ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Σχεδιασμός Ενσωματωμένων Συστημάτων
6η Σειρά Ασκήσεων

Χειμερινό Εξάμηνο, Ακαδ. Έτος: 2023-2024

Ονοματεπώνυμο: Διαμαντίδης Θεοχάρης **ΑΜ:** 03119002
Ονοματεπώνυμο: Ιωάννου Κωνσταντίνος **ΑΜ:** 03119840

Εξάμηνο: 9
Ομάδα:9

Cross-compiling προγραμμάτων για ARM αρχιτεκτονική

Το πρώτο βήμα που θα ακολοηθήσαμε είναι η εγκατάσταση τον cross compilers crosstool-ng και linaro.

custom cross-compiler building toolchain crosstool-ng:
(https://crosstool-ng.github.io/docs/toolchain-construction/)

pre-compiled building toolchain linaro:
(https://www.linaro.org/downloads/)

Η βασική διαφορά μεταξύ του Crosstool-NG και του Linaro είναι ότι το Crosstool-NG είναι ένα εργαλείο για τη δημιουργία cross-compilers, δηλαδή των εργαλείων που χρησιμοποιούνται για την ανάπτυξη λογισμικού για διαφορετικούς στόχους, ενώ το Linaro είναι μια οργάνωση που παρέχει λογισμικό και εργαλεία ανάπτυξης, συμπεριλαμβανομένων και compilers, για συγκεκριμένες αρχιτεκτονικές όπως η ARM. Συνεπώς, το Crosstool-NG μπορεί να χρησιμοποιηθεί για τη δημιουργία compilers για οποιαδήποτε αρχιτεκτονική, ενώ το Linaro εστιάζει κυρίως στην υποστήριξη των ARM-based συστημάτων.

Άσκηση 1

1) Ποια η διαφορά μεταξύ του καινούριου image που κατεβάσαμε ?

armhf (Hard Float): A new Arm port requiring the presence of a FPU would help squeeze the most performance juice out of hardware with a FPU, υποστηρίζει υψηλότερη απόδοση επεξεργασίας, καθώς χρησιμοποιεί το σύνολο εντολών που εκμεταλλεύεται την υποστήριξη "hard float" των ARM επεξεργαστών.

arme (Soft Float):Παλαιότεροι επεξεργαστές 32 bit ,χρησιμοποιεί την "soft float" και δεν εκμεταλλεύεται την υποστήριξη "hard float" των επεξεργαστών ARM.Μπορεί να είναι λιγότερο αποδοτική σε ορισμένες εργασίες που απαιτούν πολλαπλασιασμό κινητής υποδιαστολής και άλλες υπολογιστικές λειτουργίες.

2) Γιατί χρησιμοποιήσαμε την αρχιτεκτονική arm-cortexa9_neon-linux-gnueabihf;

Η αρχιτεκτονική `arm-cortexa9_neon-linux-gnueabihf` αναφέρεται στον επεξεργαστή ARM Cortex-A9 που χρησιμοποιεί το σύνολο εντολών NEON και υποστηρίζει την "hard float" αριθμητική. Όταν τρέχουμε ένα cross-compiled εκτελέσιμο στον QEMU, η επιλογή της αρχιτεκτονικής είναι κρίσιμη για τη σωστή λειτουργία του εκτελέσιμου αρχείου. Εάν χρησιμοποιήσουμε μια διαφορετική αρχιτεκτονική από αυτήν, το εκτελέσιμο αρχείο μπορεί να μην είναι συμβατό κώδικα με την πραγματική αρχιτεκτονική του QEMU.Συνολικά, η επιλογή της σωστής αρχιτεκτονικής είναι σημαντική για την επιτυχή εκτέλεση ενός cross-compiled εκτελέσιμου στον QEMU και για να διασφαλιστεί η συμβατότητα με το περιβάλλον εκτέλεσης.

