ΕΘΝΙΚΟ ΜΕΤΣΟΒΕΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ. ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
Τομέας Τεχνολογίας Υπολογιστών και Υπολογιστών
Εργαστήριο Μικροϋπολογιστών και Ψηφιακών Συστημάτων

Σχεδιασμός Ενσωματωμένων Συστημάτων
9^{Ο} Εξάμηνο ΗΜΜΥ

6η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Cross-compiling προγραμμάτων για ARM αρχιτεκτονική

Ημ/νια Παράδοσης : 12/01/2025

Ομάδα 22
Παναγιώτης Μπέλσης AM : 03120874
Θεοδώρα Εξάρχου AM : 03120865

Άσκηση 1

Ερωτήματα:

1)

Ποια η διαφορά μεταξύ του καινούριου image που κατεβάσαμε (debian_wheezy_armhf) συγκριτικά με το image που χρησιμοποιήσαμε στην άσκηση 3 (debian_wheezy_armel); Τι υποδεικνύουν τα ακρωνύμια hf και el;

Στην προηγούμενη άσκηση είχαμε κατεβάσει το image debian_wheezy_armel όπου το el σημαίνει Embedded Application Binary Interface Little-endian. Αυτό το image που κατεβάσαμε τώρα έχει κατάληξη hf αντί για el (debian_wheezy_armhf), το οποίο σημαίνει Hard Float και χρησιμοποιεί στο hardware πράξεις με floating point (είναι βελτιστοποιημένο για νεότερες αρχιτεκτονικές ARM, πχ ARMv7). Το image debian_wheezy_armel απευθύνεται σε παλαιότερες και πιο απλές αρχιτεκτονικές ARM, πχ ARMv5 και οι πράξεις με floating point γίνονται στο software.

2)

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 υποστηρίζει όλες τις εντολές του ARMv7-A, όπως και πράξεις με floating point και εντολές NEON. Επίσης ο Cortex-A9 υποστηρίζει το QEMU. Το QEMU έχει ρυθμιστεί για ARMv7-A, οπότε η επιλογή κάποιας άλλης αρχιτεκτονικής θα οδηγούσε στο να μας λείπουν κάποιες χρήσιμες εντολές.
Για παράδειγμα στην προηγούμενη άσκηση που χρησιμοποιήσαμε ARMv5-A δεν μπορούσαμε να χρησιμοποιήσουμε την εντολή udiv για διαίρεση.

3)
Ποια βιβλιοθήκη της C χρησιμοποιήσατε στο βήμα 9 και γιατί;

```
panosbel@lenovo:~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin$ ldd
arm-cortexa9_neon-linux-gnueabihf-gcc
        linux-vdso.so.1 (0x00007ffd6f812000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x000075d6c6200000)
        /lib64/ld-linux-x86-64.so.2 (0x000075d6c65de000)
```

Το εκτελέσιμο **arm-cortexa9_neon-linux-gnueabihf-gcc** χρησιμοποιεί την :

libc.so.6: Η βιβλιοθήκη C που χρησιμοποιείται είναι η glibc, η οποία είναι η πιο κοινή και πλήρης βιβλιοθήκη για Linux συστήματα. Η glibc παρέχει όλες τις βασικές λειτουργίες που απαιτούνται από τις εφαρμογές, όπως:

- Διαχείριση μνήμης
- Είσοδος/έξοδος
- Διαχείριση διαδικασιών και νήματα
- Κλήσεις συστήματος και άλλες βασικές λειτουργίες

Επιπλέον η glibc είναι συμβατή με την αρχιτεκτονική arm-cortexa9_neon-linux-gnueabihf.

4)
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

```
dora@DESKTOP-J824248:~/crosstool/bin$
~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin/arm-cortexa9_neon-linux-gnueabihf-gc
c -O0 -Wall -o phods_crosstool.out /mnt/c/Users/basok/Desktop/embedded/Lab6/phods.c
```

```
dora@DESKTOP-J824248:~/crosstool/bin$ 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, with debug_info, not stripped
```

Η εντολή file phods_crosstool.out παρέχει τις εξής πληροφορίες για το εκτελέσιμο:

- Αρχείο τύπου: ELF (Executable and Linkable Format) 32-bit LSB executable
- Η αρχιτεκτονική του αρχείου είναι: ARM
- EABI (Embedded Application Binary Interface) VERSION 1 (SYSV): Το EABI ορίζει το πρότυπο με το οποίο οι εφαρμογές αλληλεπιδρούν με το λειτουργικό σύστημα, τις βιβλιοθήκες και άλλες συστηματικές λειτουργίες. Ειδικότερα, τα ενσωματωμένα συστήματα που βασίζονται σε ARM χρησιμοποιούν το EABI για να διασφαλίσουν την ορθή λειτουργία τους με περιορισμένους πόρους, χωρίς την ανάγκη ενός πλήρους λειτουργικού συστήματος
- Dynamically linked:Το αρχείο είναι δυναμικά συνδεδεμένο (dynamically linked), που σημαίνει ότι συνδέει βιβλιοθήκες κατά την εκτέλεση, αντί να τις συμπεριλαμβάνει στο εκτελέσιμο.
- interpreter /lib/ld-linux-armhf.so.3 : Αυτή είναι η βιβλιοθήκη φόρτωσης που χρησιμοποιεί το εκτελέσιμο.

- Το εκτελέσιμο είναι για πυρήνα GNU/Linux 3.2.0.Περιέχει πληροφορίες debug info και not stripped

Η εντολή **readelf** παρέχει χρήσιμες πληροφορίες για αρχεία σε μορφή ELF (Executable and Linkable Format), δηλαδή:

```
dora@DESKTOP-J824248:~/crosstool/bin$ 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:               0x104e0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          13644 (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:         36
  Section header string table index: 35
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
```

5)

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;

```
panosbel@lenovo:~/Lab6$ arm-linux-gnueabihf-gcc -O0 -Wall -std=c99 -o phods_linaro.out phods.c
```

```
error while loading shared libraries: libstdc++.so.6: cannot open shared object file: No such file or directory
```

Το linaro έχει περιορισμένη υποστήριξη για βιβλιοθήκες 64bit, οπότε κατεβάζουμε τα παρακάτω για να μπορεί να τρέξει

```
sudo apt-get install zlib1g
sudo apt-get install zlib1g:i386
```

```
 error while loading shared libraries: libz.so.1
```

```
sudo apt-get install libz1g-armhf-dev
```

```
panosbel@lenovo:~/Lab6$ ls
builddeb_hf.patch  kernel_konfig  phods_crosstool.out  sources.list
Exercise6.pdf      phods.c        phods_linaro.out
panosbel@lenovo:~/Lab6$ ls -l phods_crosstool.out
-rwxrwxr-x 1 panosbel panosbel 15092 Jan  9 20:44 phods_crosstool.out
panosbel@lenovo:~/Lab6$ ls -l phods_linaro.out
-rwxrwxr-x 1 panosbel panosbel 8882 Jan  9 22:29 phods_linaro.out
panosbel@lenovo:~/Lab6$
```

Ναι, παρατηρούμε διαφορά στα μεγέθη των εκτελέσιμων, συγκεκριμένα το μέγεθος του εκτελέσιμου από το linaro είναι σχεδόν το μισό σε σχέση με το εκτελέσιμο απο το crosstool-ng.

```
panosbel@lenovo:~/Lab6$ file phods_linaro.out
phods_linaro.out: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked,
interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.1.1, BuildID[sha1]=8e97732276b4fbcda0513
f18040715d9f3052223, not stripped
panosbel@lenovo:~/Lab6$ file phods_crosstool.out
phods_crosstool.out: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linke
d, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0, with debug_info, not stripped
panosbel@lenovo:~/Lab6$
```

Αρχικά κατα την εκτέλεση της παραπάνω εντολής παρατηρούμε ότι το phods_linaro.out δεν περιέχει debug info, με αποτέλεσμα να είναι μικρότερο.

Επιπλέον, και το εκτελέσιμο του Linaro και του Crosstool-NG προορίζονται για αρχιτεκτονική ARM 32-bit, παρόλα αυτά το Crosstool-NG έγινε build με την 64-bit έκδοση του glibc, ενώ για το Linaro χρησιμοποιήσαμε την 32-bit έκδοση λόγω της περιορισμένης υποστήριξης του για βιβλιοθήκες 64bit. Έτσι εξηγείται το πολύ μικρότερο μέγεθος,

6)

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

Για το crosstool-ng η σωστή εκτέλεση του προγράμματος στο target μηχάνημα, παρά τη χρήση διαφορετικής βιβλιοθήκης της C, εξηγείται λογω της δυναμικής σύνδεσης με την οποία το πρόγραμμα συνδέεται με την βιβλιοθήκη glibc κατα την εκτέλεση. Αρα ανεξαρτήτως αν το σύστημα είναι 32 bit ή 64 bit με την χρήση της δυναμικής σύνδεσης μπορούμε να χρησιμοποιήσουμε την κατάλληλη βιβλιοθήκη glibc, όπως και στην περίπτωση μας το σύστημα είναι ELF 32 bit και η βιβλιοθήκη glibc είναι 64 bit στο Host για το crosstool-ng.

