Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Σχεδιασμός Ενσωματωμένων Συστημάτων
Εργαστηριακή άσκηση 3

Ομάδα Α17
Μαρίνος Δημολιάνης 03111602
Κωνσταντίνος Τουλούπας 03111018

Cross-compiling προγραμμάτων για ARM αρχιτεκτονική

Άσκηση 1η

1) Το crosstool-NG περιέχει pre-configured toolchains που αντιστοιχούν σε διαφορετικές αρχιτεκτονικές. Όπως φαίνεται και από τα screenshots παρακάτω, η αρχιτεκτονική του target μηχανήματος είναι armV7 και λειτουργεί με little endian. Επιλέξαμε την αρχιτεκτονική arm-cortexa9_neon-linux-gnueabihf, της οποίας υποσύνολο είναι η armV7. Σε περίπτωση που επιλέγαμε διαφορετική αρχιτεκτονική, κατά πάσα περίπτωση, δεν θα μπορούσαμε να τρέξουμε στο target μηχάνημα το cross-compiled αρχείο.

```
root@debian-armhf:/proc# cat cpuinfo
Processor       : ARMv7 Processor rev 0 (v7l)
processor       : 0
BogoMIPS        : 434.99

Features        : swp half thumb fastmult vfp edsp neon vfpv3 tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xc09
CPU revision    : 0

Hardware        : ARM-Versatile Express
Revision        : 0000
Serial          : 0000000000000000
root@debian-armhf:/proc# cd
root@debian-armhf:~# cat /proc/cpuinfo
Processor       : ARMv7 Processor rev 0 (v7l)
processor       : 0
BogoMIPS        : 434.99

Features        : swp half thumb fastmult vfp edsp neon vfpv3 tls
CPU implementer : 0x41
CPU architecture: 7
CPU variant     : 0x0
CPU part        : 0xc09
CPU revision    : 0

Hardware        : ARM-Versatile Express
Revision        : 0000
Serial          : 0000000000000000
```

```
root@debian-armhf:~/assembly# lscpu
Architecture:          armv7l
Byte Order:            Little Endian
CPU(s):                1
On-line CPU(s) list:   0
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):             1
```

2) Χρησιμοποιήσαμε την βιβλιοθήκη glibc, η οποία είναι και πιο συνηθισμένη. Μπορούμε να καταλάβουμε ποια βιβλιοθήκη χρησιμοποιήσαμε με χρήση της ldd, όπως φαίνεται παρακάτω

```
semina@semina:~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin$ ldd arm-cortexa9_neon-linux-gnueabihf-gcc
        linux-vdso.so.1 =>  (0x00007fff6d5cb000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f1b607bb000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f1b60b80000)
```

3) Για να κάνουμε compile με χρήση του cross-compiler που φτιάξαμε, πηγαίνουμε στο directory με τα εκτελέσιμα του cross-compiler και εκτελούμε το αντίστοιχο του gcc, όπως φαίνεται παρακάτω

```
semina@semina:~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin$ ./arm-cor
texa9_neon-linux-gnueabihf-gcc -O0 -Wall  /home/semina/ert1.c -o /home/s
emina/phods_crosstool.out
```

Για το αρχείο εξόδου παρατηρούμε ότι έχει μέγεθος 9,4kB. Η εντολή ***file()*** μας δίνει την