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

Στο path ~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin
τρέχουμε την εντολή ldd arm-cortexa9_neon-linux-gnueabihf-gcc
και έχουμε ως αποτέλεσμα :

linux-vdso.so.1 (0x00007ffe703d3000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f9de2e00000)
/lib64/ld-linux-x86-64.so.2 (0x00007f9de3065000)

Η έξοδος της εντολής `ldd arm-cortexa9_neon-linux-gnueabihf-gcc` που εκτελέσατε δείχνει τις βιβλιοθήκες που είναι εξαρτημένες από το εκτελέσιμο αρχείο `arm-cortexa9_neon-linux-gnueabihf-gcc. Η `libc.so.6` είναι η βιβλιοθήκη C (libc) που είναι απαραίτητη για την εκτέλεση του εκτελέσιμου αρχείου.

4) Χρησιμοποιώντας τον cross compiler που παρήχθει από τον crosstool-ng κάντε compile τονκώδικα phods.c

ΒΗΜΑ 1
:~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin$
sudo ./arm-cortexa9_neon-linux-gnueabihf-gcc /PATH/phods.c -O0 -Wall -o phods_crosstool.out

Χρησιμοποιώντας τον cross-compiler κάνουμε compile τον κώδικα phods.c και απενεργοποιώντας τις αυτόματες βελτιστοποιήσεις απο τον compiler δημιουργούμε το εκτελέσιμο αρχείο με όνομα phods_crosstool.out

ΒΗΜΑ 2
:~/x-tools/arm-cortexa9_neon-linux-gnueabihf/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

- Τύπος αρχείου: ELF 32-bit LSB executable
- Αρχιτεκτονική: ARM
- Έκδοση EABI: EABI5 version 1 (SYSV)
- Συνδεδεμένο δυναμικά: Ναι