7)

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;

```
panosbel@lenovo:~/Lab6$ arm-linux-gnueabihf-gcc -O0 -Wall -std=c99 -static -o phods_linaro.out phods.c
```

```
panosbel@lenovo:~/Lab6$ arm-cortexa9_neon-linux-gnueabihf-gcc -O0 -Wall -static -o phods_crosstool.out phods.c
```

αφού τα κάναμε static

```
panosbel@lenovo:~/Lab6$ file phods_crosstool.out
phods_crosstool.out: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked, for GNU/Linux 3.2.0, with debug_info, not stripped
panosbel@lenovo:~/Lab6$ file phods_linaro.out
phods_linaro.out: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), statically linked, for GNU/Linux 3.1.1, BuildID[sha1]=6207a1628043409bb6807c9db24fe7fd35535414, not stripped
```

τα μεγέθη των αρχείων είναι :

```
panosbel@lenovo:~/Lab6$ ls -l phods_crosstool.out
-rwxrwxr-x 1 panosbel panosbel 2561832 Jan 10 01:01 phods_crosstool.out
panosbel@lenovo:~/Lab6$ ls -l phods_linaro.out
-rwxrwxr-x 1 panosbel panosbel 508771 Jan 10 00:59 phods_linaro.out
panosbel@lenovo:~/Lab6$
```

Τα μεγέθη των executables είναι πολύ μεγαλύτερα με static linking σε σχέση με dynamic linking. Με το static linking γίνονται copy οι απαραίτητες βιβλιοθήκες της libc στο executable κατα το build. Με το dynamic linking αποθηκεύονται στο executables μόνο pointers που δείχνουν στην εξωτερική βιβλιοθήκη. Η βιβλιοθήκη φορτώνεται στη μνήμη μαζί με το executable κατά την εκτέλεση δυναμικά.

8)

Προσθέτουμε μία δική μας συνάρτηση στη 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

Α)Τι θα συμβεί αν εκτελέσουμε το my_foo.out στο host μηχάνημα;

> Αν προσπαθήσουμε να εκτελέσουμε το my_foo.out στο host μηχάνημα, το εκτελέσιμο δεν θα μπορέσει να τρέξει, καθώς είναι προορισμένο για αρχιτεκτονική ARM, ενώ το host μηχάνημα χρησιμοποιεί αρχιτεκτονική x86. Οι διαφορετικές αρχιτεκτονικές δεν είναι συμβατές και το λειτουργικό σύστημα του host δεν θα μπορεί να αναγνωρίσει το εκτελέσιμο για να το εκτελέσει.

Β) Τι θα συμβεί αν εκτελέσουμε το my_foo.out στο target μηχάνημα;

> Αν το target μηχάνημα διαθέτει την ανανεωμένη glibc που απαιτεί το εκτελέσιμο, τότε το πρόγραμμα θα εκτελεστεί κανονικά. Αν όμως η glibc του target μηχανήματος είναι διαφορετική ή παλιότερη από αυτή που χρησιμοποιήθηκε στο cross-compilation, το πρόγραμμα δεν θα μπορεί να τρέξει λόγω ασυμβατότητας.

Γ) 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 μηχάνημα;

Με το flag -static το εκτελέσιμο περιλαμβάνει την ανανεωμένη glibc και θα τρέξει ανεξάρτητα με την glibc του target.Αυτό σημαίνει ότι το εκτελέσιμο δεν εξαρτάται από τη glibc του target μηχανήματος και μπορεί να εκτελείται ανεξάρτητα από την έκδοση της glibc που υπάρχει στο target.

