5η Άσκηση – Σχεδιασμός Ενσωματωμένων Συστημάτων

Ομάδα 12
Φωτεινή Κυριακοπούλου AM 03120182
Ιωάννης Μπουφίδης AM 03120162

Άσκηση 1

1. Το *debian_wheezy_armhf* υποστηρίζει **"Hard Float"**, δηλαδή η εκτέλεση των πράξεων κινητής υποδιαστολής γίνεται χρησιμοποιώντας το **hardware** του επεξεργαστή. Αντιθέτως, το *debian_wheezy_armel* υποστηρίζει **"Soft Float"**, όπου οι πράξεις κινητής υποδιαστολής γίνονται μέσω λογισμικού (**software *emulation***). Το ακρωνύμιο *el* υποδεικνύει ότι η αρχιτεκτονική είναι **little-endian**.

2. Η πλατφόρμα *vexpress-a9* που προσομοιώνεται στον QEMU βασίζεται στον επεξεργαστή **ARM Cortex-A9**. Επομένως, η επιλογή της αρχιτεκτονικής arm-cortexa9_neon-linux-gnueabihf εξασφαλίζει ότι **το παραγόμενο εκτελέσιμο είναι συμβατό με το σύστημα**. Σε περίπτωση που χρησιμοποιούσαμε άλλη αρχιτεκτονική, η πιθανότητα να **μην εκτελείται σωστά το binary** θα ήταν μεγάλη, είτε λόγω **ασυμβατότητας** στο επίπεδο **υλικού** είτε στο επίπεδο **βιβλιοθηκών**.

3. Γράφουμε έναν πολύ απλό κώδικα C (πχ. “Hello, World!”) και αφού το κάνουμε compile με την βοήθεια του crosstool, **μεταφέρουμε το εκτελέσιμο** στο guest μηχάνημα. Αν εκτελέσουμε την εντολή ***ldd*** με όρισμα το **εκτελέσιμο**, θα πάρουμε την παρακάτω έξοδο. Παρατηρούμε ότι το εκτελέσιμο πρόγραμμα χρησιμοποιεί την **libc** (default).

```
root@debian-armhf:~# ldd ./hello
./hello: /lib/arm-linux-gnueabihf/libc.so.6: version `GLIBC_2.34' not found (required by ./hello)
        libc.so.6 => /lib/arm-linux-gnueabihf/libc.so.6 (0x76ec1000)
        /lib/ld-linux-armhf.so.3 (0x76fb8000)
```

   Ωστόσο, βλέπουμε ότι η έκδοση της βιβλιοθήκης που χρησιμοποιεί (**2.34**), **δεν υποστηρίζεται** από τον guest (με *ldd --version* βλέπουμε ότι χρησιμοποιεί την έκδοση **2.19**). Επομένως, πρέπει να κάνουμε configure το crosstool, ώστε να χρησιμοποιεί μια έκδοση της libc **≤2.19**.

4. Όπως φαίνεται και από την παρακάτω εικόνα, οι εντολές *file* και *readelf* παρέχουν διαφορετικές **πληροφορίες που αφορούν την επικεφαλίδα** του εκτελέσιμου αρχείου phods_crosstool.out. Συγκεκριμένα,

   Η εντολή **file** μας δίνει μια γενική περιγραφή του εκτελέσιμου αρχείου:
   A. **ELF 32-bit LSB executable**: Το εκτελέσιμο είναι σε μορφή ELF, 32-bit και χρησιμοποιεί την little-endian αναπαράσταση δεδομένων.
   B. **ARM**: Είναι προσαρμοσμένο για την αρχιτεκτονική ARM.
   C. **EABI5 version 1**: Ακολουθεί το EABI version 5, συμβατό με ARMv7.
   D. **dynamically linked**: Το πρόγραμμα είναι δυναμικά συνδεδεμένο με βιβλιοθήκες.
   E. **interpreter /lib/ld-linux-armhf.so.3**: Χρησιμοποιεί αυτόν τον dynamic linker/loader για να φορτώσει τις δυναμικές βιβλιοθήκες κατά την εκτέλεση.
   F. **for GNU/Linux 3.2.0**: Είναι σχεδιασμένο για να τρέχει σε σύστημα Linux με πυρήνα έκδοσης 3.2.0 ή νεότερη.
   G. **with debug_info**: Περιλαμβάνει πληροφορίες αποσφαλμάτωσης.
   H. **not stripped**: Δεν έχουν αφαιρεθεί τα metadata.

   Η εντολή **readelf** δίνει αναλυτικές πληροφορίες για το ELF header και τα attributes του εκτελέσιμου. Μερικά χαρακτηριστικά που δεν αναφέρονται στο output της file είναι:
   - **OS/ABI**: UNIX System V (το τυπικό ABI για Linux).
   - **Type**: EXEC (εκτελέσιμο αρχείο).
   - **Entry point address**: 0x104e0 (η διεύθυνση εκκίνησης του προγράμματος).