ΒΗΜΑ 3

:~/x-tools/arm-cortexa9_neon-linux-gnueabihf/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:               0x10478
  Start of program headers:          52 (bytes into file)
  Start of section headers:          14960 (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
```

Η εντολή `readelf -h -A phods_crosstool.out` χρησιμοποιείται για να διαβάσει την επικεφαλίδα (header) του αρχείου ELF και να εμφανίσει πληροφορίες για το αρχείο.

- Η Magic number (`7f 45 4c 46`) αναγνωρίζει το αρχείο ως ένα αρχείο ELF.
- Η κλάση είναι ELF32, δηλαδή 32-bit.
- Τα δεδομένα είναι 2's complement, little endian.
- Η έκδοση είναι 1.
- Το αρχείο είναι τύπου EXEC, δηλαδή εκτελέσιμο.
- Η αρχιτεκτονική είναι ARM.
- Το entry point address είναι `0x10478`.
- Έχουμε 9 προγράμματα επικεφαλίδες.
- Έχουμε 36 επικεφαλίδες ενοτήτων.

Η ενότητα `Attribute Section: aeabi` περιλαμβάνει αρκετές πληροφορίες σχετικά με την αρχιτεκτονική και τα χαρακτηριστικά του αρχείου. Αυτά περιλαμβάνουν το όνομα και την έκδοση του CPU, το προφίλ αρχιτεκτονικής CPU, την χρήση του ARM ISA, το Thumb-2 ISA, το VFPv3 FP αρχιτεκτονική, την NEONv1 SIMD αρχιτεκτονική και πολλά άλλα.

5) Χρησιμοποιώντας τον cross compiler που κατεβάσατε από το site της linaro κάντε compile τον ίδιο κώδικα.

~/linaro/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/bin/arm-linux-gnueabihf-gcc ~/phods.c -o ~/phods_linaro.out

error while loading shared libraries: libstdc++.so.6

sudo apt-get install lib32z1-dev

ξανατρέχουμε οπότε:

ls -l phods_linaro.out
-rwxrwxr-x 1 kostis kostis **8106** Φεβ 12 15:26 phods_linaro.out

~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin$ ls -l phods_crosstool.out
-rwxr-xr-x 1 root root **16400** Φεβ 12 13:08 phods_crosstool.out

Το γεγονός αυτό είναι αναμενόμενο καθώς όπως αναφέρθηκε προηγουμένως για την μεταγλώττιση χρειάστηκε η εγκατάσταση: sudo apt-get install lib32z1-dev που αποτελεί πακέτο για υποστήριξη 32-bit. Επομένως παρατηρείται διαφορά μεγέθους καθώς στην περίπτωση του crosstool χρησιμοποιούμε την 64-bit βιβλιοθήκη glib.

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

Η ερώτηση εστιάζει στο γεγονός ότι το πρόγραμμα φαίνεται να λειτουργεί σωστά στον στόχευστο υπολογιστή, παρόλο που χρησιμοποιεί διαφορετική βιβλιοθήκη C από αυτήν που είναι στον εκτελέσιμο κώδικα.
Το σημαντικό είναι ότι η βιβλιοθήκη C που χρησιμοποιείται κατά τη σύνδεση (linking) του εκτελέσιμου αρχείου πρέπει να είναι συμβατή και διαθέσιμη στο σύστημα στο οποίο θα εκτελεστεί το πρόγραμμα.

Για file phods_crosstool.out ->Τύπος αρχείου: ELF 32-bit LSB executable
Για phods_linaro.out: **ELF 32-bit LSB executab**le, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.1.1, BuildID[sha1]=8ec0a93bed3107cc1c58456f5015be8857fe78c2, not stripped

παρα το γεγονός ότι η βιβλιοθήκη του crossout είναι για 64 bit ενώ του linaro για 32 bit και στις δυο περιπτώσιες βλέπουμε ότι τα εκτελέσιμα έχουν ίδιο τύπο αρχείου οπότε το target μηχάνημα δεν θα εχει κάποιο πρόβλημα.

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.

1)sudo ~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin/arm-cortexa9_neon-linux-gnueabihf-gcc ~/phods.c -O0 -Wall -static -o ~/phods_**crosstool**_static.out

