



ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός Ενσωματωμένων Συστημάτων

9ο εξάμηνο

3η Εργαστηριακή Άσκηση

Βάρδια 1 Ομάδα 14

Πάυλος Καπούτσης 03110080
Παναγιώτης Μπουγουλιάς 03112025

Άσκηση 1

1. Γιατί χρησιμοποίησαμε 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;

Χρησιμοποίησαμε την αρχιτεκτονική arm-cortexa9_neon-linux-gnueabihf γιατί ανήκει στην οικογένεια αρχιτεκτονικών που υποστηρίζει το εικονικό μηχάνημα που προσομοιώνεται στο QEMU. Η ιδέα είναι να αναπτύξουμε και να αποσφαλματώσουμε το πρόγραμμά μας στο τοπικό μηχάνημα, να το κάνουμε compile με τον cross compiler για την αρχιτεκτονική που εν τέλει θα τρέξει το ενσωματωμένο σύστημα (ARM v9), και αυτός είναι ο λόγος που δεν θέλαμε κάποια άλλη αρχιτεκτονική από το list-samples.

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

Με την εντολή ldd θα φανούν οι shared libraries που χρειάζεται κάποιο εκτελέσιμο για να τρέξει, οπότε στην παρακάτω εικόνα φαίνεται ότι χρησιμοποιούμε την GLIBC

```
pavlos@pc ~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin$ ldd -v arm-cortexa9_neon-linux-gnueabihf-gcc
linux-vdso.so.1 => (0x00007ffca8af0000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fc9b4d0c000)
/lib64/ld-linux-x86-64.so.2 (0x00007fc9b50d6000)

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
pavlos@pc ~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin$
```

Στην πραγματικότητα επιλέγουμε την glibc επειδή είναι η πιο ευρέως χρησιμοποιούμενη και γνωστή, που σημαίνει ότι έχει τεσταριστεί και έχουν διορθωθεί αρκετά bugs με το πέρασμα των χρόνων. Η πρώτη έκδοση Pre-release βγήκε το 1988-01-1 και μέχρι σήμερα το development είναι ενεργό. Πηγή:

[https://sourceware.org/glibc/wiki/Glibc Timeline](https://sourceware.org/glibc/wiki/Glibc%20Timeline)

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;

Κάνουμε compile το αρχείο phods.c και μετά εκτελούμε την εντολή file και readelf:

```
pavlos@pc ~/enswmatwmeno ~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin/arm-cortexa9_neon-linux-gnueabihf-gcc -Wall -O0 phods.c -o phods_crosstool.out
phods.c: In function 'main':
phods.c:165:38: warning: unused variable 'j' [-Wunused-variable]
    motion_vectors_y[N/B][M/B], i, j;
                                   ^
phods.c:165:35: warning: unused variable 'i' [-Wunused-variable]
    motion_vectors_y[N/B][M/B], i, j;
                                   ^
pavlos@pc ~/enswmatwmeno 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
pavlos@pc ~/enswmatwmeno
```

```
pavlos@pc ~/enswmatwmeno 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:              16028 (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
```

Το πρόγραμμα file προσπαθεί να καταλάβει για ποιο σκοπό χρησιμοποιείται το συγκεκριμένο αρχείο. Στην περίπτωση μας έχει καταλάβει από το περιεχόμενο του ότι το αρχείο είναι ένα εκτελέσιμο (ELF: executable and linkable format) των 32-bit , LSB δηλαδή για little-endian αρχιτεκτονική και ARM.

Το πρόγραμμα readelf δίνει περισσότερες πληροφορίες για τα εκτελέσιμα ELF. Συγκεκριμένα το flag -A επιστρέφει πληροφορίες σχετικά με την αρχιτεκτονική του συστήματος και το -h το ELF header του εκτελέσιμου.

