4η Εργαστηριακή Ασκήσεων
Σχεδιασμός Ενσωματωμένων Συστημάτων

Λεωνίδας Αβδελάς | ΑΜ: 03113182

Άσκηση 1

1.

Χρησιμοποιήσαμε την συγκεκριμένη αρχιτεκτονική, γιατί το guest μηχάνημα έχει αυτή την αρχιτεκτονική. Αν προσπαθούσαμε να τρέξουμε ένα cross-compiled εκτελέσιμο διαφορετικής αρχιτεκτονικής, το εκτελέσιμο θα μας έβγαζε κάποιο σφάλμα, καθώς το εκτελέσιμο περιμένει μια τελείως διαφορετική αρχιτεκτονική όσον αφορά το ISA του συστήματος.

2.

Η βιβλιοθήκη της C που χρησιμοποιήσαμε είναι η glibc, καθώς ταιριάζει περισσότερο στις ανάγκες μας. Υπήρχαν δυνατότητες χρήσης βιβιοθηκών που είναι μικρότερες, αλλά αυτές ή δεν υποστηρίζουν ARM ή δεν υποστηρίζουν το distribution Debian, οπότε δεν μπορούμε να τις χρησιμοποιήσουμε.

3.

Η εντολή `file` μας δίνει τα παρακάτω αποτελέσματα:

> ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0, with debug_info, not stripped

Η εντολή `readelf`, μας δίνει:

```
ELF Header:
Magic: 7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
Class: ELF32
Data: 2's complement, little endian
Version: 1 (current)
OS/ABI: UNIX - System V
ABI Version: 0
Type: EXEC (Executable file)
Machine: ARM
Version: 0x1
Entry point address: 0x1045c
Start of program headers: 52 (bytes into file)
Start of section headers: 14724 (bytes into file)
Flags: 0x5000400, Version5 EABI, hard-float ABI
Size of this header: 52 (bytes)
Size of program headers: 32 (bytes)
Number of program headers: 9
Size of section headers: 40 (bytes)
Number of section headers: 37
Section header string table index: 36
Attribute Section: aeabi
File Attributes
Tag_CPU_name: "7-A"
Tag_CPU_arch: v7
Tag_CPU_arch_profile: Application
Tag_ARM_ISA_use: Yes
Tag_THUMB_ISA_use: Thumb-2
Tag_FP_arch: VFPv3
Tag_Advanced_SIMD_arch: NEONv1
Tag_ABI_PCS_wchar_t: 4
Tag_ABI_FP_rounding: Needed
Tag_ABI_FP_denormal: Needed
Tag_ABI_FP_exceptions: Needed
Tag_ABI_FP_number_model: IEEE 754
Tag_ABI_align_needed: 8-byte
Tag_ABI_align_preserved: 8-byte, except leaf SP
Tag_ABI_enum_size: int
Tag_ABI_VFP_args: VFP registers
Tag_CPU_unaligned_access: v6
Tag_MPextension_use: Allowed
Tag_Virtualization_use: TrustZone
```

Το πρώτο πράγμα που παρατηρούμε είναι ότι η αρχιτεκτονική πάνω στην οποία τρέχει το αρχείο είναι ARM. Ακόμα, λειτουργεί με dynamic linking και το αρχείο της βιβλιοθήκης που καλεί είναι το `/lib/ld-linux-armhf.so.3`. Έχουμε ακόμα

πολλές πληροφορίες για τις CPU για τις οποίες είναι φτιαγμένο αυτό το αρχείο , καθώς και για το ABI, ώστε να γνωρίζει το λειτουργικό λεπτομέρειες για την διαχείρηση πράξεων κλπ.

4.

Καθώς η προτεινούμενη έκδοση του linaro cross-compiler δεν δούλευε, χρησιμοποιήσαμε μια επόμενη έκδοση. Μεταξύ των δύο αρχείων βλέπουμε αρκετή μεγάλη διαφορά στο μέγεθος, με την έκδοση από το linaro compiler να είναι 12002 byte και το custom toolchain 16204. Υποθέτουμε ότι αυτή η διαφορά οφείλεται στην έκδοση του compiler που χρησιμοποιεί ο cross-compiler. Συγκεκριμένα η έκδοση του linaro είναι η 4.8.5 και η έκδοση του custom toolchain είναι 9.2.0. Έτσι, η νεότερη έκδοση θεωρητικά περιέχει περισσότερες πληροφορίες για βιβλιοθήκες και ρυθμίσεις και άρα είναι μεγαλύτερη.