Ασκηση 2

Στα πλαίσια αυτής της άσκησης θα χτίσουμε έναν νέο πυρήνα για το Debian OS που τρέχαμε στις ασκήσεις 1 και 2. Τα απαραίτητα συστατικά για την επίτευξη αυτού του στόχου είναι:

1) Για την υλοποίηση επιλέξαμε να χρησιμοποιήσουμε το Crosstool-NG ως το εργαλείο για τη δημιουργία του cross compiler.
2) Ελέγχουμε το directory /boot/ στο Debian σύστημα του QEMU για να διαπιστώσουμε ποια αρχεία του Linux image και του initrd υπάρχουν, ώστε να τα ξεχωρίσουμε και να τα συγκρίνουμε με αυτά που θα δημιουργηθούν μετά την εγκατάσταση του νέου πυρήνα.

```
root@debian-armhf:/# cd boot/
root@debian-armhf:/boot# ls
config-3.2.0-4-vexpress      lost+found                  vmlinuz-3.2.0-4-vexpress
initrd.img                   System.map-3.2.0-4-vexpress
initrd.img-3.2.0-4-vexpress  vmlinuz
root@debian-armhf:/boot#
```

Ακολουθούμε τις οδηγίες για τα βήματα 3 και 4.

5)Στο βήμα 5, κάναμε τα εξής:

```
:~/path/to/kernel$ cp /path/to/kernel_config .config
```

και έπειτα ξεκινήσαμε η διαδικασία κατασκευής του πυρήνα με χρήση cross-compiler για την αρχιτεκτονική ARM:

```
:~/path/to/kernel$ make ARCH=arm
CROSS_COMPILE=<path_to_your_cross_compiler>/bin/arm-cortexa9_neon-linux-gnueabihf-
```

Κατά την εκτέλεση της δεύτερης εντολής, αντιμετωπίσαμε το ακόλουθο σφάλμα:

```
  HOSTCC  scripts/dtc/livetree.o
  HOSTCC  scripts/dtc/srcpos.o
  HOSTCC  scripts/dtc/treesource.o
  HOSTCC  scripts/dtc/util.o
  HOSTLD  scripts/dtc/dtc
/usr/bin/ld: scripts/dtc/dtc-parser.tab.o:(.bss+0x50): multiple definition of `yylloc'; scripts/dtc/dtc-lexer.lex.o:(.bss+0x0): first defined here
collect2: error: ld returned 1 exit status
make[2]: *** [scripts/Makefile.host:127: scripts/dtc/dtc] Error 1
make[1]: *** [scripts/Makefile.build:404: scripts/dtc] Error 2
make: *** [Makefile:563: scripts] Error 2
dora@DESKTOP-J824248:~/linux-source-3.16$
```

/path/to/linuxsource/scripts/dtc/dtc.lexer.lex.c_shipped

Για την επίλυση του προβλήματος, όπως αναφέρθηκε στις οδηγίες κάναμε την μεταβλητή YYLTYPE yylloc extern.

Στη συνέχεια, προέκυψε αυτό το σφάλμα.

```
arch/arm/mm/proc-v7.S: Assembler messages: arch/arm/mm/proc-v7.S:429: Error:
junk at end of line, first unrecognized character is `#' make[1]: ***
[scripts/Makefile.build:293: arch/arm/mm/proc-v7.o] Error 1 make: ***
[Makefile:918: arch/arm/mm] Error 2
```

Για να το λύσουμε, εφαρμόσαμε αυτό το patch :
https://gitlab.com/postmarketOS/pmaports/-/blob/aa289aa350071e6afc54f6b6704ba28971b50466/device/.shared-patches/linux/linux3.4-ARM-8933-1-replace-Sun-Solaris-style-flag-on-section.patch

Και το τοποθετούμε μέσα στον Linux kernel μας, επειτα το εκτελούμε με την εντολή:

```
patch --verbose -p1 <
~/Lab6/linux-source-3.16/patches/linux3.4-ARM-8933-1-replace-Sun-Solaris-style-f
lag-on-section.patch
```

6) Με το πέρας της εντολής, δημιουργήθηκαν τρία deb αρχεία στο parent directory του Linux source. Με την εντολή dpkg –info file_name.deb πήραμε περισσότερες πληροφορίες για το κάθε ένα από αυτά τα αρχεία.