4. Χρησιμοποιώντας τον cross compiler που κατεβάσατε από το site της linaro κάντε compile τον ίδιο κώδικα με το ερώτημα 3. Βλέπετε διαφορά στο μέγεθος των δύο παραγόμενων εκτελέσιμων; Αν ναι, γιατί;

Το link που δινόταν στην άσκηση δεν ανταποκρινόταν στην αρχιτεκτονική της έκδοσης Ubuntu που είχαμε οπότε κατεβάσαμε τον latest Linaro από αυτό το link: <https://releases.linaro.org/components/toolchain/binaries/latest/arm-linux-gnueabi/hf/> .

Παρακάτω φαίνεται η εκτέλεση του compile με τον linaro:

```
pavlos@pc ~/Downloads/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/hf.tar_FILES/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/hf/bin ./arm-linux-gnueabi-gcc -O0 -Wall ~/enswmatwmeno/phods.c -o phods_linaro.out
/home/pavlos/enswmatwmeno/phods.c: In function 'main':
/home/pavlos/enswmatwmeno/phods.c:165:38: warning: unused variable 'j' [-Wunused-variable]
    motion_vectors_y[N/B][M/B], i, j;
                                   ^
/home/pavlos/enswmatwmeno/phods.c:165:35: warning: unused variable 'i' [-Wunused-variable]
    motion_vectors_y[N/B][M/B], i, j;
                                   ^
pavlos@pc ~/Downloads/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/hf.tar_FILES/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/hf/bin
```

Παρατηρούμε ότι το εκτελέσιμο που παράχθηκε είναι 13K σε σχέση με το αρχικό που ήταν 18K

```
pavlos@pc ~/Downloads/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/hf.tar_FILES/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/hf/bin ls -alh | grep phods_linaro
-rwxrwxr-x 1 pavlos pavlos 13K Φεβ 17 20:03 phods_linaro.out
```

```
pavlos@pc ~/enswmatwmeno ls -alh | grep phods_crosstool
-rwxrwxr-x 1 pavlos pavlos 18K Φεβ 17 15:44 phods_crosstool.out
```

Χρησιμοποιώντας την εντολή `readelf -a -W` και για τα 2 εκτελέσιμα μπορέσαμε να δούμε τα object files (.o) που θα χρησιμοποιηθούν για να παραχθεί το τελικό εκτελέσιμο.

Για το `crosstool-ng`:

Στην πρώτη εικόνα βλέπουμε από το output της εντολής ότι χρειάστηκαν αυτά τα 2 object files για το link:

```
43: 00000000 0 FILE LOCAL DEFAULT ABS /home/pavlos/x-tools/arm-cortexa9-neon-linux-gnueabi/arm-cortexa9-neon-linux-gnueabi/sysroot/usr/lib/crti.o
44: 00010430 0 NOTYPE LOCAL DEFAULT 12 $a
45: 00010430 0 FUNC LOCAL DEFAULT 12 call_weak_fn
46: 0001044c 0 NOTYPE LOCAL DEFAULT 12 $d
47: 0001035c 0 NOTYPE LOCAL DEFAULT 10 $a
48: 00010eb8 0 NOTYPE LOCAL DEFAULT 13 $a
49: 00000000 0 FILE LOCAL DEFAULT ABS /home/pavlos/x-tools/arm-cortexa9-neon-linux-gnueabi/arm-cortexa9-neon-linux-gnueabi/sysroot/usr/lib/crtn.o
```

και το μέγεθος τους:

```
pavlos@pc ~/x-tools/arm-cortexa9-neon-linux-gnueabi/arm-cortexa9-neon-linux-gnueabi/sysroot/usr/lib$ ls -alh crt*.o
-r--r--r-- 1 pavlos pavlos 2,1K Φεβ  4 20:40 crt*.o
pavlos@pc ~/x-tools/arm-cortexa9-neon-linux-gnueabi/arm-cortexa9-neon-linux-gnueabi/sysroot/usr/lib$ ls -alh crtn.o
-r--r--r-- 1 pavlos pavlos 1,7K Φεβ  4 20:40 crtn.o
pavlos@pc ~/x-tools/arm-cortexa9-neon-linux-gnueabi/arm-cortexa9-neon-linux-gnueabi/sysroot/usr/lib$
```

