Σχεδιασμός Ενσωματωμένων Συστημάτων
9ο εξάμηνο ΣΗΜΜΥ
5η εργαστηριακή άσκηση

Ομάδα 6:
Δούκας Θωμάς Α.Μ.: 03116081
Ψαρράς Ιωάννης Α.Μ.:03116033

Στην πέμπτη εργαστηριακή άσκηση θα πραγματοποιήσουμε cross-compiling προγραμμάτων για συγκεκριμένη ARM αρχιτεκτονική. Συγκεκριμένα, θα εγκαταστήσουμε το crosstool-ng που αποτελεί έναν cross-compiler builder με σκοπό να χτίσουμε έναν cross-compiler. Εκτός από τη χρήση του custom compiler toolchain θα χρησιμοποιήσουμε και έναν pre-compiled cross-compiler που παρέχεται από την ιστοσελίδα *www.linaro.org*.

Άσκηση 1

Για την πρώτη άσκηση θα παρουσιάσουμε τα προβλήματα που προέκυψαν κατά τη διαδικασία εγκατάστασης και κατά το χτίσιμο του cross compiler, αλλά και τις ενέργειες που πραγματοποιήθηκαν προκειμένου να επιλυθούν. Στη συνέχεια καλούμαστε να επιλύσουμε συγκεκριμένες τεχνικές και θεωρητικές απορίες.

Προβλήματα και Επίλυση

Αρχικά, δημιουργήσαμε ένα δεύτερο εικονικό μηχάνημα στο QEMU χρησιμοποιώντας τα αρχεία που παρατίθενται στην εκφώνηση. Ακολουθώντας τα βήματα του αντίστοιχου οδηγού που δόθηκε στην 3η εργαστηριακή άσκηση εκκινούμε το εικονικό μηχάνημα. Κατά την εκτέλεση των παραπάνω δεν παρατηρήθηκαν προβλήματα.

Στο σημείο αυτό αναφέρουμε τα προβλήματα που παρουσιάστηκαν στη διαδικασία εγκατάστασης του custom cross-compiler building toolchain crosstool-ng. Τα παραπάνω οφείλονται σε έλλειψη απαιτούμενων πακέτων από το σύστημα μας. Εγκατάσταση των πακέτων αυτών μας επέτρεψε να χτίσουμε τον cross-compiler ώστε να δημιουργηθεί ο φάκελος *$HOME/x-tools/arm-cortexa9_neon-linux-gnueabihf* με τα απαραίτητα αρχεία.

Για το βήμα 1 της διαδικασίας παρατηρήσαμε το μήνυμα:
./bootstrap: line 830: autoreconf: command not found

Η εντολή ***sudo apt-get install autoconf*** επιτρέπει την εκτέλεση του bootstrap χωρίς την εμφάνιση παρόμοιων μηνυμάτων.

Επιπλέον, κατά το configure του βήματος 4 παρατηρήσαμε ότι χρειάστηκε η εγκατάσταση αρκετών πακέτων ώστε να ολοκληρωθεί επιτυχώς η διαδικασία. Συνοπτικά, εκτελέσαμε τα παρακάτω:

sudo apt-get install flex
sudo apt-get install texinfo
sudo apt-get install help2man
sudo apt-get install gawk

sudo apt-get install libtool-bin
sudo apt-get install ncurses-dev

Τέλος, για την επιτυχή εκτέλεση των make και make install εγκαταστήσαμε το πακέτο bison ***sudo apt-get install bison.***

Αναφέρουμε ότι κατά την διαδικασία εγκατάστασης pre-compiled building toolchain linaro δεν εμφανίστηκαν προβλήματα. Ωστόσο κατά την προσπάθεια μας να το χρησιμοποιήσουμε ώστε να μεταγλωττιστεί ο κώδικας *phods.c* στο πρώτο ερώτημα παρατηρήθηκε το εξής error παρά την ύπαρξη του αρχείου:
bash: ~/linaro/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/bin/arm-linux-gnueabihf-gcc No such file or directory

Το παραπάνω πρόβλημα επιλύθηκε με την εκτέλεση της εντολής:
sudo apt-get install lib32z1-dev

Ερωτήσεις

1. Η διαφορά του image που χρησιμοποιούμε στη συγκεκριμένη άσκηση από εκείνο της τρίτης εργαστηριακής άσκησης οφείλεται στο γεγονός ότι κάθε εικόνα αναφέρεται σε διαφορετική αρχιτεκτονική ενός ARM επεξεργαστή. Συγκεκριμένα με το image *debian_wheezy_armhf* που κατεβάσαμε μπορούμε μέσω του QEMU να προσομοιώσουμε ένα σύστημα ARM αρχιτεκτονικής armhf. Αντιθέτως η εικόνα της εργαστηριακής άσκησης 3 χρησιμοποιείται για την αρχιτεκτονική armel.

