ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΙΣΤΩΝ

ΣΧΕΔΙΑΣΜΟΣ ΕΝΣΩΜΑΤΩΜΕΝΩΝ ΣΥΣΤΗΜΑΤΩΝ – 7ο ΕΞΑΜΗΝΟ

Μαρκέτος Νικόδημος – ΑΜ:03117095

Τζομάκα Αφροδίτη – ΑΜ: 03117107

5η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Άσκηση 1

1. Η armel (ARM EABI) port στοχεύει σε ένα σύνολο παλαιότερων 32bit ARM συσκευών, ενώ η νεότερη armhf (ARM hard-float) υποστηρίζει νεότερες και ισχυρότερες 32bit συσκευές.

2. Η επιλογή της αρχιτεκτονικής που θα χρησιμοποιηθεί συνδέεται άμεσα με το target σύστημα του cross compiler. Εν προκειμένω 'στοχεύουμε' μέσω του QEMU σε σύστημα ARM αρχιτεκτονικής που τρέχει επεξεργαστή cortex a9 με επέκταση neon για SIMD εντολές. Ο πυρήνας του host μηχανήματός μας είναι Linux(GNU EABI). Είναι λοιπόν προφανής ο λόγος επιλογής της αρχιτεκτονικής 'arm-cortexa9_neon-linux-gnueabihf'.

Εάν επιλέγαμε κάποια άλλη αρχιτεκτονική από το list-samples και τρέχαμε το cross compiled εκτελέσιμο στο QEMU οι πιθανές περιπτώσεις είναι οι εξής:

- Το εκτελέσιμο δεν τρέχει καθόλου επιστρέφοντας exec format error.

- Παράγεται πιο αργός κώδικας

- Το πρόγραμμα τρέχει με λειτουργία διαφορετική από την προβλεπόμενη.

3. Χρησιμοποιήθηκε η default επιλογή της συγκεκριμένης αρχιτεκτονικής, η glibc 2.25. Εκτελώντας την εντολή ldd -v ~/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin/arm-cortexa9_neon-linux-gnueabihf-gcc βλέπουμε ότι όντως χρησιμοποιήθηκε glibc.

4. Εκτελώντας την εντολή /home/aphrodite/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin/arm-cortexa9_neon-linux-gnueabihf-gcc phods2.c -O0 -Wall -o phods_crosstool.out κάνουμε compile τον ζητούμενο κώδικα και τρέχοντας:

- την εντολή file παίρνουμε:

```
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.
```

- την εντολή readelf -h -A παίρνουμε:

```
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:          16724 (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 ***file*** χρησιμοποιείται για να κατηγοριοποιήσει το αρχείο με το οποίο την καλούμε τρέχοντας κάποια αντίστοιχα tests (filesystem, magic, language). Εν προκειμένω αναγνωρίζεται ότι το .out αρχείο είναι εκτελέσιμο των 32 bit (Little Endian), η αρχιτεκτονική του, η EABI version του, ο τύπος του linking, το interpreter path, η kernel version και το αν είναι stripped ή όχι.

Η εντολή ***readelf*** παρέχει περισσότερες πληροφορίες για το εκτελέσιμο. Οι παράμετροι -h και -A καθορίζουν την άντληση πληροφοριών από τα headers του elf αρχείου και συγκεκριμένα εκείνες που αφορούν την αρχιτεκτονική του (αν υπάρχουν) αντίστοιχα.

5. Ο κώδικας έγινε compile με την εντολή:

/home/aphrodite/x-tools/arm-cortexa9_neon-linux-gnueabihf/bin/arm-cortexa9_neon-linux-gnueabihf-gcc -O0 -Wall -o phods_linaro.out phods2.c

Προκειμένου να τρέξει ο compiler της linaro αναγκαστήκαμε να εγκαταστήσουμε 32-bit – packages (libc6-i386, lib32stdc++6, lib32gcc1, lib32ncurses5, lib32z1) επομένως σε σύγκριση με τις 64-bit εντολές του cross compiler που χτίσαμε εμείς είναι λογικό να παράγει μικρότερου μεγέθους εκτελέσιμα.

phods_linaro.out – 8.2KB / phods_crosstool.out – 18.2KB

6. Κάνοντας χρήση της εντολής ldd για τον compiler του linaro βλέπουμε ότι χρησιμοποιεί κ αυτός τις libc βιβλιοθήκες, των 32 όμως bit. Εν τέλει, όπως είναι γνωστό, ένα 64 bit σύστημα δύναται να λειτουργήσει και σε 32-bit mode παράγοντας σωστά αποτελέσματα.

7. Αρχικά παρατηρούμε την αύξηση του μεγέθους και των δύο αρχείων σε σχέση με τα προηγούμενα dynamically linked εκτελέσιμα, κάτι το οποίο είναι αναμενόμενο εφόσον περιέχουν και τον κώδικα των αντίστοιχων συνδεδεμένων βιβλιοθηκών. Ωστόσο παρατηρούμε ότι και η μεταξύ τους διαφορά έχει μεγαλώσει. Αυτό οφείλεται αφένος στον γεγονός ότι οι βιβλιοθήκες που ενσωματώνει το linaro είναι των 32 bit ενώ του cross compiler που χτίσαμε εμείς είναι 64 bit.

phods_crosstool_static.out – 2.9MB / phods_linaro_static.out – 507.9KB.

8. -Α. Προφανώς δεν θα τρέξει στο host μηχάνημα αφού είναι cross compiled για arm αρχιτεκτονική.

-Β. Σε αυτήν την περίπτωση, στο target σύστημα, θα τρέξει μόνο αν μεταφέρουμε και την ανανεομένη glibc.

-Γ. Θα τρέξει κανονικά εφόσον οι συναρτήσεις της νέας βιβλιοθήκης θα έχουν ενσωματωθεί στον κώδικα κατά το linking.

Άσκηση 2

1. Εκτελώντας uname -a πριν και μετά την εγκατάσταση του νέου πυρήνα παίρνουμε τα εξής:

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 Thu Jan 7 19:57:52 EET 2021 armv7l GNU/Linux***

Παρατηρούμε ότι πλέον εμφανίζεται η νέα έκδοση του πυρήνα που εγκαταστήσαμε εμείς ενώ άλλη μια διαφορά είναι η εμφάνιση της ώρας και την ημερομηνίας που έγινε compile ο πυρήνας.

2. Αρχικά δημιουργούμε ένα directory με όνομα oursys στο ***~/linux-source-code*** και προσθέτουμε σε αυτό το αρχείο κώδικα oursys.c:

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

asmlinkage long sys_greeting(void)

{

      printk("Greeting from kernel and team no7\n");

      return 0;

}
```