- **Flags**: 0x5000400
  - **hard-float ABI**: Χρήση floating-point μονάδων.

- **Tag_CPU_name: "7-A"**: Το πρόγραμμα προορίζεται για ARMv7-A CPUs.
- **Tag_CPU_arch: v7**: Υποστηρίζει ARMv7 αρχιτεκτονική.
- **Tag_FP_arch: VFPv3**: Υποστηρίζει το Vector Floating Point version 3 (VFPv3).
- **Tag_Advanced_SIMD_arch: NEONv1**: Υποστηρίζει NEON για SIMD επεξεργασία.
- **Tag_ABI_PCS_wchar_t: 4**: wchar_t έχει μέγεθος 4 bytes.
- **Tag_ABI_FP_number_model: IEEE 754**: Χρησιμοποιεί το πρότυπο IEEE 754 για floating-point αριθμούς.
- **Tag_CPU_unaligned_access: v6**: Υποστηρίζει unaligned memory accesses.

5. Προκειμένου να γίνει επιτυχώς το compile, έπρεπε να συμπεριλάβουμε στην εντολή και το flag ***-std=c99***, διότι ο κώδικας χρησιμοποιεί δηλώσεις μεταβλητών μέσα σε βρόχους for, κάτι που υποστηρίζεται μόνο σε C99 και νεότερες εκδόσεις της C. Ο linaro cross-compiler χρησιμοποιεί από προεπιλογή την πιο παλιά έκδοση της C, η οποία δεν υποστηρίζει αυτή τη δυνατότητα.

   Αφού μεταγλωττίσουμε το πρόγραμμα, μπορούμε με την παρακάτω εντολή να συγκρίνουμε το μέγεθος των 2 εκτελέσιμων που προέκυψαν.

```
:~/embd_ask6$ du -h phods_crosstool.out phods_linaro.out
16K     phods_crosstool.out
12K     phods_linaro.out
```

   Παρατηρούμε ότι το εκτελέσιμο που προέκυψε μέσω του **crosstool** είναι **16 KB**, ενώ αυτό που παράχθηκε από το **linaro 12 KB**. Αυτή η διαφορά μεγέθους μπορεί να οφείλεται σε διαφορές στις βιβλιοθήκες, τις επιλογές που χρησιμοποιούνται από τους compilers και το επίπεδο γενίκευσης/βελτιστοποίησης που εφαρμόζει κάθε compiler.

6. Το πρόγραμμα του ερωτήματος 4 εκτελείται σωστά στο target μηχάνημα, καθώς αν το πρόγραμμα κατασκευάζεται με μια παλιότερη έκδοση της glibc (π.χ., από τον cross compiler της Linaro) και εκτελείται σε ένα target μηχάνημα με νεότερη έκδοση, αυτό είναι δυνατό λόγω της **backward compatibility της glibc**. Το πρόγραμμα μπορεί να χρησιμοποιεί τις συνήθεις λειτουργίες της glibc που υποστηρίζονται από την παλαιότερη έκδοση, οπότε η εκτέλεση δεν διακόπτεται.

7. Εάν εκτελέσουμε τα ερωτήματα 4 και 5 με επιπλέον flag *–static* και συγκρίνουμε τα μεγέθη των 2 εκτελέσιμων που προκύπτουν παρατηρούμε το εξής:

```
:~/embd_ask6$ du -h phods_crosstool.out phods_linaro.out
2.5M    phods_crosstool.out
500K    phods_linaro.out
```

   Βλέπουμε ότι το μέγεθος των εκτελέσιμων έχει αυξηθεί δραματικά. Αυτό συμβαίνει, καθώς όταν χρησιμοποιούμε το flag *-static*, ο compiler ζητάει από το πρόγραμμα να κάνει **στατικό linking** των βιβλιοθηκών, αντί για **δυναμικό linking**. Η διαφορά στο μέγεθος των εκτελέσιμων προκύπτει από το στατικό linking, καθώς πλέον **ο compiler ενσωματώνει όλες τις βιβλιοθήκες στον εκτελέσιμο κώδικα**.

8. Αφού προσθέσουμε μία δική μας συνάρτηση *mlab_foo()* στη glibc, δημιουργούμε έναν crosscompiler με τον crosstool-ng που κάνει χρήση της ανανεωμένης βιβλιοθήκης. Στην συνέχεια, δημιουργούμε ένα αρχείο *my_foo.c* στο οποίο κάνουμε χρήση της νέας συνάρτησης που δημιουργήσαμε και το κάνουμε cross compile.
   A. Εάν προσπαθήσουμε να τρέξουμε το εκτελέσιμο που προέκυψε στο **host μηχάνημα**, **θα αποτύχει**. Αυτό συμβαίνει, καθώς **η αρχιτεκτονική που έχουμε θέσει ως target** στον cross compiler είναι **διαφορετική** από την **αρχιτεκτονική του host**.
   B. Αν εκτελέσουμε το πρόγραμμα στο **target μηχάνημα**, **θα «τρέξει» κανονικά**. Βέβαια, αυτό θα συμβεί μόνο υπό την προϋπόθεση ότι έχουμε **προσθέσει την συνάρτηση** και στην glilbc του target μηχανήματος, καθώς **το εκτελέσιμο θα γίνει dynamic linked με εκείνη την βιβλιοθήκη**.
   C. Το πρόγραμμα **θα εκτελεστεί κανονικά**, καθώς πλέον **το εκτελέσιμο περιλαμβάνει όλες τις απαραίτητες συναρτήσεις**.

