

# ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ



## ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ & ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

### Σχεδιασμός Ενσωματωμένων Συστημάτων

#### 9ο εξάμηνο

### **3η Εργαστηριακή Άσκηση**

Χειμερινό εξάμηνο 2016-2017

#### **Ομάδα 15**

Ασημακόπουλος Κωνσταντίνος  
Χορτομάνης Ιωάννης

03112419  
03110825

# ΑΣΚΗΣΗ 3

## Cross-compiling προγραμμάτων για ARM αρχιτεκτονική

### Άσκηση 1

**1. Γιατί χρησιμοποιήσαμε την αρχιτεκτονική arm-cortexa9\_neon-linux-gnueabihf; Τι μπορεί να συνέβαινε αν χρησιμοποιούσαμε κάποια άλλη αρχιτεκτονική από το list-samples όταν θα τρέχαμε ένα cross compiled εκτελέσιμο στον QEMU και γιατί;**

Η αρχιτεκτονική arm-cortexa9\_neon-linux-gnueabihf χρησιμοποιήθηκε γιατί ανταποκρίνεται στην αρχιτεκτονική του εικονικού μας συστήματος (το οποίο αποτελείται από επεξεργαστή A9 αρχιτεκτονικής ARM ο οποίος υποστηρίζει εντολές “neon” αλλά και κινητή υποδιαστολή). Σε περίπτωση που επιλέγαμε κάποιο άλλο είδος αρχιτεκτονικής, η διαδικασία δεν θα εμφάνιζε κάποιο πρόβλημα. Ωστόσο, οι μεταγλωττίσεις που θα προέκυπταν από τον crosscompiler μας δεν θα ήταν συμβατές με το σύστημα το οποίο κάνει virtualize ο QEMU, επειδή αντιστοιχεί στην αρχιτεκτονική του VM μας (arm).

Εάν χρησιμοποιούσαμε cross compiler για οποιαδήποτε άλλη αρχιτεκτονική, δεν θα μπορούσαμε να εκτελέσουμε κάποιο από τα παραχθέντα εκτελέσιμα στο VM μας. Αυτό συμβαίνει λόγω του ότι ο cross compiler παράγει κώδικα assembly για την επιλεγμένη αρχιτεκτονική. Επομένως είναι αδύνατη η εκτέλεση κώδικα assembly σε αρχιτεκτονική άλλη από αυτή για την οποία προορίζεται.

**2. Ποια βιβλιοθήκη της C χρησιμοποιήσατε στο βήμα 10 και γιατί; (Χρήσιμη εντολή: ldd)**

Όπως φαίνεται και στην ακόλουθη εικόνα, προχωρήσαμε στην “συμβατική” επιλογή της βιβλιοθήκης **glibc**. Αυτό έγινε επειδή είναι η πιο γενική βιβλιοθήκη της C και είναι συμβατή με την βιβλιοθήκη που χρησιμοποιεί ο QEMU. Εξετάστηκαν οι εναλλακτικές επιλογές των βιβλιοθηκών **eglibc** και **uclibc** οι οποίες όμως τελικώς δεν επιλέχθηκαν λόγω εγκαταλελειμμένης πλέον ανάπτυξης και χαμηλής απόδοσης αντίστοιχα.

```
kostas@kostas-SATELLITE-L500-B:~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin$ ldd -v arm-cortexa9_neon-linux-gnueabihf-gcc
linux-vdso.so.1 => (0x00007ffe8ffeb000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fae9d6fc000)
/lib64/ld-linux-x86-64.so.2 (0x0000556080a4a000)

Version information:
./arm-cortexa9_neon-linux-gnueabihf-gcc:
    ld-linux-x86-64.so.2 (GLIBC_2.3) => /lib64/ld-linux-x86-64.so.2
    libc.so.6 (GLIBC_2.3) => /lib/x86_64-linux-gnu/libc.so.6
    libc.so.6 (GLIBC_2.14) => /lib/x86_64-linux-gnu/libc.so.6
    libc.so.6 (GLIBC_2.4) => /lib/x86_64-linux-gnu/libc.so.6
    libc.so.6 (GLIBC_2.11) => /lib/x86_64-linux-gnu/libc.so.6
    libc.so.6 (GLIBC_2.2.5) => /lib/x86_64-linux-gnu/libc.so.6
    libc.so.6 (GLIBC_2.3.4) => /lib/x86_64-linux-gnu/libc.so.6
    /lib/x86_64-linux-gnu/libc.so.6:
    ld-linux-x86-64.so.2 (GLIBC_2.3) => /lib64/ld-linux-x86-64.so.2
    ld-linux-x86-64.so.2 (GLIBC_PRIVATE) => /lib64/ld-linux-x86-64.so.2
```