kostis@Kostis-laptop:~$ ls -l phods_crosstool_static.out
-rwxr-xr-x 1 root root 2891056 Φεβ 12 16:15 phods_crosstool_static.out

2) ~/linaro/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/bin/arm-linux-gnueabihf-gcc -static ~/phods.c -o ~/phods_linaro_static.out

kostis@Kostis-laptop:~$ ls -l phods_linaro_static.out
-rwxrwxr-x 1 kostis kostis 507853 Φεβ 12 16:19 phods_linaro_static.ou

Παρατηρούμε ότι σε κάθε περίπτωση ότι όταν έχουμε στατική την βιβλιοθήκη τότε το εκτελέσιμο καταλαμβάνει πολύ περισσότερο χώρο αυτό συμβαίνει γιατί αντικαθίστανται οι συναρτήσεις που χρησιμοποιούνται και γίνονται compile απευθείας στο εκτελέσιμο. Βέβαια πρέπει να σκεφτούμε ότι στην περίπτωση που έχουμε δυναμική χρειάζεται να αποθηκεύσουμε τις βιβλιοθήκες ολόκληρες στο ενσωματωμενο αλλά το ίδιο το εκτελέσιμο θα είναι πολύ μικρότερο. Προφανώς η linaro εξακολουθεί να είναι χρειάζεται λιγότερο χώρο απο την crosstool για υον λόγο που αναφέραμε προηγουμένως.

(SIZES)	linaro	crosstool
dunamic	**8106**	**16400**
static	**507853**	**2891056**

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.

Αρχικά δημιουργούμε μια νέα συνάρτηση mlab_foo() που επιτελεί μια συγκεκριμένη λειτουργία και την προσθέτουμε στην βιβλιοθήκη glibc και δημιουργούμε έναν cross-compiler με τον crosstool-ng που κάνει χρήση της ανανεωμένης glibc. (αυτό ίσως υπερβαίνει το παρόν μάθημα)

Έπειτα δημιουργούμε έναν κώδικα my_foo.c που χρησιμοποιεί την νεα συνάρτηση που προσθέσαμε στην βιβλιοθήκη και κάνουμε cross compile ώστε να φτιάξουμε το εκτελέσιμο .
arm-cortexa9_neon-linux-gnueabihf-gcc -Wall -O0 my_foo.c -o my_foo.out

Τώρα που έχουμε το εκτελέσιμο οπότε:

1) Περιμένουμε το πρόγραμμα να μην τρέξει Η διαδικασία cross-compilation παράγει εκτελέσιμα αρχεία που είναι σχεδιασμένα να εκτελούνται σε στόχους (target machines) με διαφορετική αρχιτεκτονική από αυτήν του συστήματος ανάπτυξης (host system). Ως εκ τούτου, είναι αλήθεια ότι τα εκτελέσιμα αρχεία που παράγονται από τη διαδικασία cross-compilation μπορεί να μην είναι συμβατά ή να μην εκτελούνται σωστά στο host μηχάνημα.

2)Αν τρέξουμε στο target μηχάνημα (στο qemu ή απλα φορτώσουμε σε εναν arm τον κώδικα μας) ,περιμένουμε να μην εκτελεστεί σωστά καθώς ενώ θα έχει μεταφραστεί σε binary γλώσσα το πρόγραμμα μας ,όταν έρθει η στιγμή να εκτελεστεί η νέα συνάρτηση τότε (δυναμικά) θα πάει στην βιβλιοθήκη που ήταν φορτωμένη στον μικροεπεξεργαστή να βρεί την συνάρτηση αλλα δεν θα την βρει γιατι στο target μηχάνημα δεν ενημερώσαμε ποτέ την βιβλιοθήκη.

3)Αν τώρα το τρέξουμε φορτώνοντας στατικά τις βιβλιοθήκες ,δηλαδή αντικαθιστώντας στο course code τις συναρτήσεις και δημιουργώντας ένα τελικό εκτελέσιμο (μεγαλύτερο προφανώς σε μέγεθος) ανεξάρτητο από τις βιβλιοθήκες τότε περιμένουμε να τρέξει κανονικά στο target μηχάνημα (eg arm,stm) καθώς δεν χρειάζεται να γίνει καμία κλήση στις βιβλιοθήκες .

ΑΣΚΗΣΗ 2

θα χτίσουμε έναν νέο πυρήνα για το Debian OS :

1)Τελικά αποφασίσαμε να χρησιμοποιήσουμε τον cross compiler του linaro ,καθώς αντιμετωπίσαμε error με τον crosstool-ngl το οποίο φάνηκε να οφείλεται στην έκδοση του gcc καθως δεν αναγνώριζε κάποια flags στην assembly κατά την διάρκεια του make.