Για τον `linaro`:

Βλέπουμε τα αντίστοιχα αρχεία:

```
43: 00000000 0 FILE LOCAL DEFAULT ABS /home/pavlos/Downloads/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/bin/../arm-linux-gnueabi/libc/usr/lib/crti.o
44: 00010468 0 NOTYPE LOCAL DEFAULT 13 $a
45: 00010468 0 FUNC LOCAL DEFAULT 13 call_weak_fn
46: 00010484 0 NOTYPE LOCAL DEFAULT 13 $d
47: 000103a0 0 NOTYPE LOCAL DEFAULT 11 $a
48: 00010ad0 0 NOTYPE LOCAL DEFAULT 14 $a
49: 00000000 0 FILE LOCAL DEFAULT ABS /home/pavlos/Downloads/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/bin/../arm-linux-gnueabi/libc/usr/lib/crtn.o
```

και το μέγεθος τους:

```
pavlos@pc ~/Downloads/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/arm-linux-gnueabi/libc/usr/lib$ ls -alh crt*.o
-rw-r--r-- 1 pavlos pavlos 2,2K Νοέ 19 17:40 crt*.o
pavlos@pc ~/Downloads/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/arm-linux-gnueabi/libc/usr/lib$ ls -alh crtn.o
-rw-r--r-- 1 pavlos pavlos 1,8K Νοέ 19 17:40 crtn.o
pavlos@pc ~/Downloads/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/arm-linux-gnueabi/libc/usr/lib$
```

Βλέπουμε λοιπόν ότι υπάρχουν διαφορετικά μεγέθη στα object-files. Επίσης ένας άλλος παράγοντας είναι ότι ο κάθε compiler δεσμεύει αλλιώς τη μνήμη (memory allocated at compile time).

Πάντως και τα 2 εκτελέσιμα χρησιμοποιούν `GLIBC_2.4`

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

Και οι 2 compilers κάνουν χρήση της glibc όπως είδαμε από το αποτέλεσμα του readelf στο πάνω ερώτημα αλλά όπως αναφέρει και το site:

<https://wiki.linaro.org/WorkingGroups/ToolChain/FAQ>

Does Linaro gcc toolchain target glibc or eglibc? Due to the end of development of eglibc, we have transitioned to glibc beginning version 2.20.

```
root@debian-armhf:~# ./phods_crosstool.out
190395
root@debian-armhf:~# ./phods_linaro.out
151389
root@debian-armhf:~#
```

Και τα 2 προγράμματα εκτελούνται σωστά.

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;

Εκτελούμε το compile με τον crosscompiler ng και παρατηρούμε ότι τώρα το μέγεθος είναι 4.0MB:

```
pavlos@pc > ~/enswmatwmeno > ~/x-tools/arm-cortexa9_neon-linux-gnueabi/bin/arm-cortexa9_neon-linux-gnueabi-gcc -O0 -Wall -static phods.c -o phods_crosstool_static.out
phods.c: In function 'main':
phods.c:165:38: warning: unused variable 'j' [-Wunused-variable]
    motion_vectors_y[N/B][M/B], i, j;
                                   ^
phods.c:165:35: warning: unused variable 'i' [-Wunused-variable]
    motion_vectors_y[N/B][M/B], i, j;
                                   ^
pavlos@pc > ~/enswmatwmeno > ls -alh phods_crosstool_static.out
-rwxrwxr-x 1 pavlos pavlos 4,0M Φεβ 17 21:51 phods_crosstool_static.out
pavlos@pc > ~/enswmatwmeno
```