dpkg --info ../linux-libc-dev_3.16.84-4_armhf.deb

```
panosbel@lenovo:~/Lab6$ dpkg --info linux-libc-dev_3.16.84-3_armhf.deb
 new Debian package, version 2.0.
 size 770840 bytes: control archive=18512 bytes.
     459 bytes,    13 lines      control
   50703 bytes,   773 lines      md5sums
 Package: linux-libc-dev
 Source: linux-upstream
 Version: 3.16.84-3
 Architecture: armhf
 Maintainer: Anonymous <root@lenovo>
 Installed-Size: 3743
 Provides: linux-kernel-headers
 Section: devel
 Priority: optional
 Homepage: http://www.kernel.org/
 Description: Linux support headers for userspace development
  This package provides userspaces headers from the Linux kernel.  These headers
  are used by the installed headers for GNU glibc and other system libraries.
panosbel@lenovo:~/Lab6$
```

dpkg --info ../linux-image-3.16.8 4_3.16.84-4_armhf.deb

```
panosbel@lenovo:~/Lab6$ dpkg --info linux-image-3.16.84_3.16.84-3_armhf.deb
 new Debian package, version 2.0.
 size 12454748 bytes: control archive=28867 bytes.
     460 bytes,    14 lines      control
   98433 bytes,  1121 lines      md5sums
     285 bytes,    12 lines   *  postinst             #!/bin/sh
     281 bytes,    12 lines   *  postrm               #!/bin/sh
     283 bytes,    12 lines   *  preinst              #!/bin/sh
     279 bytes,    12 lines   *  prerm                #!/bin/sh
 Package: linux-image-3.16.84
 Source: linux-upstream
 Version: 3.16.84-3
 Architecture: armhf
 Maintainer: Anonymous <root@lenovo>
 Installed-Size: 40357
 Suggests: linux-firmware-image-3.16.84
 Provides: linux-image, linux-image-2.6, linux-modules-3.16.84
 Section: kernel
 Priority: optional
 Homepage: http://www.kernel.org/
 Description: Linux kernel, version 3.16.84
  This package contains the Linux kernel, modules and corresponding other
  files, version: 3.16.84.
```

dpkg --info ../linux-headers-3.16.84_3.16.84-4_armhf.deb

```
panosbel@lenovo:~/Lab6$ dpkg --info linux-headers-3.16.84_3.16.84-3_armhf.deb
 new Debian package, version 2.0.
 size 6723148 bytes: control archive=199573 bytes.
     446 bytes,    14 lines      control
  953868 bytes, 10073 lines      md5sums
 Package: linux-headers-3.16.84
 Source: linux-upstream
 Version: 3.16.84-3
 Architecture: armhf
 Maintainer: Anonymous <root@lenovo>
 Installed-Size: 42153
 Provides: linux-headers, linux-headers-2.6
 Section: kernel
 Priority: optional
 Homepage: http://www.kernel.org/
 Description: Linux kernel headers for 3.16.84 on armhf
  This package provides kernel header files for 3.16.84 on armhf
  .
  This is useful for people who need to build external modules
```

7)

Μέσω της χρήσης της εντολής scp, μεταφέρουμε τα αρχεία .deb στο target μηχάνημα

```
scp          -P          22223          linux-image-3.16.84_3.16.84-3_armhf.deb
linux-headers-3.16.84_3.16.84-3_armhf.deb     linux-libc-dev_3.16.84-3_armhf.deb
root@localhost:/root/
```

Όταν προσπαθήσαμε με την εντολή dpkg -i να εγκαταστήσουμε τα πακέτα αντιμετωπίσαμε αυτό το error:

```
root@debian-armhf:~# ls
linux-headers-3.16.84_3.16.84-3_armhf.deb  linux-libc-dev_3.16.84-3_armhf.deb
linux-image-3.16.84_3.16.84-3_armhf.deb
root@debian-armhf:~# dpkg -i linux-libc-dev_3.16.84-3_armhf.deb
dpkg-deb: error: archive 'linux-libc-dev_3.16.84-3_armhf.deb' uses unknown compression for
member 'control.tar.zst', giving up
dpkg: error processing archive linux-libc-dev_3.16.84-3_armhf.deb (--install):
 subprocess dpkg-deb --control returned error exit status 2
Errors were encountered while processing:
 linux-libc-dev_3.16.84-3_armhf.deb
```

Για να προχωρήσουμε, ανοίξαμε το πακέτο .deb χειροκίνητα με τη χρήση της εντολής:

```
ar x linux-libc-dev_3.16.84-3_armhf.deb
```

Παρατηρήσαμε ότι δεν μπορούσε να γίνει η αποσυμπίεση 2 αρχείων (control.tar.zst, data.tar.zst) επειδή το dpkg (ακόμα και μετά από update) δεν υποστήριζε αυτό τον τύπο συμπίεσης.