2) Αρχικά το qemu έχει ως image τα αρχεία που κατεβάσαμε πρίν ,μέσα σε κατάλληλο φάκελο που φτιάξαμε:

vmlinuz-3.2.0-4-vexpress: Αυτό το αρχείο περιέχει τον πυρήνα (kernel) του Linux που χρησιμοποιείται από το σύστημα Debian στο QEMU. Ο πυρήνας είναι υπεύθυνος για την εκκίνηση του λειτουργικού συστήματος και τη διαχείριση των πόρων του υπολογιστή.

initrd.img-3.2.0-4-vexpress: Το αρχείο initrd (initial ramdisk) περιέχει ένα αρχείο συστήματος που φορτώνεται κατά την εκκίνηση του Linux για να παρέχει βασικές λειτουργίες και προσαρμογές πριν από το φορτώσει τον πραγματικό πυρήνα.

debian_wheezy_armhf_standard.qcow2` φαίνεται να είναι ένα αρχείο εικονικού δίσκου που περιέχει το λειτουργικό σύστημα Debian Wheezy για αρχιτεκτονική ARM.

Με όλα αυτά τα αρχεία μαζί, μπορείτε να δημιουργήσετε ένα εικονικό περιβάλλον Debian στο QEMU χρησιμοποιώντας τον πυρήνα Linux, το αρχείο initrd και το εικονικό δίσκο που περιέχει το σύστημα αρχείων Debian.

3)Αφού συνδεθούμε στο qemu με την αρχιτεκτονική:

sudo qemu-system-arm -M vexpress-a9 -kernel vmlinuz-3.2.0-4-vexpress -initrd initrd.img-3.2.0-4-vexpress -drive if=sd,file=debian_wheezy_armhf_standard.qcow2 -append "root=/dev/mmcblk0p2" -net nic -net user,hostfwd=tcp:127.0.0.1:22223-:22

Τότε τρέχουμε τις εξεις εντολές τις προηγούμενης άσκηση σχετικά με τα permision nano /etc/apt/sources.list και edit το source list και φτιάχνουμε σφάλματα σχετικά με τα κλειδιά των distributions και των packages που γίνονται update.

Τελος εκτελούμε τις εντολές
root@qemu:~$ apt-get update
root@qemu:~$ apt-get install linux-source

Η εκτέλεση αυτή θα κατεβάσει στο directory /usr/src το αρχείο linux-source-3.16.tar.xz

4)Στέλνουμε το παραπάνω αρχείο από το qemu στο host μηχάνημα για να γίνει πιο γρήγορα compile εκεί .

Με την εντολή:
scp -P 22223 root@localhost:/usr/src/linux-source-3.16.tar.xz ~/Desktop
κάνουμε copy το αρχείο linux-source-3.16.tar.xz στο host μηχάνημα.

αφού το κάνουμε extract κάνουμε cd και φτιάχνουμε κατάλληλο config για το target μηχάνημα, στην περίπτωση μας απλώς θα αντιγράψουμε το έτοιμο που μας δίνεται.

έπειτα τρέχουμε(διαρκεί κάποια ώρα)
make ARCH=arm CROSS_COMPILE=~//linaro/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/bin/arm-linux-gnueabihf-

error1:https://github.com/BPI-SINOVOIP/BPI-M4-bsp/issues/4\

5)Στην συνέχεια δημιουργούμε τα deb αρχεία

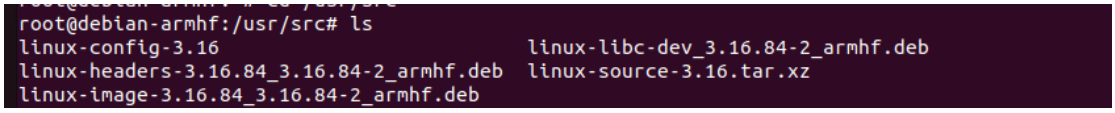
make ARCH=arm CROSS_COMPILE=/home/kostis/linaro/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/bin/arm-linux-gnueabihf-deb-pkg (και αυτό παίρνει κάποια ώρα ~αρκετή~)

και δημιουργούνται αυτά τα 3 αρχεία
linux-headers-3.16.84_3.16.84-2_armhf.deb linux-image-3.16.84_3.16.84-2_armhf.deb linux-libc-dev_3.16.84-2_armhf.deb

Προσοχή δοκιμάσαμε να στείλουμε απευθείας τα αρχεία στο qemu ,αλλά αυτό δεν δούλεψε καθώς η αρχιτεκτονική διαθέτει παλαιά έκδοση Debian και είχαμε error με την εντολή dpkg .Επομένως πρώτα τα μετατρέπουμε ξανά σε deb χωρίς φακέλους zst.(https://unix.stackexchange.com/questions/669004/zst-compression-not-supported-by-apt-dpkg)

7)Στην συνέχεια στέλνουμε τα τρία αρχεία deb από το host στο Qemu μέσω της εντολής :

