Σχεδιασμός Ενσωματωμένων Συστημάτων

Φιλιππόπουλος Ορφέας el18082
Παπαρρηγόπουλος Θοδωρής el18040

6η Εργαστηριακή Άσκηση

Προετοιμασία για την άσκηση

Το πρώτο βήμα που κάναμε ήταν να εγκαταστήσουμε τα απαραίτητα αρχεία που δίνονταν στην άσκηση.

Στη συνέχεια εκκινήσαμε το VM με τη παρακάτω εντολή:

```
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
```

Αφού το εκκινήσαμε επιτυχώς, συνδεθήκαμε σε αυτό με ssh. Για κάποιο λόγο εκεί είχαμε ένα περίεργο error:

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:vsd1d4bIDIejcLXSmxMbWQRifTXk5s5hgqU6Ii6E2+E.
Please contact your system administrator.
Add correct host key in /home/hdron/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/hdron/.ssh/known_hosts:34
  remove with:
  ssh-keygen -f "/home/hdron/.ssh/known_hosts" -R "[localhost]:22223"
ECDSA host key for [localhost]:22223 has changed and you have requested strict checking.
Host key verification failed.
```

Το οποίο και λύθηκε εύκολα εκτελώντας την εντολή ssh με sudo.

Τέλος αντικαταστήσαμε το αρχείο sources.list μέσα στο VM, με αυτό που μας δίνεται στην εκφώνηση της άσκησης.

Παρακάτω θα δείξουμε τα βήματα εγκατάστασης ενός custom cross-compiler με χρήση του crosstool-ng καθώς και την εγκατάσταση ενός pre-compiled cross-compiler.

Εγκατάσταση custom cross-compiler building toolchain crosstool-ng

1.

Αρχικά κατεβάσαμε τα απαραίτητα αρχεία για το κτίσιμο του toolchain από το αντίστοιχο github repository στο host μηχάνημά. Εκτελούμε την εντολή:

```
git clone https://github.com/crosstool-ng/crosstool-ng.git
```

2.

Στη συνέχεια, στο φάκελο που δημιούργησε η προηγούμενη εντολή, εκτελέσαμε την εντολή :

```
/crosstool-ng$ ./boostrap
```

Εδώ είχαμε ένα θέμα, καθώς μας ζητούσε autoconf από 2.71 version και πάνω (είχαμε 2.69) και για αυτό το λόγο εκτελέσαμε:

dpkg –remove autoconf

και στη συνέχεια κατεβάσαμε ένα dpkg αρχείο του autoconf που ήταν έκδοση 2.71 με την εντολή wget και το URL:

http://archive.ubuntu.com/ubuntu/pool/main/a/autoconf/autoconf_2.71-all.deb

3.

Στη συνέχεια εκτελέσαμε την εντολή:

```
/crosstool-ng$ mkdir $HOME/crosstool && mkdir $HOME/src
```

Στο πρώτο φάκελο (crosstool) θα εγκατασταθεί το πρόγραμμα crosstool-ng ενώ στο δεύτερο θα αποθηκεύει τα απαραίτητα πακέτα που κατεβάζει για να χτίσει τον cross-compiler.

4.

Εκτελούμε τώρα την παρακάτω εντολή για να κάνουμε configure την εγκατάσταση του crosstool-ng:

```
/crosstool-ng$ ./configure --prefix=${HOME}/crosstool
```

Κατά τη διάρκεια εκτέλεσης αυτής της εντολής μας έλλειπαν κάποια πακέτα. Για παράδειγμα έλλειπε η βιβλιοθήκη curse την οποία και κατεβάσαμε με την εντολή

```
sudo apt-get install libncurses5-dev libncursesw5-dev
```

5.

Στη συνέχεια εκτελέσαμε την εντολή:

```
/crosstool-ng$ make && make install
```

και έτσι εγκαταστήσαμε το crosstool-ng στο φάκελο crosstool που προαναφέραμε.

6.

Πηγαίνουμε στο φάκελο crosstool/bin στον οποίο θα κάνουμε και build τον compiler μας.