Δοκιμάζουμε να εγκαταστήσαμε το zstd ώστε να κάνουμε αποσυμπίεση manually, με την εντολή:

```
apt-get install zstd
```

Η εγκατάσταση του zstd απέτυχε, οπότε ανανεώνουμε το sources.list ώς εξής:

```
sudo nano /etc/apt/sources.list
```

με αυτό :

```
deb http://deb.debian.org/debian buster main contrib non-free
deb-src http://deb.debian.org/debian buster main contrib non-free

deb  http://security.debian.org/debian-security  buster/updates  main  contrib
non-free
deb-src  http://security.debian.org/debian-security  buster/updates  main  contrib
non-free
```

Έπειτα :

```
apt-get update
apt-get install zstd
```

Τελικά, αποσυμπιέσαμε τα αρχεία control.tar.zst και data.tar.zst.

```
unzstd control.tar.zst
unzstd data.tar.zst
```

Διαγράφουμε το παλιό αρχείο linux-libc-dev_3.16.84-3_armhf.deb.

Φτιάχνουμε το καινούργιο .deb file με την εντολή:

```
ar r linux-libc-dev_3.16.84-3_armhf.deb control.tar data.tar debian-binary
```

Πλέον το .deb file είναι έτοιμο και το τρέχουμε με την εντολή `dpkg -i.` Επαναλαμβάνουμε την ίδια διαδικασια και για τα 3 .deb files διαγράφοντας τα παλια αρχεία zst.

8)

Η εγκατάσταση του πακέτου linux-image εγκατέστησε στον κατάλογο /boot/ του target συστήματος ένα image του πυρήνα και ένα ανανεωμένο initrd.

```
root@debian-armhf:/usr/src# scp  /boot/vmlinuz-3.16.84 /boot/initrd.img-3.16
.84 dora@172.25.20.192:/home/dora/
dora@172.25.20.192's password:
vmlinuz-3.16.84
                                              100% 2388KB   1.2MB/s   00:02
initrd.img-3.16.84
                                              100% 2804KB   2.7MB/s   00:01
root@debian-armhf:/usr/src#
```

Για να λειτουργήσει ορθά το σύστημά μας με τον νέο πυρήνα, πρέπει να τα δίνουμε σαν ορίσματα στον QEMU όταν το τρέχουμε. Επομένως, κατεβάσαμε τα αρχεία αυτά στο host μηχάνημα

```
dora@DESKTOP-J824248:~$ ls -l ~/vmlinuz-3.16.84 ~/initrd.img-3.16.84
-rw-r--r-- 1 dora dora 2871260 Jan 11 15:50 /home/dora/initrd.img-3.16.84
-rwxr-xr-x 1 dora dora 2445032 Jan 11 15:50 /home/dora/vmlinuz-3.16.84
```

Επανεκκινήσαμε τον QEMU δίνοντας τα ως ορίσματα στις παραμέτρους kernel και initrd.

```
sudo qemu-system-arm \
       -M vexpress-a9 \
       -kernel ~/vmlinuz-3.16.84 \
       -initrd ~/initrd.img-3.16.84 \
       -drive
if=sd,file=/home/panosbel/arm_v7_vm/debian_wheezy_armhf_standard.qcow2 \
       -append "root=/dev/mmcblk0p2 rw" \
       -net nic \
       -net user,hostfwd=tcp:127.0.0.1:22223-:22
```

Ερωτήματα:

1)
Εκτελέσαμε `:~$ uname -a` στο Qemu, πριν και αφότου εγκαταστήσουμε τον νέο πυρήνα

Πριν την εγκατάσταση του νέου πυρήνα::

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

Linux debian-armhf 3.2.0-4-vexpress #1 SMP Debian 3.2.51-1 armv7l GNU/Linux

Μετά την εγκατάσταση του νέου πυρήνα::

```
root@debian-armhf:~#  uname -a
Linux debian-armhf 3.16.84 #4 SMP Sat Jan 11 00:58:22 EET 2025 armv7l GNU/Linux
root@debian-armhf:~#
```