και ένα αντίστοιχο Makefile το οποίο περιέχει μόνο την γραμμή:

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

Το παραπάνω λέει στο kbuild (μέρος των Kernel Makefiles) ότι στο συγκεκριμένο directory υπάρχει ένα αντικείμενο greetings.o το οποίο έχει γίνει build απο το greetings.c.

Στην συνέχεια μεταβαίνουμε στο parent directory του πυρήνα και στο αντίστοιχο Makefile όπου προσθέτουμε στην γραμμή:

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

το directory oursys:

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

προκειμένου να πούμε στον compiler ότι σε αυτό βρίσκεται το system call μας.

Στο ***~/linux-source-code/include/linux/syscalls.h*** προσθέτουμε την γραμμή:

```
asmlinkage long sys_greeting(void);
```

Το παραπάνω καθορίζει ένα prototype της συνάρτησης του system call μας. Το keyword 'asmlinkage' καταδεικνύει πως όλες οι παράμετροι της συνάρτησης είναι διαθέσιμες στην στοίβα.

Στο ***~/linux-source-code/arch/arm/kernel/call.S*** προσθέτουμε την γραμμή:

```
/*386*/ CALL(sys_greeting)
```

Στο ***~/linux-source-code/arch/arm/include/uapi/asm/unistd.h*** προσθέτουμε την γραμμή:

```
#define __NR_sys_greeting      (__NR_SYSCALL_BASE+388)
```

προκειμένου να αντιστοιχίσει ο πυρήνας την ρουτίνα εξυπηρέτησης του system call με το αντίστοιχο system call number. (Παρόλο που αντιστοιχίσαμε το system call μας με τον αριθμό 386 στο offset του __NR_SYSCALL_BASE βάζουμε το 388 καθώς το προηγούμενο syscall, memfd_create( ), διέθετε padding +2).

3. Προκειμένου να ελέγξουμε την ορθή λειτουργία του syscall μας φτιάχνουμε ένα πολύ απλό πρόγραμμα:

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

int main()
{
      long ret = syscall(386);
      printf("Return num of sys_greeting = : %ld\n",ret);
      return 0;
}
```

Εκτελέσαμε το πρόγραμμα και ύστερα τρέξαμε dmesg. Τα αποτελέσματα φαίνονται παρακάτω και καταδεικνύουν ότι όλα λειτουργούν όπως θα έπρεπε.

```
root@debian-armhf:~# ./t
[  130.851724] Greeting from kernel and team no7
Return num of sys_greeting = 0
```

```
[   72.515996] FS-Cache: Loaded
[   72.642675] FS-Cache: Netfs 'nfs' registered for caching
[   72.849701] Installing knfsd (copyright (C) 1996 okir@monad.swb.de).
[  130.851724] Greeting from kernel and team no7
root@debian-armhf:~#
```

Γενικότερα, δεν αντιμετωπίσαμε κάποιο ιδιαίτερο πρόβλημα με την άσκηση. Τα οποιαδήποτε πακέτα έλειπαν εγκαταστάθηκαν στην πορεία ενώ ίσως αξίζει να αναφέρουμε πως λόγω παλαιότερων εκδόσεων των gcc και gdb, χρειάστηκε να γίνει αναβάθμιση του λειτουργικού του μηχανήματος μας από Ubuntu 16.04 σε 18.04 ώστε να δουλέψει το build του crosstool.