```
/crosstool-ng$ cd $HOME/crosstool/bin
```

7.

Τώρα, θα τυπώσουμε του συνδυασμούς αρχιτεκτονικών, λειτουργικών συστημάτων και βιβλιοθηκών της C που παρέχονται από το εργαλείο. Αυτό το πετυχαίνουμε με την εντολή:

```
/crosstool/bin$ ./ct-ng list-samples
```

Τα αποτελέσματα της εντολής αυτής είναι αρκετά, ωστόσο εμείς θα επιλέξουμε τον συνδυασμό:

```
[G..X]   arm-cortexa9_neon-linux-gnueabihf
```

8.

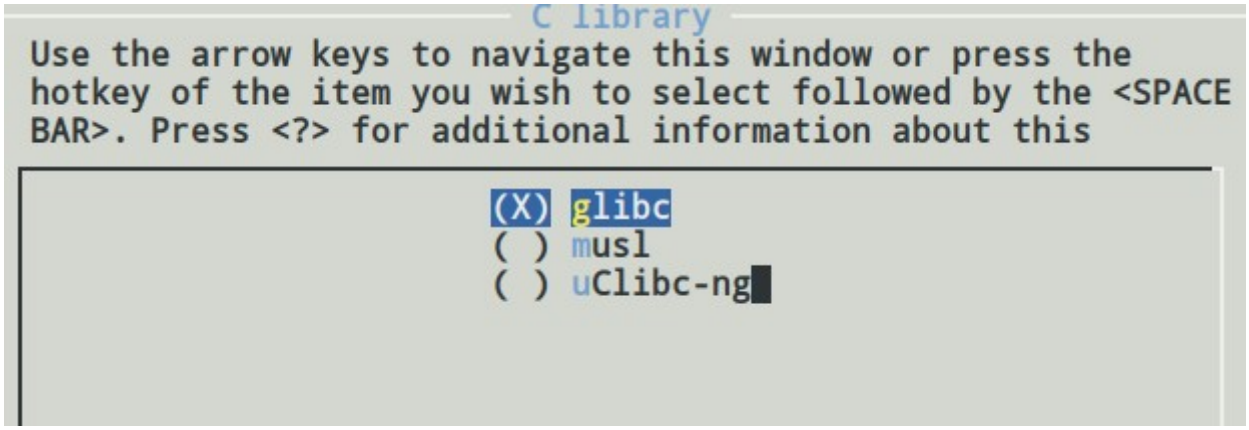
Προκειμένου να επιλέξουμε και να παραμετροποιήσουμε το crosstool-ng για τη συγκεκριμένη αρχιτεκτονική εκτελούμε την εντολή:

```
/crosstool/bin$ ./ct-ng menuconfig
```

9.

Ωστόσο, εάν θέλουμε να αλλάξουμε με γραφικό τρόπο κάποια χαρακτηριστικά του preconfigured συνδυασμού target machine, όπως για παράδειγμα ποια βιβλιοθήκη της C θα χρησιμοποιήσουμε, μπορούμε να εκτελέσουμε την εντολή:

```
/crosstool/bin$ ./ct-ng menuconfig
```



Για παράδειγμα οι βιβλιοθήκες που μας παρέχονται είναι: glibc, musl, uClibc-ng

Παρατηρούμε ότι η musl είναι compatible και με **armhf** και με **armel,** καθώς στο documentation λέει:

musl-tools (1.2.3-1+b1 [riscv64], 1.2.3-1 [amd64, arm64, armel, armhf, i386, m68k, mips64el, mipsel, ppc64el, s390x, sh4])

όπου περιέχεται και το armel και το armhf. Επίσης είναι compatible και με linux.

Όσον αφορά τη uClibc-ng και αυτή είναι compatible με armel και armhf αλλά και με linux.

10.

Αφού έχουμε κάνει όλα τα παραπάνω, θα build-αρουμε τον cross-compiler μας με την εντολή:

```
/crosstool/bin$ ./ct-ng build
```

Διήρκεσε αρκετή ώρα (περίπου μιάμιση ώρα).

11.

Τέλος, έχει δημιουργηθεί ο φάκελος:

$HOME/x-tools/arm-cortexa9_neon-linux-gnueabihf

όπου μέσα στον υποφάκελο bin περιέχει τα εκτελέσιμα αρχεία του cross-compiler μας.

Εγκατάσταση pre-compiled building toolchain linaro

1.

Αρχικά κατεβάζουμε τα binaries από το host μηχάνημα με τη παρακάτω εντολή:

```
:~$ mkdir ~/linaro && cd ~/linaro
:~/linaro$ wget https://releases.linaro.org/archive/14.04/components/toolchain/binaries/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux.tar.bz2
```

2.

Αφού κατεβάσαμε τα παραπάνω tarball, το κάνουμε extract με την εντολή:

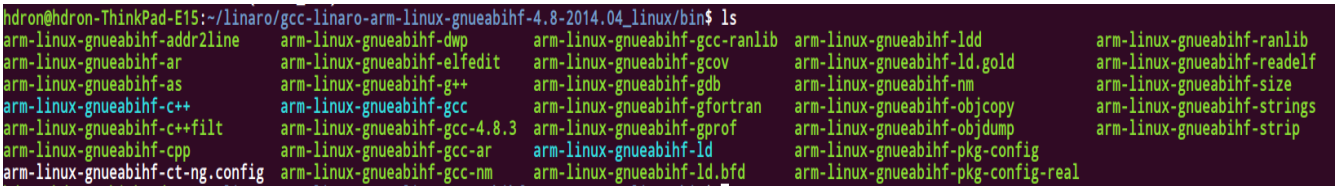
```
:~/linaro$ tar -xvf gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux.tar.bz2
```

3.

Παρατηρούμε ότι τα binaries του cross-compiler βρίσκονται στο πακέτο που κατεβάσαμε στο φάκελο:

$HOME/linaro/gcc-linaro-arm-linux-gnueabihf-4.8-2014.04_linux/bin

Κάνοντας και ένα ls στο παραπάνω φάκελο:



Άσκηση 1

1.

Η **armel (**η ARM EABI, **soft float**) υποστηρίζει τις ARM αχιτεκτονικές ARM4T, ARM5T και ARM6. Υποστηρίζει floating-poing hardware μέσω του compatibility mode. Αυτό χειροτερεύει το performance αλλά επιτρέπει τη συμβατότητα με κώδικα που έχει γραφτεί για CPUs χωρίς FPUs.

Η **armhf** (**hard float**) υποστηρίζει αρχιτεκτονικές με FPUs σε πιο σύγχρονους 32-bit ARM επεξεργαστές. Αυτό είναι πολύ σημαντικό, καθώς οι FPUs είναι πολύ σημαντικές όταν υπάρχουν απαιτήσεις ακρίβειας στους υπολογισμούς. Οι αρχιτεκτονικές που υποστηρίζει είναι η ARMv7 και μετά.

Και οι 2 αρχιτεκτονικές είναι σε little endian.

2.

Η αρχιτεκτονική που χρησιμοποιήσαμε είναι η:

arm-cortexa9_neon-linux-gnueabihf

Ακόμα, τη συγκεκριμένη αρχιτεκτονική έχει και το guest μηχάνημα (armhf αρχιτεκτονική). Εάν χρησιμοποιούσαμε κάποια διαφορετική αρχιτεκτονική για το cross-compiler μας, τότε όταν πηγαίναμε να τρέξουμε κάποιο εκτελέσιμο (που έχει παραχθεί από αυτόν), τότε θα μας έβγαζε error καθώς η αρχιτεκτονική για την οποία έχει παραχθεί (το εκτελέσιμο) είναι διαφορετική από αυτή που πάει να εκτελετεί (διαφορετικά ISAs).

3.

Εάν και συνειδητά επιλέξαμε στο βήμα 9 τη βιβλιοθήκη glibc, αυτό φαίνεται και άμα χρησιμοποιήσουμε την εντολή **ldd** σε οποιοδήποτε εκτελέσιμο μέσα στο **υποφάκελο bin** στο φάκελο **x-tools/arm-cortexa9_neon-linux-gnueabihf**, για παράδειγμα:

```
hdron@hdron-ThinkPad-E15:~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin$ ldd -v arm-cortexa9_neon-linux-gnueabihf-c++
        linux-vdso.so.1 (0x00007ffcf9fba000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f49e87b4000)
        /lib64/ld-linux-x86-64.so.2 (0x00007f49e89e4000)