Άσκηση 2

1. Εκτελώντας την εντολή *uname -a* στο Qemu, πριν και αφότου έχουμε εγκαταστήσει τον νέο πυρήνα, παίρνουμε τα εξής αποτελέσματα:

Παλιός πυρήνας

```
root@debian-armhf:~# uname -a
Linux debian-armhf 3.2.0-4-vexpress #1 SMP Debian 3.2.51-1 armv7l GNU/Linux
```

Νέος πυρήνας

```
root@debian-armhf:~# uname -a
Linux debian-armhf 3.16.84 #2 SMP Thu Jan 2 12:51:37 EET 2025 armv7l GNU/Linux
```

Ανάμεσα στα 2 output παρατηρούμε τις εξής διαφορές:

- Η έκδοση (**όνομα**) του πυρήνα άλλαξε από 3.2.0-4-vexpress σε **3.16.84**.
- **Ο αριθμός έκδοσης** του πυρήνα ανανεώθηκε **από #1 σε #2**, καθώς πρόκειται για την δεύτερη κατασκευή του πυρήνα.
- **Η ημερομηνία και ώρα δημιουργίας** του πυρήνα αναφέρεται στην δεύτερη έκδοση, κάτι που δείχνει ότι είναι μια καινούρια κατασκευή του πυρήνα.

2. Προκειμένου να υλοποιήσουμε το ζητούμενο system call, αρχικά δημιουργήσαμε ένα subdirectory στον φάκελο linux-source-3.16 του host, το οποίο περιέχει τον παρακάτω κώδικα C που υλοποιεί το call, καθώς και το αντίστοιχο Makefile.

```c
#include <linux/kernel.h>

asmlinkage long sys_greetk(void) {
    printk(KERN_INFO "Greeting from kernel and team no 12\n");
    return 0;
}
```

Στην συνέχεια, προκειμένου να κάνουμε register την κλήση στο header file προσθέτουμε την γραμμή `asmlinkage long sys_greetk(void);` στο αρχείο *include/linux/syscalls.h* του φακέλου linux-source-3.16.

Προκειμένου να αναθέσουμε στο system call τον αριθμό 386*, συμπληρώνουμε στα παρακάτω αρχεία του directory linux-source-3.16 τις εξής γραμμές:

- *arch/arm/kernel/calls.S*: `/* 386 */ CALL(sys_greetk)`
- *arch/arm/include/asm/unistd.h*: `#define __NR_sys_greetk (__NR_SYSCALL_BASE+386)`
- *arch/x86/syscalls/syscall_32.tbl*: `386 i386 sys_greetk sys_greetk`

*βλέπουμε από τα αρχεία ότι στο τελευταίο system call ανατίθεται ο αριθμός 385

Συμπληρώνοντας το φάκελο /greetk στην γραμμή του *Makefile* που βρίσκεται στο *linux-source-3.16* `core-y += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ greetk/`, συμπεριλαμβάνουμε το directory /greetk.

Προκειμένου να μεταγλωττίσουμε τον νέο πυρήνα, ακολουθούμε τις οδηγίες της εκφώνησης.

3. Προκειμένου να ελέγξουμε την σωστή υλοποίηση του system call *sys_greetk*, γράφουμε το παρακάτω C πρόγραμμα (*greetk_test.c*):

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/syscall.h>
#include <errno.h>

// Ο αριθμός του system call (αντικαταστήστε τον αν είναι διαφορετικός)
#ifndef __NR_sys_greetk
#define __NR_sys_greetk 386
#endif

int main() {
    // Κλήση του system call
    long result = syscall(__NR_sys_greetk);

    if (result == -1) {
        // Αν υπάρχει σφάλμα, εκτυπώστε τον κωδικό και την περιγραφή του
        perror("syscall sys_greetk failed");
        return 1;
    }

    printf("System call sys_greetk executed successfully, result: %ld\n", result);
    return 0;
}
```

Αν μεταγλωττίσουμε το πρόγραμμα και το τρέξουμε στο guest μηχάνημα, θα λάβουμε την παρακάτω έξοδο:

```
root@debian-armhf:~# ./greetk_test
System call sys_greetk executed successfully, result: 0
```