Εκτελούμε το compile με τον linaro και παρατηρούμε ότι τώρα το μέγεθος είναι 3.9MB:

```

pavlos@pc > ~/Downloads/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/bin > ./arm-linux-
gnueabi-gcc -O0 -Wall -static ~/enswmatwmeno/phods.c -o phods_linaro_static.out
/home/pavlos/enswmatwmeno/phods.c: In function 'main':
/home/pavlos/enswmatwmeno/phods.c:165:38: warning: unused variable 'j' [-Wunused-variable]
    motion_vectors_y[N/B][M/B], i, j;
                                   ^
/home/pavlos/enswmatwmeno/phods.c:165:35: warning: unused variable 'i' [-Wunused-variable]
    motion_vectors_y[N/B][M/B], i, j;
                                   ^
pavlos@pc > ~/Downloads/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/bin > ls -alh phod
s_linaro_static.out
-rwxrwxr-x 1 pavlos pavlos 3,9M Φεβ 17 21:49 phods_linaro_static.out
pavlos@pc > ~/Downloads/gcc-linaro-7.2.1-2017.11-x86_64_arm-linux-gnueabi/bin >

```

Ο λόγος που αυθήθηκαν αντίστοιχα τα μεγέθη των 2 αρχείων είναι ότι τώρα πια οι συναρτήσεις της C που χρειάζονται έχουν ενσωματωθεί στο τελικό εκτελέσιμο.

Η διαφορά στα μεγέθη των 2 αρχείων αναλύονται στην απάντηση του ερωτήματος 3.

7. Έστω ότι προσθέτουμε μία δική μας συνάρτηση `mlab_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 AMD64 αρχιτεκτονική.