πληροφορία ότι πρόκειται για 32-bit binary εκτελέσιμο αρχείο

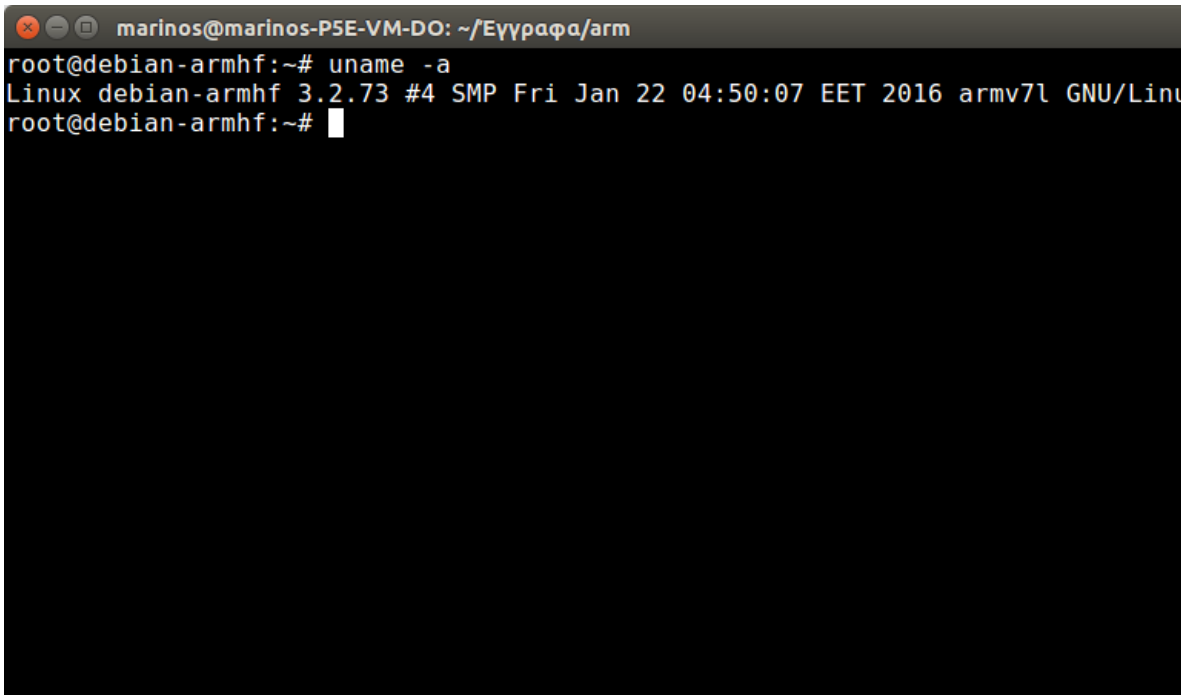
```
semina@semina:~$ file phods_crosstool.out
phods_crosstool.out: ELF 32-bit LSB  executable, ARM, EABI5 version 1 (SYSV), dy
namically linked (uses shared libs), for GNU/Linux 2.6.32, not stripped
```

Η εντολή ***readelf –h - A*** μας δίνει συγκεκριμένες πληροφορίες για την αρχιτεκτονική του target μηχανήματος και για τα headers