Linux debian-armhf 3.16.84 #4 SMP Sat Jan 11 00:58:22 EET 2025 armv7l GNU/Linux

Ο πυρήνας του συστήματός μας έχει αναβαθμιστεί επιτυχώς από την έκδοση 3.2.0-4-vexpress στην έκδοση 3.16.84 που είναι πολύ πιο πρόσφατη. Ο νέος πυρήνας δεν γράφει Debian διότι είναι custom και φτιάχτηκε από εμάς. Αυτή η αναβάθμιση είναι αποτέλεσμα των προσπαθειών μας να δημιουργήσουμε έναν νέο πυρήνα, όπως αναφέρεται στα προηγούμενα βήματα της διαδικασίας.

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. Τι αλλαγές κάνατε στον πηγαίο κώδικα του πυρήνα;

Αρχικά πάμε από τον host στο linux-source-3.16/kernel
και φτιάχνουμε ένα νέο directory hello, μέσα στο οποίο θα βάλουμε ένα hello.c

hello.c

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

 asmlinkage long sys_hello(void)
{
        printk("Greeting from kernel and team no %d\n",22);
        return 0;
 }
```

Στη συνέχεια, φτιάχνουμε ένα Makefile που περιέχει αυτή τη γραμμή:
obj-y := hello.o

Το οποίο λέει στο kernel build system να συμπεριλάβει το hello.o στη λίστα με τα objects που θα γίνουν linked.

Βάζουμε το hello directory στο kernel’s Makefile

βάζουμε στο Makefile του kernel να γίνει compile και το directory hello

```
CFLAGS_REMOVE_irq_work.o = -pg
endif

# cond_syscall is currently not LTO compatible
CFLAGS_sys_ni.o = $(DISABLE_LTO)

obj-y += sched/
obj-y += locking/
obj-y += power/
obj-y += printk/
obj-y += irq/
obj-y += rcu/
obj-y += hello/
```

Αυτή η συγκεκριμένη γραμμή που αλλάζουμε λέει στον kernel όταν κάνει build να συμπεριλάβει τα ακόλουθα directories (μετά το +=) τα οποία πρέπει να γίνουν compile. Έτσι προσθέτουμε το directory hello.

Βάζουμε το νέο system call (sys_hello()) στο calls.S file

Το επόμενο είναι να φτιάξουμε ένα νέο system-call
Πηγαίνουμε στο directory linux-source-3.16/arch/arm/kernel και ανοίγουμε πρώτα το αρχείο calls.S

```
/* 380 */       CALL(sys_sched_setattr)
                CALL(sys_sched_getattr)
                CALL(sys_renameat2)
                CALL(sys_ni_syscall)            /* seccomp */
                CALL(sys_ni_syscall)            /* getrandom */
/* 385 */       CALL(sys_memfd_create)
/* 386 */       CALL(sys_hello)
#ifndef syscalls_counted
.equ syscalls_padding, ((NR_syscalls + 3) & ~3) - NR_syscalls
#define syscalls_counted
#endif
.rept syscalls_padding
                CALL(sys_ni_syscall)
.endr
```

προσθέτουμε κάτω κάτω το syscall hello
Βάζουμε το νέο system call(sys_hello()) στο system call header file(syscalls.h).

Πηγαίνουμε στο directory linux-source-3.16/include/linux
και ανοίγουμε το αρχείο syscalls.h

```
asmlinkage long sys_kcmp(pid_t pid1, pid_t pid2, int type,
                         unsigned long idx1, unsigned long idx2);
asmlinkage long sys_finit_module(int fd, const char __user *uargs, int flags);
asmlinkage long sys_seccomp(unsigned int op, unsigned int flags,
                            const char __user *uargs);
asmlinkage long sys_hello(void);
#endif
```

Προσθέτουμε κάτω κάτω : *asmlinkage long sys_hello(void);*

Η λέξη-κλειδί asmlinkage χρησιμοποιείται στον πυρήνα Linux για να καθορίσει ότι τα επιχειρήματα της συνάρτησης πρέπει να μεταβιβαστούν μέσω της στοίβας (stack), και όχι μέσω των καταχωρητών.

Πηγαίνουμε στο linux-source-3.16/arch/arm/include/uapi/asm directory και στο αρχείο unistd.h