```
sudo scp -P 22223 ~/Desktop/my_Qemu/new_deb/linux-headers-3.16.84_3.16.84-2_armhf.deb\  ~/Desktop/my_Qemu/new_deb/linux-image-3.16.84_3.16.84-2_armhf.deb \
      ~/Desktop/my_Qemu/new_deb/linux-libc-dev_3.16.84-2_armhf.deb \
      root@localhost:/usr/src
```




```
sudo apt-get upgrade dpkg
sudo apt-get upgrade bzip2
```

ΑΦΟΥ τα κάνουμε όλα dpkg -i package_name.deb μεταφέρουμε τα αρχεία vmlinuz-3.16.84 και initrd.img-3.16.84 στον host απο το qemu.
eg
scp -P 22223 root@localhost:/boot/vmlinuz-3.16.84 ~/Desktop

8) Φτιάχνουμε νέο φάκελο με τα νέα kernel και image

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

και τρέχουμε το “νέο” qemu!

Ερώτημα 1 (uname -a)
πριν :Linux debian-armhf 3.2.0-4-vexpress #1 SMP Debian 3.2.51-1 armv7l GNU/Linux
μετά:Linux debian-armhf 3.16.84 #2 SMP Tue Feb 13 20:03:02 EET 2024 armv7l GNU/Linux

Παρατηρήσεις : Αλλαγή της έκδοσης του πυρήνα Linux από 3.2.0-4-vexpress σε 3.16.84 ----> **Αναβάθμιση Πυρήνα**

Ερώτημα 2 : Προσθήκη νέου System Call σε αρχιτεκτονική τύπου arm.

Link1 :https://www.csd.uoc.gr/~hy345/assignments/2012/tutorial_5_2012.pdf)
Link2: https://stackoverflow.com/questions/32930775/cannot-create-new-system-call-on-raspberry-pi)

Αρχικά πάμε να δημιουργήσουμε deb αρχεία τα οποία θα υποστηρίζουν το δικό μας system call,αυτό θα γίνει στο host αφού θα χρειαστούμε make.

Step 1: Add new system call number

cd linux-source-3.16/arch/arm/include/asm# vim unistd.h
#define __NR_syscalls (389) // it was 388

Βέβαια τελικά θα ανακαλύψουμε στην συνέχεια ότι δεν χρησιμοποιούνται όλες οι θέσεις για systemcalls αρα τα 388 αρκούν. Διαφορετικά αν τα αυξήσουμε χωρίς να χρειάζεται θα έχουμε error ότι δεν ταυτίζεται το πλήθος των system calls που έχουμε ορίσει με αυτό που υπάρχει στο table.

ΆΡΑ το αφήνουμε όπως είναι : #define __NR_syscalls (388)

Step 2: Add new entry to system call table

cd linux-source-3.16/arch/arm/kernel# **vim calls.s**

```
                CALL(sys_finit_module)
/* 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_dummy)
#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 όπως φαίνεται παραπάνω το δικό μας sys_call sys_dummy συνάρτηση θα υλοποιείτο dummy system call.

Παρατηρούμε ότι το system call που δημιουργήσαμε βρίσκεται στην 386 των system calls.

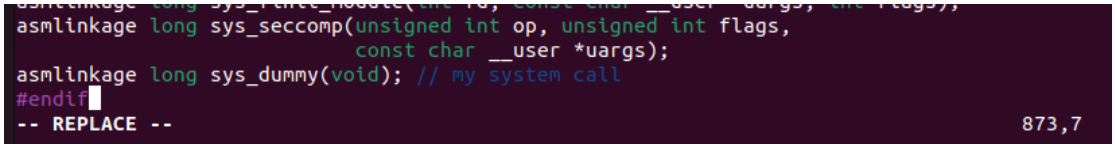
Step 3: Implement the system call’s function

finally implemented the syscall in arch/arm/kernel/sys_arm.c with the name sys_dummy

```
asmlinkage long sys_dummy(void)
 {
  printk("Greeting from kernel and team K&X aka no 9");
  return 0;
}
```

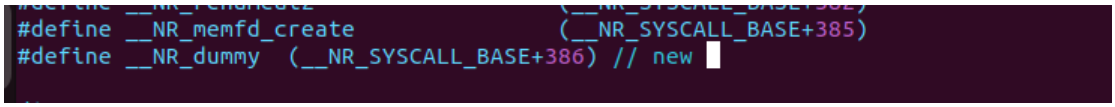
STEP 4: Define the sys call function

cd linux-source-3.16/include/linux#
vim syscalls.h



step 5:Define the index of my system call

cd linux-source-3.16/arch/arm/include/uapi/asm
vim unistd.h



Τελικά από μόνο του είχε νούμερο 388 και τα syscalls ήταν 385 οπότε απλώς πρόσθεσα ενα στο step 5 δεν χρειάστηκε να αλλάξω το πλήθος των sys calls.