4) Ακολουθήσαμε παρόμοια διαδικασία για να κάνουμε compile με τον linaro cross-compiler. Κατά την εκτέλεση του gcc, όμως, παρουσιάστηκε πρόβλημα καθώς ο compiler δεν έβρισκε την βιβλιοθήκη libstdc++.so.6. Η εντολή sudo apt-get install lib32stdc++6 δεν δούλευε, μέχρι που χρησιμοποιήσαμε την apt-get –f install, για διορθώσουμε τα dependencies και έπειτα εγκαταστήσαμε την lib32stdc++6. Το μέγεθος του αρχείο δεν έχει μεγάλη διαφορά, καθώς είναι περίπου 8 kB. Αυτό οφείλεται στο ότι και ο linaro χρησιμοποιεί την ίδια βιβλιοθήκη της C, όπως διαπιστώσαμε κάνοντας χρήση της ***ldd***.
5) Εαν είχαμε δύο διαφορετικές βιβλιοθήκες στους δύο cross-compilers, το binary θα εκτελούταν κανονικά σε κάθε περίπτωση, με την προϋπόθεση ότι το target μηχάνημα θα έχει εγκατεστημένη την αντίστοιχη βιβλιοθήκη.
6) Εκτελούμε τις ίδιες εντολές με παραπάνω, μόνο που προσθέτουμε το flag –static. Τα δύο εκτελέσιμα που προκύπτουν έχουν πλέον πολύ μεγάλο μέγεθος σε σχέση με τα πρώτα. Συγκεκριμένα, το phods_linaro_static.out έχει μέγεθος 508 kB ενώ το phods_crosstool_static.out 618kB. Η μεγάλη διαφορά στο μέγεθος των binaries σε σχέση με τα αρχικά οφείλεται στο ότι κατά το compilation, μέρος του κώδικα της βιβλιοθήκης που χρησιμοποιεί το πρόγραμμα μετατρέπεται σε binary και ενσωματώνεται στο εκτελέσιμο. Ιδιαίτερα στην περίπτωση της glibc που χρησιμοποιούν και οι δύο compilers, παρατηρείται πολύ μεγάλη διαφορά σε μέγεθος.
7)
   a) Το binary που προκύπτει μετά το cross-compilation δεν μπορεί να εκτελεστεί στο host μηχάνημα, γιατί δεν αντιστοιχεί στην αρχιτεκτονική του. Εμφανίζεται μήνυμα λάθους.
   b) Το αρχείο δεν θα εκτελεστεί σωστά, γιατί στο target μηχάνημα δεν θα είναι εγκατεστημένη η glibc στην ανανεωμένη της μορφή. Με δεδομένο ότι το αρχείο θα προσπαθεί να χρησιμοποιήσει την glibc δυναμικά για να βρει την mlab_foo(), το πιθανότερο είναι ότι θα υπάρξει μήνυμα λάθους, κατά την εκτέλεση.
   c) Σε αυτή την περίπτωση το αρχείο θα εκτελεστεί κανονικά. Αυτό οφείλεται στο ότι, όπως αναφέραμε και παραπάνω, η συνάρτηση mlab_foo() και ίσως κάποια άλλα κομμάτια της glibc που χρησιμοποιήσαμε θα μετατραπούν σε binary και θα ενσωματωθούν στο τελικό αρχείο. Έτσι, δεν χρειάζεται το binary να χρησιμοποιήσει καμία βιβλιοθήκη του target μηχανήματος.

Άσκηση 2η

Στόχος της παρούσας άσκησης ήταν να χτίσουμε έναν νέο πυρήνα για το Debian OS , μέσα στον οποίο προσθέσαμε ένα δικό μας system call. Ακολουθώντας τα βήματα που μας δόθηκαν και με τη χρήση του linaro cross compiler κατασκευάσαμε τον πυρήνα που μας ζητήθηκε. Παρακάτω παρατίθεται screenshot από την εκτέλεση της εντολής uname –a στο guest μηχάνημα με τον νέο πυρήνα:



Για την εισαγωγή ενός δικού μας system call στον πυρήνα έπρεπε να ακολουθήσουμε τα παρακάτω βήματα:

- Κάνουμε define τον αριθμό του system call μας στο αρχείο unistd.h (/arch/arm/include/asm/unistd.h).
- Στη συνέχεια ορίζουμε το system call στο αρχείο calls.S όπου πρέπει να βρίσκεται στην αντίστοιχη γραμμή με τον αριθμό που ορίσαμε παραπάνω(arch/arm/kernel/calls.S).
- Έπειτα κρίνεται απαραίτητο να ορίσουμε την συνάρτηση του system call που δημιουργούμε. Για αυτό το λόγο τοποθετούμε σε έναν φάκελο το αρχείο .c με την επιθυμητή συνάρτηση καθώς και ένα Makefile με περιεχόμενο obj-y := (όνομα αρχείου).o .
- Μετά πρέπει να συμπεριλάβουμε στο Makefile του πυρήνα τη δική μας συνάρτηση(ώστε να γίνει compile) , οπότε προσθέτουμε στα path με τους φακέλους που πρέπει να γίνουν compile και τον δικό μας φάκελο με την συνάρτηση που αντιστοιχεί στο system call μας.
- Τέλος προσθέτουμε στην βιβλιοθήκη syscalls.h(include/linux/syscalls.h) την συνάρτηση που αντιστοιχεί στο system call μας.

Μαζί με την παρούσα εργασία επισυνάπτονται τα αρχεία source του πυρήνα που αλλάξαμε , το τελικό image του πυρήνα καθώς και ένα πρόγραμμα σε γλώσσα C που κάνει χρήση του system call μας.