5.

Όπως αναφέραμε παραπάνω με την εντολή file, ο linker κάνει dynamic linking τις βιβλιοθήκες. Έτσι υποθέτει ότι θα βρεί τις ARM βιβλιοθήκες στον φάκελο `lib`. Έτσι παρ΄όλο που η βιβλιοθήκη του host και του target είναι διαφορετικές, το πρόγραμμα υποθέτει ότι θα βρεί την συγκεκριμένη βιβλιοθήκη στο συγκεκριμένο σημείο, ως convetion των UNIX-type systems.

6.

Όπως βλέπουμε και εδώ η διαφορά είναι πολύ μεγάλη. Συγκεκριμένα για το αρχείο από τον linaro cross-compiler έχουμε μέγεθος 2702174 bytes, ενώ για το αρχείο από τον custom cross-compiler 4139036 bytes. Αυτό μπορεί να συμβαίνει για δύο λόγους:

- Αρχικά, μπορεί ο linaro cross-compiler να έχει απενεργοποιήσει μέρη της βιβλιοθήκης κάτα την δημιουργία του, καθώς αυτά μπορεί να θεωρήθηκαν μη-χρήσιμα από τους δημιουργούς.
- Ακόμα, καθώς η διαφορά μεταξύ των δύο εκδόσεων του compiler είναι 5 χρόνια, πολλά πράγματα έχουν αλλάξει και έχουν προστεθεί νέες συναρτήσεις στην glibc, με αποτέλεσμα να μεγαλώσει σε μέγεθος.

7.

Η τροποποίηση της glibc αποδείχτηκε μια αρκετά δύσκολη εργασία. Αποφασίσαμε να προσθέσουμε να προσθέσουμε την συνάρτηση σε μια ήδη υπάρχουσα βιβλιοθήκη συναρτήσεων και συγκεκριμένα στην `stdlib`, καθώς αυτό ήταν αρκετά ευκολότερο. Χρήσιμος οδηγός μας φάνηκε αυτό το documentation. Τα βήματα που ακολουθήσαμε είναι τα παρακάτω:

1. Αρχικά ορίσαμε την συνάρτηση στο αρχείο `stdlib/stdlib.h`.

2. Έπειτα δημιουργήσαμε το αρχείο `mlab_foo.c` και προσθέσαμε την συνάρτηση μας.
3. Προσθέσαμε στο `Makefile` το mlab_foo στις routines, ώστε να γίνει compile το αρχείο μας.
4. Τέλος, για να δουλεύει ο Dynamic Linker έπρεπε να προσθέσουμε την συνάρτηση στο αρχείο Versions, το οποίο όπως καταλάβαμε από το ελάχιστο documentation χρησιμοποιήται κάπως σαν symbol table για τις συναρτήσεις. Αυτό το βήμα εντοπίστηκε τυχαία από συνεργάτη από άλλη ομάδα και μετά χτίσαμε τις πληροφορίες για το λόγο που δουλεύει από το wiki της glibc. Πιο συγκεκριμένα, ένα κομμάτι της λογικής μας βγήκε από αυτό το άρθρο και ένα κομμάτι από της γνώσεις μας στο πώς δουλεύουν οι compilers. Το θέμα αυτό είναι ένα που δεν έχουμε κατανοήσει πλήρως, αλλά θεωρούμε ότι ξεφεύγει αρκετά και από το θέμα του μαθήματος, οπότε αποφασίσαμε να μην επεκταθούμε.

Ακόμα ανακαλύψαμε ότι το qemu έχει συνδέσει τα εκτελέσιμα για ARM με το πρόγραμμα `qemu arm` και έτσι το host μηχάνημας μας μπορεί να τρέχει ARM αρχεία. Περισσότερες πληροφορίες εδώ.