```
#define __NR_syncfs                     (__NR_SYSCALL_BASE+373)
#define __NR_sendmmsg                   (__NR_SYSCALL_BASE+374)
#define __NR_setns                      (__NR_SYSCALL_BASE+375)
#define __NR_process_vm_readv           (__NR_SYSCALL_BASE+376)
#define __NR_process_vm_writev          (__NR_SYSCALL_BASE+377)
#define __NR_kcmp                       (__NR_SYSCALL_BASE+378)
#define __NR_finit_module               (__NR_SYSCALL_BASE+379)
#define __NR_sched_setattr              (__NR_SYSCALL_BASE+380)
#define __NR_sched_getattr              (__NR_SYSCALL_BASE+381)
#define __NR_renameat2                  (__NR_SYSCALL_BASE+382)
#define __NR_memfd_create               (__NR_SYSCALL_BASE+385)
#define __NR_hello                      (__NR_SYSCALL_BASE+386)
/*
 * The following SWIs are ARM private.
 */
```

Βάζουμε σαν τελευταίο define την παρακάτω γραμμή

#define __NR_hello (__NR_SYSCALL_BASE+386)

Έτσι προσθέτουμε τον αριθμό του system call μας.

Όπως προηγουμένως, ξαναπαράγουμε τα 3 .deb files

```
make ARCH=arm
CROSS_COMPILE=~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin/arm-cortexa9_neon-
linux-gnueabihf- -j 4 deb-pkg
```

και τα περνάμε στο target:

```
scp -P 22223 linux-image-3.16.84_3.16.84-4_armhf.deb
linux-headers-3.16.84_3.16.84-4_armhf.deb
linux-libc-dev_3.16.84-4_armhf.deb root@localhost:/root/new/
```

Σβήνουμε από το /boot τα παλιά initrd.img-3.16.84 vmlinuz-3.16.84
Τρέχουμε τα .deb files στο target και προκύπτουν τα νέα initrd.img-3.16.84 vmlinuz-3.16.8, τα οποία τα περνάμε με scp στον host.

```
panosbel@lenovo:~/Lab6/new$ ls
initrd.img-3.16.84  vmlinuz-3.16.84
panosbel@lenovo:~/Lab6/new$ sudo qemu-system-arm -M vexpress-a9 -kernel ~/Lab6/new/vmlinuz-
3.16.84 -initrd ~/Lab6/new/initrd.img-3.16.84 -drive if=sd,file=debian_wheezy_armhf_standar
d.qcow2 -append "root=/dev/mmcblk0p2 -net nic -net user,hostfwd=tcp:127.0.0.1:22223-:22"
```

Με τον νέο πυρήνα τρέχουμε πάλι το target device.

```
sudo qemu-system-arm \
       -M vexpress-a9 \
       -kernel ~/Lab6/new/vmlinuz-3.16.84 \
       -initrd ~/Lab6/new/initrd.img-3.16.84 \
       -drive
if=sd,file=/home/panosbel/arm_v7_vm/debian_wheezy_armhf_standard.qcow2 \
       -append "root=/dev/mmcblk0p2 rw" \
       -net nic \
       -net user,hostfwd=tcp:127.0.0.1:22223-:22
```

```
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
root@debian-armhf:~# uname -a
Linux debian-armhf 3.16.84 #4 SMP Sun Jan 12 18:53:57 EET 2025 armv7l GNU/Linux
root@debian-armhf:~# _
```

Παρατηρούμε ότι έχει αλλάξει και η ημερομηνία οπότε έχουμε βάλει τον σωστό πυρήνα.

3)

Γράψτε ένα πρόγραμμα σε γλωσσα C το οποίο θα κάνει χρήση του system call που προσθέσατε.

Για να ελέγξουμε την ορθότητα του system call που φτιάξαμε, κάνουμε το παρακάτω πρόγραμμα (syscall.c)

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

#define SYS_hello 386

int main(){
long result = syscall(SYS_hello);

if(result==0){
        printf("sys_hello executed successfully.\n");
}
else{
        perror("sys_hello failed.\n");
}
return 0;
}
```

```
root@debian-armhf:/home/user# gcc -o syscall syscall.c
root@debian-armhf:/home/user# ls
syscall  syscall.c  test.c
root@debian-armhf:/home/user# ./syscall
[  921.878754] Greeting from kernel and team no 22
sys_hello executed successfully.
root@debian-armhf:/home/user#
```

Το system call δουλεύει !