   Οι 2 αρχιτεκτονικές παρουσιάζουν σημαντικές διαφορές με την armel να αναφέρεται σε παλαιότερους 32bit little endian ARM επεξεργαστές ενώ το armhf (arm hard float) αναφέρεται σε νεότερους arm επεξεργαστές οι οποίοι χαρακτηρίζονται από hardware floating point support.

2. Χρησιμοποιούμε την συγκεκριμένη αρχιτεκτονική καθώς αντιστοιχεί σε εκείνη που προσομοιώνεται με τη βοήθεια του QEMU. Με αυτό τον τρόπο το εκτελέσιμο αρχείο θα μπορεί να εκτελεστεί επιτυχώς από το target machine, δηλαδή το εικονικό μηχάνημα που δημιουργήσαμε. Σε αντίθετη περίπτωση, επιλογής διαφορετικής αρχιτεκτονικής από το list samples, το εκτελέσιμο που δημιουργείται από το toolchain δεν θα μπορέσει να εκτελεστεί στο QEMU.

3. Εκτελώντας την εντολή ***ldd arm-cortexa9_neon-linux-gnueabihf-gcc*** μέσα στο φάκελο *~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin* εμφανίζεται το αποτέλεσμα:

   linux-vdso.so.1 (0x00007fff533f5000)
   libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f731084f000)
   /lib64/ld-linux-x86-64.so.2 (0x00007f7310a54000)

   Επομένως στο βήμα 9 χρησιμοποιήθηκε η default βιβλιοθήκη **glibc**. Η συγκεκριμένη βιβλιοθήκη είναι η πιο διαδεδομένη και χρησιμοποιείται με στόχο την βελτιστοποίηση του performance. Στα πλαίσια της συγκεκριμένης άσκησης η παραπάνω βιβλιοθήκη μας καλύπτει, καθώς παράγει αρχείο που εκτελείται στο target system.

Εναλλακτικά στην περίπτωση των ενσωματωμένων συστημάτων ιδιαίτερα χρήσιμη κρίνεται και η βιβλιοθήκη *uClibc* η οποία αποτελεί μια πιο ελαφριά έκδοση. Η συγκεκριμένη εστιάζει στην παραγωγή μικρότερων σε μέγεθος εκτελέσιμων αρχείων, γεγονός που ευνοεί αρκετά την εξοικονόμηση της χρησιμοποιούμενης μνήμης. Σημειώνεται, βέβαια, ότι η ελαχιστοποίηση μεγέθους των αρχείων εμποδίζει την επίτευξη της μέγιστης δυνατής επίδοσης.

4. Προκειμένου να κάνουμε compile τον κώδικα εκτελούμε την εντολή ***~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin/arm-cortexa9_neon-linux-gnueabihf-gcc phods.c -Wall -O0 -o phods_crosstool.out***

   Η εντολή *file phods_crosstool.out* εμφανίζει:

   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

   Συγκεκριμένα, μας δίνει πληροφορίες για το τύπο του αρχείου και αναγνωρίζει ότι πρόκειται για εκτελέσιμο. Ακολούθως αναφέρει την αρχιτεκτονική, Little Endian η Big Endian, έκδοση EABI, αν το αρχείο είναι dynamicaly η statically linked, το path του interpreter, την έκδοση του πυρήνα αλλά και αν είναι stripped, αν περιέχονται δηλαδή ή όχι πληροφορίες αποσφαλμάτωσης.

   Αντίστοιχα η εντολή *readelf -h -A phods_crosstool.out* εμφανίζει:

   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: 0x1045c
   Start of program headers: 52 (bytes into file)
   Start of section headers: 12608 (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: 37
   Section header string table index: 36
   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 μας δίνει πληροφορίες συγκεκριμένα για αρχεία τύπου ELF. Με τις σημαίες -h(*--file-header*) και -A(*--arch-specific*) που έχουμε ενεργοποιήσει, ζητάμε όσες πληροφορίες περιέχονται στο ELF header του αρχείου και όσες αναφέρονται στην αρχιτεκτονική του.