Για τα ερωτήματα έχουμε:

1. Απενεργοποιώντας την παραπάνω λειτουργία, παίρνουμε το παρακάτω error message:

   ```
   bash: ./my_foo: cannot execute binary file: Exec format error
   ```

2. Αν το τρέξουμε στο guest μηχάνημα, θα έχουμε:

   ```
   ./my_foo: relocation error: ./my_foo: symbol mlab_foo, version
   GLIBC_2.4 not defined in file libc.so.6 with link time reference
   ```

   Αυτό το περιμέναμε, αφού ο Dynamic Linker του target μηχανήματος δεν μπορεί να βρει το σύμβολο της συνάρτησης που ψάχνουμε.

3. Κάνοντας static linking της βιβλιοθήκης το πρόγραμμα τρέχει κανονικά, όπως και περιμέναμε, αφού η βιβλιοθήκη περιλαμβάνεται στο αρχείο.

Άσκηση 2

Ακολουθώντας τα βήματα, αντιμετωπίσαμε πρόβλημα όταν εκκαταστήσαμε τον πυρήνα στο target μηχάνημα. Πιο συγκεκριμένα, ενώ αν χρησιμοποιούσαμε το αρχικό kernel το μηχάνημα ξεκινούσε κανονικά, αν δοκιμάζαμε το καινούριο,δεν δούλευε καθόλου, απλά έβγαζε μια μαύρη οθόνη και δεν αντιδρούσε σε καμία εντολή από το τερματικό.

Έπειτα από υπόδειξη συναδέλφου, εντοπίσαμε ότι υπάιτιος ήταν το configuration file. Έτσι πήραμε το configuration file που ήταν στο `/boot` φάκελο του μηχανήματος από πριν (ονομαζόταν συγκεκριμένα `config-3.2.0-4-vexpress`) και το χρησιμοποιήσαμε για να ρυθμίσουμε το compilation του kernel.

Το σύστημα μας δούλεψε κανονικά μετά και το αποτέλεσμα του `uname -a` ήταν:

> Linux debian-armhf 3.16.81 #1 SMP Thu Feb 20 23:39:30 EET 2020 armv7l GNU/Linux

Αυτό που βλέπουμε να αλλάζει ουσιαστικά είναι η έκδοση του λειτουργικού, η οποία πλέον είναι `3.18.81`, ενώ πριν ήταν `3.2.0.4`.

Για να φτιάξουμε το δικό μας system call, ακολουθούμε τα παρακάτω βήματα:

1. Αρχικά, προσθέτουμε ένα φάκελο `hello`, στον κύριο φάκελο του kernel. Εκεί θα προσθέσουμε την συνάρτηση μας.
2. Μέσα δημιουργούμε ένα αρχείο `hello.c` και ένα `Makefile` που θα το μετατρέψει σε object file.
3. Ακόμα προσθέτουμε τον φάκελο του system call μας στο Makefile του kernel στον κανόνα για τα `core-y`.
4. Τώρα πρέπει να προσθέσουμε το interrupt για το system call μας. Αρχικά την προσθέτουμε στο αρχείο `/arch/arm/kernel/calls.S` με κωδικό 386. Εκεί το προσθέτουμε γράφοντας `CALL(sys_hello)`. Εδώ υπολογίζουμε και το padding που πρέπει να προσθέσουμε, το οποίο ισούται με ((NR_SYSCALLS + 3) &~3) − NR_SYSCALLS, άρα 2 στην περίπτωση μας.
5. Μετά την προσθέτουμε στο αρχείο `/arch/arm/include/uapi/asm/unistd.h` γράφοντας `#define __NR_hello (__NR_SYSCALL_BASE+386)`.
6. Ακόμα προσθέτουμε στο πλήθος των system calls σε 388 στο `/arch/arm/include/asm/unistd.h`, σύμφωνα με τον υπολογισμό του padding.
7. Τέλος, προσθέτουμε τον ορισμό της συνάρτησης μας στο `include/linux/syscalls.h`.

Μετά από αυτά τα βήματα, το compilation του προγράμματος έγινε κανονικά. Προσθέσαμε το kernel στον guest με τον ίδιο τρόπο και δοκιμάσαμε το system call μας, με τον δοκιμαστικό κώδικα. Γράφοντας `dmesg | tail` πήραμε

> [266.871655] Greeting from kernel and team no B5.