**3. Χρησιμοποιώντας τον cross compiler που παρήχθει από τον crosstool-ng κάντε compile τον κώδικα phods.c με flags -O0 -Wall -o phods\_crosstool.out από το 2ο ερώτημα της 1ης άσκησης (τον απλό κώδικα phods μαζί με την συνάρτηση gettimeofday()). Τρέξτε στο τοπικό μηχάνημα τις εντολές:**

**file phods\_crosstool.out**

**readelf -h -A phods\_crosstool.out**

**Τι πληροφορίες μας δίνουν οι εντολές αυτές;**

Η εντολή file επιχειρεί να προσδιορίσει το είδος του αρχείου. Η εντολή readelf επιστρέφει πληροφορίες για αρχεία ELF. Το flag -A μας επιστρέφει πληροφορίες σχετικές με την αρχιτεκτονική του συστήματος, ενώ το -h μας επιστρέφει το ELF header του εκτελέσιμου. Στην περίπτωση μας:

```
kostas@kostas-SATELLITE-L500-B:~/Documents/Embedded_2016-17/embedded_exercise_1$ ~/x-tools/arm-cortexa9_neon-linux-gnueabi/bin/arm-cortexa9_neon-linux-gnueabi-gcc -O0 -Wall phods.c -o phods_crosstool.out
kostas@kostas-SATELLITE-L500-B:~/Documents/Embedded_2016-17/embedded_exercise_1$ ls
akiyo0.y  embedded_exercise_1.tar.gz  phods_2a.c  phods_alternate.c  phods_crosstool.out  script_3.sh
akiyo1.y  Makefile                    phods_2.c  phods.c            script_2.sh         script_4.sh
kostas@kostas-SATELLITE-L500-B:~/Documents/Embedded_2016-17/embedded_exercise_1$ file phods_crosstool.out
phods_crosstool.out: 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, not stripped
```

```
kostas@kostas-SATELLITE-L500-B:~/Documents/Embedded_2016-17/embedded_exercise_1$ readelf -h -A phods_crosstool.out
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:               0x103f4
  Start of program headers:          52 (bytes into file)
  Start of section headers:         10960 (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:         8
  Size of section headers:           40 (bytes)
  Number of section headers:         37
  Section header string table index: 34
Attribute Section: aeabi
File Attributes
  Tag_CPU_name: "Cortex-A9"
  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
```

**4. Χρησιμοποιώντας τον cross compiler που κατεβάσατε από το site της [linaro](http://linaro.org) κάντε compile τον ίδιο κώδικα με το ερώτημα 3. Βλέπετε διαφορά στο μέγεθος των δύο παραγόμενων εκτελέσιμων; Αν ναι, γιατί;**

Υπάρχει μια διαφορά της τάξης του 50% ανάμεσα στα δύο εκτελέσιμα (12.4 kB για την περίπτωση του crosstool-ng και 8.2 kB για την περίπτωση χρήσης του Linaro). Αυτό δεν θα έπρεπε να συμβαίνει καθώς παρά τα όσα αναγράφονται στην εκφώνηση (ότι δηλαδή ο Linaro χρησιμοποιεί την βιβλιοθήκη newlibc) διαπιστώθηκε πως και οι δύο cross-compilers φαίνεται να χρησιμοποιούν την ίδια βιβλιοθήκη (glibc όπως είδαμε και στις δύο περιπτώσεις μέσω της εντολής ldd). Αυτό μας κάνει να υποπτευόμαστε πως στην περίπτωση του linaro η newlibc βιβλιοθήκη (η οποία είναι βελτιστοποιημένη για ενσωματωμένα συστήματα και παράγει μικρότερο κώδικα) ενδεχομένως χρησιμοποιείται “κρυμμένη” (ενσωματωμένη στην glibc ίσως;).