5. Μεταγλωττίζουμε των κώδικα με την εντολή *~/linaro/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/bin/arm-linux-gnueabihf-gcc phods.c -Wall -O0 -o phods_linaro.out*

   Παρατηρούμε ότι υπάρχει διαφορά στο μέγεθος των αρχείων

   -rwxrwxr-x 1 thomas thomas **14088** Ιαν 5 18:28 phods_crosstool.out
   -rwxrwxr-x 1 thomas thomas **8106** Ιαν 5 19:28 phods_linaro.out

   Με την εντολή ***ldd arm-linux-gnueabihf-gcc*** στον φάκελο *~/linaro/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/bin* παρατηρούμε πως χρησιμοποιείται βιβλιοθήκη C των 32-bit.

   linux-gate.so.1 (0xf7f38000)
   libstdc++.so.6 => /lib32/libstdc++.so.6 (0xf7d42000)
   libm.so.6 => /lib32/libm.so.6 (0xf7c3e000)
   libpthread.so.0 => /lib32/libpthread.so.0 (0xf7c1b000)
   libdl.so.2 => /lib32/libdl.so.2 (0xf7c15000)
   libgcc_s.so.1 => /lib32/libgcc_s.so.1 (0xf7bf6000)
   libc.so.6 => /lib32/libc.so.6 (0xf7a0b000)
   /lib/ld-linux.so.2 (0xf7f39000)

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 glibc.

6. Με την βοήθεια της εντολής file και για τις δύο περιπτώσεις (*phods_crosstool.out*, *phods_linaro.out*) παρατηρούμε ότι πρόκειται για αρχεία ELF 32-bit. Παρά το γεγονός ότι χρησιμοποιούνται διαφορετικές βιβλιοθήκες, 64 και 32, το εκτελέσιμο αρχείο προορίζεται για target machine αρχιτεκτονικής των 32bit. Επομένως καμία από τις 2 υλοποιήσεις δεν θα παρουσιάσει πρόβλημα κατά την εκτέλεση της.

7. Για την επιλογή static linking εκτελούμε αντίστοιχα τις εντολές:
   ~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin/arm-cortexa9_neon-linux-gnueabihf-gcc phods.c -Wall -O0 -static -o phods_static_crosstool.out

   ~/linaro/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/bin/arm-linux-gnueabihf-gcc phods.c -Wall -O0 -static -o phods_static_linaro.out

   Τα μεγέθη των παραγόμενων αρχείων παρατίθενται στη συνέχεια:
   -rwxrwxr-x 1 thomas thomas ***14088*** *Ιαν 5 20:36 phods_crosstool.out*
   -rwxrwxr-x 1 thomas thomas ***8106*** *Ιαν 5 19:28 phods_linaro.out*
   -rwxrwxr-x 1 thomas thomas ***2923052*** *Ιαν 5 20:36 phods_static_crosstool.out*
   -rwxrwxr-x 1 thomas thomas ***507853*** *Ιαν 5 20:37 phods_static_linaro.out*

   Όπως είναι αναμενόμενο τα statically compiled εκτελέσιμα είναι σημαντικά μεγαλύτερα από τα dynamically compiled. Αυτό συμβαίνει καθώς στην πρώτη περίπτωση, ο μεταγλωττιστής ενσωματώνει όλες τις συναρτήσεις εξωτερικών βιβλιοθηκών που χρησιμοποιούνται από τον κώδικα phods.c μέσα στο ίδιο εκτελέσιμο. Η αντιμετώπιση αυτή, αν και αυξάνει σημαντικά το μέγεθος του αρχείου, μειώνει τον χρόνο εκτέλεσης καθώς δεν χρειάζεται να κληθούν εξωτερικές βιβλιοθήκες, κατά το runtime. Αντιθέτως, στη περίπτωση του dynamic linking, οι κλήσεις συναρτήσεων από εξωτερικές βιβλιοθήκες πραγματοποιούνται κατά την διάρκεια του runtime.

8.
   A. Αναμένουμε ότι το αρχείο που παράγεται **δεν μπορεί να εκτελεστεί** στο host μηχάνημα καθώς έχουμε πραγματοποιήσει διαδικασία cross-compilation που προορίζεται για target machine.