        Version information:
        ./arm-cortexa9_neon-linux-gnueabihf-c++:
                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_2.9) => /lib/x86_64-linux-gnu/libc.so.6
                libc.so.6 (GLIBC_2.7) => /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
```

Παρατηρούμε ότι το συγκεκριμένο εκτελέσιμο έχει γίνει linked με πολλές δυναμικές βιβλιοθήκες της glibc.

4.

Εκτελούμε τα βήματα του ερωτήματος:

compile:
sudo /home/hdron/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin/arm-ortexa9_neon-linux-gnueabihf-gcc phods.c -o ../../../6th_lab/phods_crosstool.out

```
hdron@hdron-ThinkPad-E15:~/Documents/OrfeasDir/uni/9th_sem/embedded/Embedded-Systems-Ntua23/6th_lab$ 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
```

```
hdron@hdron-ThinkPad-E15:~/Documents/OrfeasDir/uni/9th_sem/embedded/Embedded-Systems-Ntua23/6th_lab$ 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:          72384 (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
```

Οι εντολές αυτές μας δίνουν πληροφορίες για το τύπο του αρχείου (32-bit ELF executable), για το ABI του (EABI version 1), για την target αρχιτεκτονική (ARM v7), ότι είναι dynamically linked με τη βιβλιοθήκη *lib*ld-linux-armhf.so.3. Επίσης μας δίνει και πολλές άλλες πληροφορίες για το ABI όπως ABI_FP_rounding: Needed (και πολλά άλλα για floating point (FP)), για το alignment κλπ.

5.

Παρατηρούμε ότι:

με το linaro έχουμε 12K byte αρχείο:

```
hdron@hdron-ThinkPad-E15:~/Documents/OrfeasDir/uni/9th_sem/embedded/Embedded-Systems-Ntua23/6th_lab/linaro$ du -h phods_crosstool.out
12K     phods_crosstool.out
```

ενώ με το custom cross compiler έχουμε 24K byte:

```
hdron@hdron-ThinkPad-E15:~/Documents/OrfeasDir/uni/9th_sem/embedded/Embedded-Systems-Ntua23/6th_lab/linaro$ du -h ../custom_cross_compiler/phods_crosstool.out
24K     ../custom_cross_compiler/phods_crosstool.out
```

Ο λόγος για τον οποίο το εκτελέσιμο με το linaro είναι μικρότερου μεγέθους είναι ότι χρησιμοποιεί 32 bit version της βιβλιοθήκης libc (i386-linux-gnu), ενώ ο custom cross compiler χρησιμοποιεί 64 bit version της βιβλιοθήκης libc (x86_64-linux-gnu).

6.

Η εκτέλεση είναι σωστή καθώς το εκτελέσιμο του 4ου ερωτήματος έχει δυναμική σύνδεση με την αντίστοιχη βιβλιοθήκη της libc. Επομένως, θα εκτελέσει τη συνάρτηση που θα βρει ο linker μέσα στο target μηχάνημα κατά το runtime.

7.

Χρησιμοποιώντας τον custom cross compiler έχουμε:

```
24K     ../custom_cross_compiler/phods_crosstool_dyn.out
2,9M    ../custom_cross_compiler/phods_crosstool_static.out
```

και χρησιμοποιώντας τον prebuild cross compiler έχουμε:

```
12K     phods_linaro_dyn.out
500K    phods_linaro_static.out
```

Προφανώς παρατηρούμε ότι το μέγεθος των αρχείων έχει αυξηθεί πολύ καθώς οι βιβλιοθήκες της C είναι πλέον statically linked με αποτέλεσμα να ενσωματωμένες στο ίδιο το εκτελέσιμο και να μην καλείται κατά το runtime ο linked προκειμένου να τις βρει.

Από τα παραπάνω μπορεί να πει κανείς πως εάν και ο compiler επηρεάζει το μέγεθος του εκτελέσιμου, ο πιο βασικός παράγοντας είναι το εάν θα κάνουμε στατικό η δυναμικό linking των βιβλιοθηκών της C.

8.

α) Δεν μπορεί να εκτελεστεί στο host μηχάνημα εκτός εάν το target μηχάνημα είναι ίδιας αρχιτεκτονικής με αυτό του host.

b) Δεν θα μπορέσει να εκτελεστεί λόγω του dynamic linking. Πιο συγκεκριμένα δεν θα μπορέσει στο target μηχάνημα ο linker να βρει τη συνάρτηση mlab_foo().

c) Θα μπορέσει να εκτελεστεί καθώς η συνάρτηση πλέον είναι ενσωματωμένη στο εκτελέσιμο που παρήγαγε ο cross compiler.

Άσκηση 2

1.

Πριν κατεβάσουμε το νέο πυρήνα η εντολή **uname -a** τυπώνει:

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

Αφού κατεβάσουμε το νέο πυρήνα η εντολή **uname -a** τυπώνει:

```
root@debian-armhf:~# uname -a
Linux debian-armhf 3.16.84 #1 SMP Thu Jan 26 22:04:50 EET 2023 armv7l GNU/Linux
```

Παρατηρούμε ότι η έκδοση του λειτουργικού έχει αλλάξει από 3.2.0-4 σε 3.16.84.

2.

Για να προσθέσουμε το Custom System Call στο πυρήνα ακολουθήσαμε το παρακάω tutorial:

http://www.newfreesoft.com/linux/arm_linux_system_call_6292/

Πρακτικά φτιάξαμε ένα νέο φάκελο με το όνομα greetings στο root directory του linux.

Δημιουργήσαμε ένα αρχείο greetings.c που πρακτικά είναι ο πηγαίος κώδικας του system all:

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

asmlinkage long sys_greetings(void)
{
        printk("Greeting from kernel and team no %d\n", 9);
        return 0;
}
```

Στη συνέχεια φτιάξαμε το Makefile που χρειάζεται.

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

Μετά προσθέσαμε στο Makefile του root directory του linux το φάκελο greetings/ στο παρακάτω σημείο (προκειμένου να γίνει make):

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

Μετά προσθέσαμε στο **linux-source-3.16/arch/arm/kernel/calls.S** (system call table για arm, άμα θέλαμε για x86 αρχιτεκτονική θα προσθέταμε στο system_{32, 64}.tbl) το custom system call και τον αριθμό που του αντιστοιχεί:

```
/* 385 */       CALL(sys_memfd_create)
                CALL(sys_greetings)
```

Δηλαδή στο custom system call ανατέθηκε η τιμή 386.

Τέλος στο αρχείο **linux-source-3.16/arch/arm/include/asm/unistd.h**:

```
#define __NR_sys_greetings (__NR_SYSCALL_BASE+386)
```

Κατά αυτό το τρόπο προσθέσαμε το custom system call στο πυρήνα των linux. Στη συνέχεια κάναμε make τον πυρήνα και ακολουθήσαμε την ίδια διαδικασία με το ερώτημα 1 για να περάσουμε τις αλλαγές στο Qemu.

3.

Ένα πρόγραμμα που κάνει κλήση του παραπάνω system call είναι:

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

int main(){

    printf("System calling.\n");
    long int ret = syscall(386);
    printf("System call sys_greetings returned %ld\n", ret);

    return 0;
}
```

Το output του παραπάνω προγράμματος όταν εκτελέστει μέσα στο Qemu είναι:

```
root@debian-armhf:~# ./test_sys_call
System calling.
System call sys_greetings returned 0
root@debian-armhf:~# dmesg | tail 1
tail: cannot open `1' for reading: No such file or directory
root@debian-armhf:~# ./test_sys_call
System calling.
System call sys_greetings returned 0
root@debian-armhf:~# dmesg | tail
[  104.287579] RPC: Registered named UNIX socket transport module.
[  104.295257] RPC: Registered udp transport module.
[  104.299339] RPC: Registered tcp transport module.
[  104.304000] RPC: Registered tcp NFSv4.1 backchannel transport module.
[  104.589068] FS-Cache: Loaded
[  104.719517] FS-Cache: Netfs 'nfs' registered for caching
[  104.919944] Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
[  327.006886] Greeting from kernel and team no 9
```

Παρατηρούμε στο τέλος ότι το στο log του kernel έχει εκτυπωθεί το ζητούμενο μήνυμα.

Όλα τα παραπάνω αρχεία έχουν επισυναφθεί στο zip.