**5. Γιατί το πρόγραμμα του ερωτήματος 4 εκτελείται σωστά στο target μηχανήμα εφόσον κάνει χρήση διαφορετικής βιβλιοθήκης της C;**

Το πρόγραμμα του προηγούμενου ερωτήματος εκτελείται επιτυχώς στο target μηχανήμα, διότι αυτή είναι η χρήση του crosscompiler, τον οποίο χρησιμοποιήσαμε για να παράξουμε το εκτελέσιμο. Προσφέρει τη δυνατότητα σε μηχανήματα που τρέχουν πυρήνα μίας A αρχιτεκτονικής, να μεταγλωττίσουν για μία B, μη συμβατή αρχιτεκτονική. Αυτό γίνεται με την δυναμική σύνδεση των κατάλληλων βιβλιοθηκών. Όλα αυτά βέβαια με δεδομένο πως γίνεται χρήση διαφορετικής βιβλιοθήκης της C, όπως αναφέρθηκε στο προηγούμενο ερώτημα.

```
root@debian-armhf:~/askisi3# ./phods_crosstool1.out
149570
root@debian-armhf:~/askisi3# ./phods_crosstool.out
182059
root@debian-armhf:~/askisi3# █
```

**6. Εκτελέστε τα ερωτήματα 3 και 4 με επιπλέον flag -static. Το flag που προσθέσαμε ζητάει από τον εκάστοτε compiler να κάνει στατικό linking της αντίστοιχης βιβλιοθήκης της C του κάθε compiler. Συγκρίνετε τώρα τα μεγέθη των δύο αρχείων. Παρατηρείτε διαφορά στο μέγεθος; Αν ναι, που οφείλεται;**

Παρατηρούμε πως η προσθήκη της σημαίας -static οδηγεί σε πολύ μεγαλύτερο αρχείο (η χωρητικότητα του παραγόμενου εκτελέσιμου από τα 507.8 KB “εκτοξεύεται” στα 4.1MB). Αυτό συμβαίνει λόγω του ότι πλέον όλες οι απαραίτητες βιβλιοθήκες ενσωματώνονται μέσα στο εκτελέσιμο κατά τη διάρκεια της σύνδεσης, μεγαλώνοντας υπερβολικά το μέγεθός του. Χωρίς την χρήση της σημαίας -static, οι απαραίτητες βιβλιοθήκες φορτώνονται δυναμικά κατά την διάρκεια εκτέλεσης, επομένως το μέγεθος του εκτελέσιμου προκύπτει σημαντικά μικρότερο.

7. Έστω ότι προσθέτουμε μία δική μας συνάρτηση `m1ab_foo()` στη `glibc` και δημιουργούμε έναν `cross-compiler` με τον `crosstool-ng` που κάνει χρήση της ανανεωμένης `glibc`. Δημιουργούμε ένα αρχείο `my_foo.c` στο οποίο κάνουμε χρήση της νέας συνάρτησης που δημιουργήσαμε και το κάνουμε `cross compile` με `flags -Wall -O0 -o my_foo.out`

A. Τι θα συμβεί αν εκτελέσουμε το `my_foo.out` στο `host` μηχανήμα;

B. Τι θα συμβεί αν εκτελέσουμε το `my_foo.out` στο `target` μηχανήμα;

C. Προσθέτουμε το `flag -static` και κάνουμε `compile` ξανά το αρχείο `my_foo.c`. Τι θα συμβεί τώρα αν εκτελέσουμε το `my_foo.out` στο `target` μηχανήμα;

A. Το αρχείο `myfoo.out` είναι δημιουργημένο για εκτέλεση σε σύστημα αρχιτεκτονικής ARM οπότε δεν θα εκτελεστεί στο `host` μηχανήμα το οποίο έχει 8086 αρχιτεκτονική.

B. Το αρχείο δεν θα καταφέρει να εκτελεστεί στο `target` μηχανήμα, μιας και η `glibc` του δεν έχει κάπου δηλωμένη την `m1ab_foo()`.