   B. Και πάλι δεν περιμένουμε το αρχείο να εκτελεστεί με επιτυχία. Αυτό συμβαίνει καθώς πραγματοποιείται dynamic compiling γεγονός που σημαίνει ότι η συνάρτηση που υλοποιήσαμε και προσθέσαμε στη βιβλιοθήκη glibc δεν ενσωματώνεται στο εκτελέσιμο αρχείο. Έτσι, όταν επιδιώξουμε την εκτέλεση του στο target μηχάνημα, αυτό θα προσπαθήσει να εντοπίσει την συνάρτηση *mlab_foo()* μέσα στην βιβλιοθήκη glibc. Ωστόσο η ανανεωμένη glibc υπάρχει μόνο στο host μηχάνημα μας, και άρα δεν είναι δυνατή η εκτέλεση στο target.

   C. Στην περίπτωση του static compiling η συνάρτηση *mlab_foo()* θα ενσωματωθεί στο εκτελέσιμο αρχείο. Έτσι, κατά την διάρκεια του runtime καμία από τις

συναρτήσεις που χρειάζεται ο κώδικας phods.c δεν απαιτείται να κληθεί από την αντίστοιχη βιβλιοθήκη της. Επομένως ο κώδικας θα εκτελεστεί κανονικά.

Άσκηση 2

Στην 2η άσκηση θα χτίσουμε έναν νέο πυρήνα για το Debian OS που εγκαταστήσαμε προηγουμένως. Για να επιτευχθεί αυτό ακολουθούμε τα βήματα, όπως αυτά περιγράφονται από την εκφώνηση της άσκησης.

Επιλέγουμε να χρησιμοποιήσουμε τον cross-compiler crosstool από την άσκηση 1. Κατεβάζοντας τα απαραίτητα αρχεία όπως αναφέρονται από την εκφώνηση κατασκευάζουμε το armhf Qemu ακολουθώντας τις οδηγίες της άσκησης 3. Δεν παρουσιάστηκε κανένα πρόβλημα στη συγκεκριμένη διαδικασία.

Στο φάκελο boot του VM παρατηρούμε τα αρχεία:

config-3.2.0-4-vexpress
initrd.img-3.2.0-4-vexpress
System.map-3.2.0-4-vexpress
vmlinuz-3.2.0-4-vexpress
initrd.img
lost+found
vmlinuz

Κατεβάζουμε τον πηγαίο κώδικα που μας παρέχει η Debian στο guest μηχάνημα εκτελώντας τις εντολές *apt-get update* και *apt-get install linux-source*. Στο σημείο αυτό παρουσιάστηκαν προβλήματα σχετικά με μη διαθέσιμα public κλειδιά. Ακολουθώντας τις οδηγίες της άσκησης 3 εκτελούμε την *apt-key adv --keyserver keyserver.ubuntu.com –recv-keys <key>* για τα αντίστοιχα κλειδιά. Η διαδικασία ολοκληρώθηκε με επιτυχία καθώς το αρχείο linux-source-3.16.tar.xz υπάρχει στο directory */usr/src*.

Στα βήματα 4,5,6 και 7 δεν παρουσιάστηκε κανένα πρόβλημα. Αναφέρουμε ότι επιβεβαιώσαμε την επιτυχή εκτέλεση του patch ελέγχοντας χειροκίνητα τις αλλαγές στο αρχείο *scripts/package/builddeb.*

Στο βήμα 8 μεταφέρουμε τα *.deb* αρχεία *linux-image*, *linux-libc-dev* και *linux-headers* στο guest μηχάνημα. Για την επιτυχή εκτέλεση της εντολής dpkg -i package_name.deb χρειάστηκε να προηγηθεί η εντολή *apt-get upgrade bzip2*.

Ερωτήσεις

1. Στο περιβάλλον Qemu εκτελούμε την εντολή *uname -a* προκειμένου να λάβουμε πληροφορίες για τον πυρήνα που χρησιμοποιούμε. Για την περίπτωση του πυρήνα που χρησιμοποιεί ο Qemu προτού εγκαταστήσουμε τον καινούργιο τα αποτελέσματα που προκύπτουν είναι:

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

Εκτελώντας την εντολή για τον πυρήνα που δημιουργήσαμε παρατηρούμε:

root@debian-armhf:~# uname -a
Linux debian-armhf 3.16.84 #4 SMP Mon Jan 11 22:04:14 EET 2021 armv7l GNU/Linux

Από τα αποτελέσματα της παραπάνω εντολής παρατηρούμε ότι έχει αλλάξει η έκδοση του πυρήνα που χρησιμοποιείται.