/* same με την δημιουργία νεου kernel */

Τώρα έχουμε ένα νέο αρχείο linux-source το οποίο περιέχει το νέο system call ,και χρειάζεται να το κάνουμε ξανά make.

cd linux-source // το νέο
make ARCH=arm CROSS_COMPILE=~/linaro/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/bin/arm-linux-gnueabihf-

ωραία έγινε το build τώρα κάνουμε το make που δημιουργεί τα πρώτα 3 deb αρχεία τα deb τα αναδιαμορφώνουμε με την μέθοδο που αναφέραμε προηγουμένως.

τα στέλνουμε στο qemu
ΑΦΟΥ τα κανουμε όλα dpkg -i package_name.deb
μεταφέρουμε τα ΝΕΑ αρχεία vmlinuz-3.16.84 και initrd.img-3.16.84 στον host απο το qemu και με αυτά τρέχουμε το νεο VM στο QEMU
sudo scp -P 22223 /home/kostis/Desktop/my_syscall_build/recreated_debs/linux-image-3.16.84_3.16.84-7_armhf.deb root@localhost:/usr/src (X3)

ΑΦΟΥ τα κανουμε όλα dpkg -i package_name.deb

μεταφέρουμε τα αρχεία vmlinuz-3.16.84 και initrd.img-3.16.84
στον host απο το qemu
eg
scp -P 22223 root@localhost:/boot/vmlinuz-3.16.84 ~/Desktop

/* end δημιουργίας νεου kernel */

Έχουμε λοιπόν τα αρχεία τα αρχεία vmlinuz-3.16.84 και initrd.img-3.16.84
για να τρέξουμε VM με φορτωμένο στο syscall μας μένει μόνο να το δοκιμάσουμε.

Ερώτηση 3 Test system call in qemu

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

#define my_sys_call 386

int main(void) {
    long result;
    print("Here i call my dummy sys call...\n");
    result = syscall(my_sys_call);
    printf("the system call return: %d. \n",result);
    return 0;
}
```

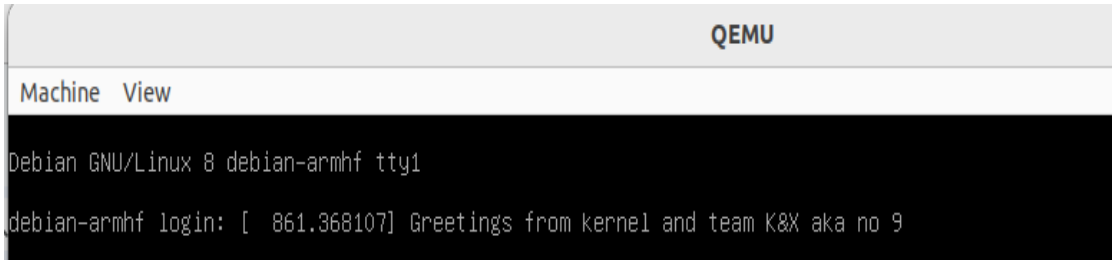
Το κάνουμε compile με linaro και στέλνουμε το εκτελέσιμο στο QEMU

~/linaro/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/bin/arm-linux-gnueabihf-gcc ~/tester_syscall.c -o ~/test_sycall.out

το μεταφέρουμε στο qemu με
sudo scp -P 22223 /home/kostis/test_sycall.out root@localhost:/usr/src

Αφού τρέξουμε το εκτελέσιμο όπως φαίνεται παρακάτω διαπιστώνουμε πως πράγματι το system_call λειτουργεί κανονικά

```
test_sycall.out
root@debian-armhf:~# ./test_sycall.out
Here i call my dummy sys call...
the system call return: 0.
root@debian-armhf:~#
```



Σημειώνουμε ότι στο zip θα περιέχονται τόσο τα αρχεία που παράχθηκαν για κάθε kernel για να «τρέκξει» το qemu όσο και το ίδιο το linus-source που περιέχει τις αλλαγές στα κατάλληλα αρχεία για να προσθέσουμε το system call.