C. Με την προσθήκη της σημαίας `-static`, το πρόγραμμά μας θα εκτελεστεί κανονικά στο `target` μηχανήμα, μιας και οι απαραίτητες βιβλιοθήκες θα συμπεριληφθούν μέσα στο εκτελέσιμό μας αφού πλέον η σύνδεση θα γίνει στατικά.

Γενικότερα (αυτό αφορά το σύνολο της άσκησης) εγκαταστάθηκαν όποια πακέτα απαιτήθηκαν κατά την διάρκεια της διαδικασίας εκτέλεσης των διαδοχικών βημάτων.

## **Άσκηση 2**

1. Εκτελέστε `uname -a` στον `qemu` εφόσον έχετε εγκαταστήσει τον νέο πυρήνα και σημειώστε το όνομα του νέου σας πυρήνα.

Παρατηρούμε ότι το όνομα του νέου μας πυρήνα είναι πλέον το **3.2.84**.

```
root@debian-armhf:~# uname -a
Linux debian-armhf 3.2.84 #2 SMP Sun Jan 29 15:12:10 EET 2017 armv7l GNU/Linux
```

2. Προσθέστε στον πυρήνα του `linux` ένα καινούριο `system call` που θα χρησιμοποιεί την συνάρτηση `printk` για να εκτυπώνει στο `log` του πυρήνα την φράση `"Greeting from kernel and team no %d"` μαζί με όνομα της ομάδας σας.

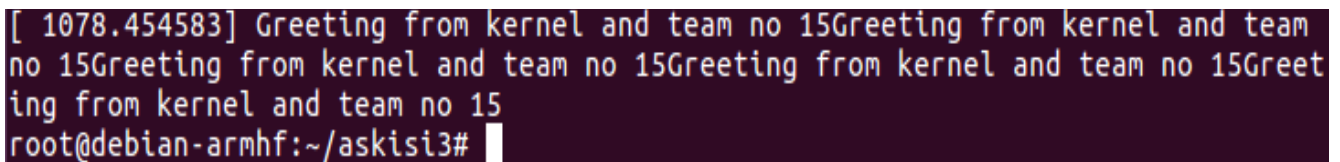
Στο συμπιεσμένο αρχείο επισυνάπτονται και τα αρχεία όπου τροποποιήθηκαν για την εισαγωγή του `system call`.

### 3. Γράψτε ένα πρόγραμμα σε γλώσσα C το οποίο θα κάνει χρήση του **system call** που προσθέσατε.

Με εκτέλεση του κάτωθι προγράμματος και κατόπιν με χρήση της εντολής **dmesg** στο terminal λάβαμε τα ακόλουθα αποτελέσματα (σημειωτέον ότι είμαστε η ομάδα **15**):

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/syscall.h>
#define HELLO 378

int main()
{
    syscall(HELLO);
    return 0;
}
```



```
[ 1078.454583] Greeting from kernel and team no 15Greeting from kernel and team
no 15Greeting from kernel and team no 15Greeting from kernel and team no 15Greet
ing from kernel and team no 15
root@debian-armhf:~/askisi3#
```

Στην αρχή δοκιμάσαμε να μεταγλωττίσουμε τον καινούριο kernel με τον “δικό μας” μεταγλωττιστή αλλά η επιλογή αυτή δεν “περπάτησε” αφού προέκυψαν αρκετά προβλήματα. Έτσι επιστρέψαμε και κάναμε την διαδικασία από την αρχή χρησιμοποιώντας τον **linaro cross\_compiler** οπότε η διαδικασία ολοκληρώθηκε απροβλημάτιστα.

Για την μεταγλώττιση του πυρήνα με το νέο system call μας δεν απαιτούνται διαφορετικά βήματα από αυτά που χρειάζονται για το χτίσιμο του αρχικού πυρήνα.

Για την προσθήκη του system call χρειάστηκε να τροποποιηθούν τα επισυναπτόμενα αρχεία **calls.S** (δήλωση νέου system call υπ' αριθ. 378), **unistd.h** (define **\_\_NR\_hello** στην τιμή 378) και **sys\_arm.c** (προσθήκη του “χαιρετισμού” της ομάδας).