολής και επιστροφής στην κανονική ροή του προγράμματος.

Ο `kinst` κάνει χρήση τραμπολίνων για την εκτέλεση των εντολών των οποίων παρακολουθεί, σε αντίθεση με τον FBT (3.1.1), ο οποίος τις «μιμείται»/υλοποιεί (emulation). Ωστόσο, μία βασική διαφορά του FBT είναι ότι εφόσον έχει την δυνατότητα να παρακολουθήσει μόνο την αρχή και το τέλος μίας συνάρτησης, τα οποία σημεία ορίζονται με συγκεκριμένες εντολές assembly για κάθε αρχιτεκτονική [1], το σύνολο των εντολών που πρέπει να υλοποίησει ο FBT είναι μικρό, καθώς και οι εντολές απλές, με αποτέλεσμα να μην χρειάζεται η επιπλέον πολυπλοκότητα που προσθέτει το τραμπολίνο. Ο `kinst` όμως μπορεί να παρακολουθήσει εν δυνάμει (σχεδόν) όλες τις εντολές assembly για κάθε αρχιτεκτονική που υποστηρίζει, οπότε είναι προφανές πως η προσέγγιση της υλοποίησης εντολών, δεν μπορεί να λειτουργήσει σε αυτήν την περίπτωση. Παρ' όλα αυτά, υπάρχουν μερικές εξαιρέσεις που θα συζητηθούν παρακάτω.

Η χρήση του τραμπολίνου γίνεται ως εξής. ¡++¿

5.3.1 Διάταξη

¡++¿

5.3.2 Σημειώσεις υλοποίησης για x86-64

¡++¿

5.3.3 Σημειώσεις υλοποίησης για ARM64 και RISC-V

¡++¿

5.4 Παρακολούθηση inline συναρτήσεων

¡++¿

5.4.1 Πρότυπο DWARF

¡++¿

5.4.2 Πρότυπο ELF

¡++¿

5.4.3 Υπολογισμός ορίων κλήσεων

¡++¿

5.4.4 Εύρεση καλούσας συνάρτησης

¡++¿

5.4.5 Υπολογισμός των offsets `entry` και `return`

¡++¿

6 Πειράματα

¡++¿

[1] Για παράδειγμα, στην αρχιτεκτονική x86-64, η εντολή `push %rbp` δηλώνει το ξεκίνημα του *προλόγου* μίας συνάρτησης, και η εντολή `ret` το τέλος του *επιλόγου*.

7 Συμπεράσματα

¡++¿

8 Βιβλιογραφία

[1] Illumos Operating System “Dynamic Tracing Guide”. https://illumos.org/books/dtrace

[2] FreeBSD src “kinst” https://cgit.freebsd.org/src/tree/sys/cddl/dev/kinst

[3] Tamches, Ariel & Miller, Barton P. “Fine-Grained Dynamic Instrumentation of Commodity Operating System Kernels”. https://www.usenix.org/legacy/publications/library/proceedings/osdi99/full_papers/tamc

[4] The DWARF Standard. https://dwarfstd.org/

[5] Christos Margiolis “Using DWARF to find call sites of inline functions“. https://margiolis.net/w/dwarf_inline/

[6] Christos Margiolis “Inline function tracing with the kinst DTrace provider”. https://margiolis.net/w/kinst_inline/

[7] Sourcehut, inlinecall(1). https://github.com/christosmarg/inlinecall

[8] Mark Johnston “Introduction to DTrace on FreeBSD”. https://www.youtube.com/watch?v=E06GVdH-LX0

9 Παράρτημα

¡++¿