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

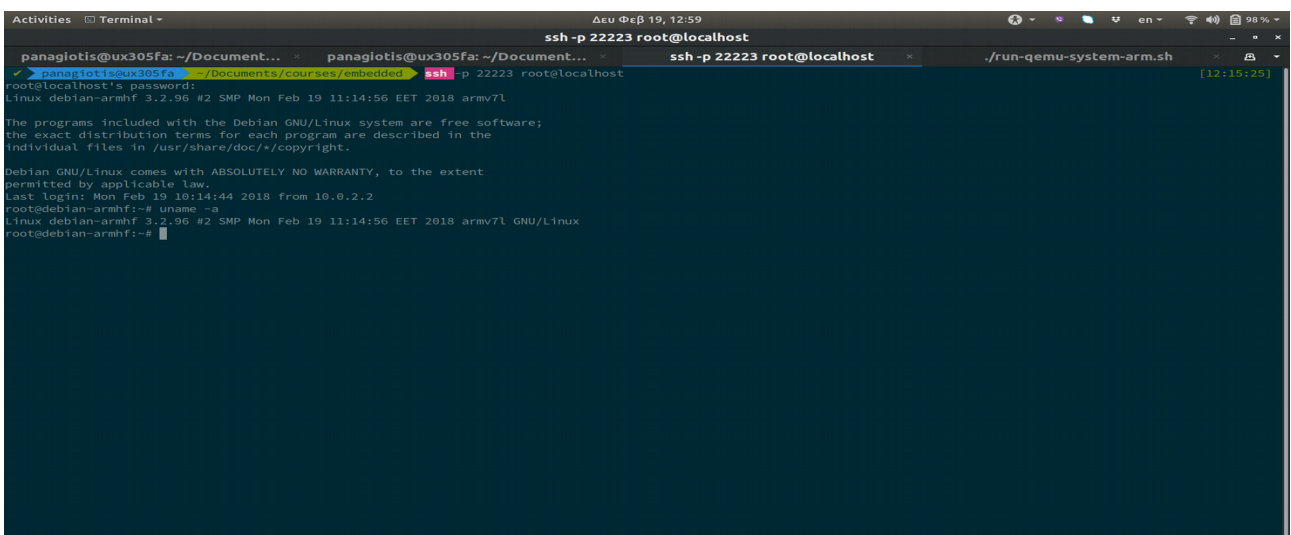
2η Άσκηση

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



```
Activities Terminal
Δευ Φεβ 19, 12:33
ssh -p 22223 root@localhost
sftp-P 22223 root@localhost x ssh -p 22223 root@localhost x panagiotis@ux305fa: ~/Document... x ./run-qemu-system-arm.sh x
root@debian-armhf:~# dpkg -i
linux-headers-3.2.96_3.2.96-2_armhf.deb linux-image-3.2.96_3.2.96-2_armhf.deb linux-libc-dev_3.2.96-2_armhf.deb
root@debian-armhf:~# dpkg -i linux-headers-3.2.96_3.2.96-2_armhf.deb
Selecting previously unselected package linux-headers-3.2.96.
(Reading database ... 28427 files and directories currently installed.)
Unpacking linux-headers-3.2.96 (from linux-headers-3.2.96_3.2.96-2_armhf.deb) ...
Setting up linux-headers-3.2.96 (3.2.96-2) ...
root@debian-armhf:~# dpkg -i linux-image-3.2.96_3.2.96-2_armhf.deb
Selecting previously unselected package linux-image-3.2.96.
(Reading database ... 38751 files and directories currently installed.)
Unpacking linux-image-3.2.96 (from linux-image-3.2.96_3.2.96-2_armhf.deb) ...
Setting up linux-image-3.2.96 (3.2.96-2) ...
update-initramfs: Generating /boot/initrd.img-3.2.96
root@debian-armhf:~# dpkg -i linux-libc-dev_3.2.96-2_armhf.deb
Selecting previously unselected package linux-libc-dev.
(Reading database ... 40440 files and directories currently installed.)
Unpacking linux-libc-dev (from linux-libc-dev_3.2.96-2_armhf.deb) ...
Setting up linux-libc-dev (3.2.96-2) ...
root@debian-armhf:~# ls /boot
config-3.2.0-4-vexpress initrd.img initrd.img-3.2.96 System.map-3.2.0-4-vexpress vmlinuz vmlinuz-3.2.96
config-3.2.96 initrd.img-3.2.0-4-vexpress lost+found System.map-3.2.96 vmlinuz-3.2.0-4-vexpress
root@debian-armhf:~#
```