2. Για την κλήση συστήματος που θα υλοποιήσουμε επιλέγουμε το όνομα ***sys_hello()***. Στη συνέχεια, δημιουργούμε έναν φάκελο *linux-source/**custom*** μέσα στον οποίο υλοποιούμε την συνάρτηση **hello.c** και το αντίστοιχο Makefile για την μετατροπή της σε object file.

linux-source-3.16 > custom > C hello.c > ...

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

asmlinkage long sys_hello(void)
{
        //printk prints to the kernel's log file.
        printk("Greetings from kernel and team no 6\n");
        return 0;
}
```

linux-source-3.16 > custom > M Makefile

```
obj-y := hello.o
```

Στη συνέχεια πρέπει να προσθέσουμε την κλήση συστήματος που υλοποιούμε στον *architecture-specific syscall table*. Στο αρχείο **linux-source/arch/arm/kernel/calls.S** προσθέτουμε την εντολή **CALL(sys_hello).** Επιλέγουμε συγκεκριμένο index για την εντολή μας. Στη περίπτωση μας επιλέξαμε το #386 που δεν χρησιμοποιείται από καμία άλλη.

```
/* 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

```

Στο αρχείο **linux-source/arch/arm/include/asm/unistd.h** προσθέτουμε:

```
56   #define __NR_sys_hello (__NR_SYSCALL_BASE+386)
```

Στο αρχείο **linux-source/include/linux/syscall.h** προσθέτουμε τον ορισμό της συνάρτησης μας.

```
872  asmlinkage long sys_hello(void);
873  #endif
874
```

Τέλος συμπεριλαμβάνουμε το directory με την υλοποίηση του syscall μας στο φάκελο custom, στο **root-level Makefile** του kernel-source.

```
859   core-y      += kernel/ mm/ fs/ ipc/ security/ crypto/ block/ custom/
```

Ακολούθως επαναλαμβάνουμε τη διαδικασία για να πραγματοποιήσουμε make του πυρήνα. Κατεβάζουμε τα αρχεία ***initrd.img-3.16.84*** και ***vmlinuz-3.16.84*** που χρειάζονται και εκκινούμε το qemu με την εντολή:

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

3. Για να διαπιστώσουμε την ομαλή λειτουργία της υλοποιημένης συνάρτησης μας συντάσσουμε το πρόγραμμα *test_syscall.c*:

home > thomas > GuestQemuESD > newDebian > C test_syscall.c > ...

```c
#include <unistd.h>
#include <sys/syscall.h>
#include <stdio.h>
#define sys_hello 386

int main(void) {
    printf("Calling sys_hello...\n");
    long ret = syscall(sys_hello);
    printf("Return value %ld.\n", ret);
    return 0;
}
```

Για να μπορέσει το πρόγραμμα να εκτελεστεί με επιτυχία στο target μηχάνημα θα χρειαστεί να το μεταγλωττίσουμε. Ακολουθούμε την διαδικασία της άσκησης 1 και εκτελούμε την εντολή
~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin/arm-cortexa9_neon-linux-gnueabihf-gcc test_sys_hello.c -Wall -O0 -o test_sys_hello.out

Αρκεί να μεταφέρουμε το εκτελέσιμο αρχείο *test_sys_hello.out* στο guest και με ./test_sys_hello εμφανίζονται τα παρακάτω αποτελέσματα. Το μήνυμα εμφανίζεται στο παράθυρο του Qemu και ως αποτέλεσμα της εντολής **dmesg | tail**.


root@debian-armhf:~# ./test_sys_hello.out
Calling sys_hello...
Return value 0.
root@debian-armhf:~# dmesg | tail
[75.277130] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[76.249777] IPv6: ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready
[82.730617] RPC: Registered named UNIX socket transport module.
[82.735882] RPC: Registered udp transport module.
[82.738519] RPC: Registered tcp transport module.
[82.741021] RPC: Registered tcp NFSv4.1 backchannel transport module.
[82.868978] FS-Cache: Loaded
[83.036174] FS-Cache: Netfs 'nfs' registered for caching
[83.419560] Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
[165.993086] Greetings from kernel and team no 6
root@debian-armhf:~#
Debian GNU/Linux 8 debian-armhf tty1
debian-armhf login: [165.993086] Greetings from kernel and team no 6