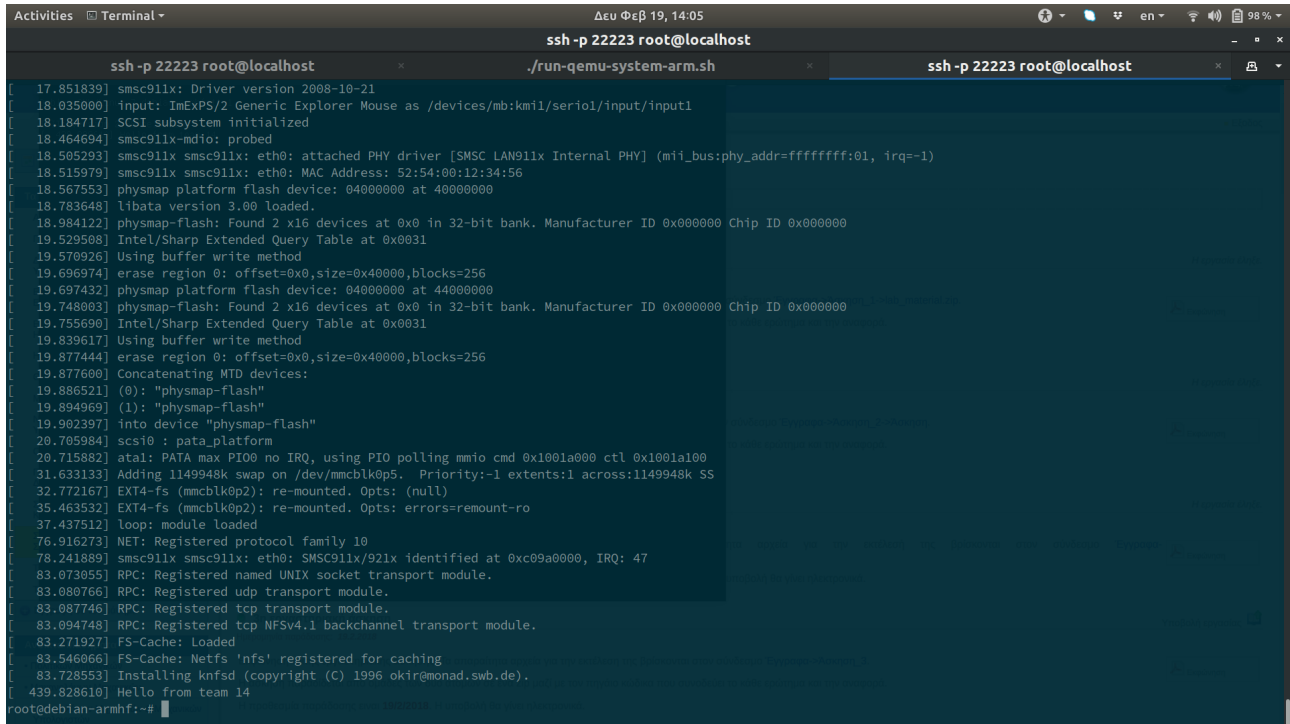
Παραπάνω φαίνεται μόλις εκτελέσουμε τις εντολές `dpkg -i {package}.deb`, όπου `package = {linux-headers, linux-image, linux-glibc}`, που δημιουργήθηκαν με το cross compilation του πυρήνα με τον `linaro`, που αντιστοιχούσε στη `host` και τη `guest` αρχιτεκτονική των μηχανημάτων που χρησιμοποιήθηκαν (https://releases.linaro.org/components/toolchain/binaries/4.9-2017.01/arm-linux-gnueabi/gcc-linaro-4.9.4-2017.01-x86_64-arm-linux-gnueabi.tar.xz). Στη συνέχεια κατεβάσαμε στο `host` τα `{initrd.img,vmlinuz}-3.2.96` και τα δώσαμε ως ορίσματα στο `boot` του `debian` του `guest`. Μετά εκτελέσαμε στο `guest` το `uname -a`:



```
Activities Terminal
Δευ Φεβ 19, 12:59
ssh -p 22223 root@localhost
panagiotis@ux305fa: ~/Document... x panagiotis@ux305fa: ~/Document... x ssh -p 22223 root@localhost x ./run-qemu-system-arm.sh x
root@localhost's password:
Linux debian-armhf 3.2.96 #2 SMP Mon Feb 19 11:14:56 EET 2018 armv7l
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Feb 19 10:14:44 2018 from 10.0.2.2
root@debian-armhf:~# uname -a
Linux debian-armhf 3.2.96 #2 SMP Mon Feb 19 11:14:56 EET 2018 armv7l GNU/Linux
root@debian-armhf:~#
```


Για την εγκατάσταση του system call:

1. Αντικαθιστούμε τα αρχεία linux/arch/arm/kernel/calls.S, linux/arch/arm/kernel/sys_arm.c, arch/arm/include/asm/unistd.h με τα αντίστοιχα που επισυνάπτονται.
2. Ξανακάνουμε compile τον πυρήνα.
3. Εκτελούμε το hello.c που επισυνάπτεται και εκτελούμε την εντολή dmesg για να δούμε το log του πυρήνα.



```
Activities Terminal Δευ Φεβ 19, 14:05
ssh -p 22223 root@localhost
./run-qemu-system-arm.sh
ssh -p 22223 root@localhost

[ 17.851839] smsc911x: Driver version 2008-10-21
[ 18.035080] input: ImExPS/2 Generic Explorer Mouse as /devices/mb:kmil/serial/input/input1
[ 18.184717] SCSI subsystem initialized
[ 18.464694] smsc911x-mdio: probed
[ 18.505293] smsc911x smsc911x: eth0: attached PHY driver [SMSC LAN911x Internal PHY] (mi1_bus:phy_addr=ffffff:01, irq=-1)
[ 18.515979] smsc911x smsc911x: eth0: MAC Address: 52:54:00:12:34:56
[ 18.567553] physmap platform flash device: 04000000 at 40000000
[ 18.783648] libata version 3.00 loaded.
[ 18.984122] physmap-flash: Found 2 x16 devices at 0x0 in 32-bit bank. Manufacturer ID 0x000000 Chip ID 0x000000
[ 19.529508] Intel/Sharp Extended Query Table at 0x0031
[ 19.570926] Using buffer write method
[ 19.696974] erase region 0: offset=0x0,size=0x40000,blocks=256
[ 19.697432] physmap platform flash device: 04000000 at 44000000
[ 19.748003] physmap-flash: Found 2 x16 devices at 0x0 in 32-bit bank. Manufacturer ID 0x000000 Chip ID 0x000000
[ 19.755690] Intel/Sharp Extended Query Table at 0x0031
[ 19.839617] Using buffer write method
[ 19.877444] erase region 0: offset=0x0,size=0x40000,blocks=256
[ 19.877600] Concatenating MTD devices:
[ 19.886521] (0): "physmap-flash"
[ 19.894969] (1): "physmap-flash"
[ 19.902397] into device "physmap-flash"
[ 20.705984] scsi0 : pata_platform
[ 20.715882] ata1: PATA max PIO0 no IRQ, using PIO polling mmio cmd 0x1001a000 ctl 0x1001a100
[ 31.633133] Adding 1149948k swap on /dev/mmcblk0p5. Priority:-1 extents:1 across:1149948k SS
[ 32.772167] EXT4-fs (mmcblk0p2): re-mounted. Opts: (null)
[ 35.463532] EXT4-fs (mmcblk0p2): re-mounted. Opts: errors=remount-ro
[ 37.437512] loop: module loaded
[ 76.916273] NET: Registered protocol family 10
[ 78.241889] smsc911x smsc911x: eth0: SMSC911x/921x identified at 0xc09a0000, IRQ: 47
[ 83.073055] RPC: Registered named UNIX socket transport module.
[ 83.080766] RPC: Registered udp transport module.
[ 83.087746] RPC: Registered tcp transport module.
[ 83.094748] RPC: Registered tcp NFSv4.1 backchannel transport module.
[ 83.271927] FS-Cache: Loaded
[ 83.546066] FS-Cache: Netfs 'nfs' registered for caching
[ 83.728553] Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
[ 439.828610] Hello from team 14
root@debian-armhf:~#
```

Το αποτέλεσμα φαίνεται στην τελευταία γραμμή(Hello from team 14).

Bonus άσκηση: Προγραμματισμός του Zybo Zync-7000 σε FreeRTOS

Αρχικά έπρεπε να γίνει σωστά η διαμόρφωση του FPGA fabric, ώστε να συνδεθεί η είσοδος/έξοδος των επεξεργαστών με το GPIO της πλατφόρμας, το οποίο έγινε με το Xilinx Vivado. Οι οδηγίες που ακολουθήθηκαν είναι στο παρακάτω link:

<https://reference.digilentinc.com/learn/programmable-logic/tutorials/zybo-getting-started-with-zynq/start>

Είναι πολύ σημαντικό να περιγραφεί σωστά το hardware σε block διάγραμμα για να παραχθεί το bitstream που θα προγραμματίσει το FPGA.

Χρησιμοποιήθηκε η xgpio.h για ανάγνωση από τα dip switches και εγγραφή στα leds. Από το FreeRTOS έγινε χρήση των tasks, ένα task/led, η εισαγωγή της καθυστέρησης με την vTaskDelay και με τη βοήθεια του scheduler με την κλήση vTaskStartScheduler ήρθε το επιθυμητό αποτέλεσμα(zybo.c) για τα 4 LEDs.