Work Log – Step1

M2M Lectures Grenoble University

Thomas Lerchundi January 24, 2017

1 Preface by Pr. Olivier Gruber

First of all, let me welcome you to this course and let me thank you for you vote of confidence, I know you have other courses that you could have attended instead. Do not hesitate to let me know what are your thoughts on this course.

This document is your work log for the first step in the M2M course, master-level, at the University of Grenoble Alpes, in France. You will have such a document for each step of our course together.

This document has two parts. One part is about diverse sections, each with a bunch of questions that you have to answers. The other part is really a laboratory log, keeping track of what you do, as you do it.

The questions provide a guideline for your learning. They are not about getting a good grade if you answer them correctly, they are about giving your pointers on what to learn about.

The goal of the questions is therefore not to be answered in three lines of text and be forgotten about. The questions must be researched and thoroughly understood. Ask questions around you if things are unclear, to your fellow students and to me, your professor.

Writing down the answers to the questions is a tool for helping your learn and remember. Also, it keeps track of what you know, the URLs you visited, the open questions that you are trouble with, etc. The tools you used and their corky syntax. It is intended to be a living document, written as you go.

Ultimately, the goal of the document is to be kept for your personal records. If you will ever work on embedded systems, trust me, you will be glad to have a written trace about all this.

REMEMBER: plaggia is a crime that can get you evicted forever from french universities... The solution is simple, write using your own words or quote, giving the source of the quoted text. Also, remember that you do not learn through cut&paste. You also do not learn much by watching somebody else doing.

Last but not least, help improve this course. Send me feedback[1]. Let me know what would have helped you and was missing. You can also propose something that you discovered and that was really cool. I will make my best effort to integrate your suggestions. Many students in the past have helped improve this course, so can you. Many thanks in advance.

2 Qemu

What is Qemu for?

Qemu est un hyperviseur permettant d'émuler une machine réel. Il permet à travers ses options de configurer au niveau matériel les différents composant de la machine émulée (exemple : processeur ARM x86). Cette machine peut donc faire tourner un système d'exploitation. Le système d'exploitation installé (s'il lon en met un) se comporte comme s'il se trouvait sur une machine réel.

Why cannot you run a linux kernel in a regular linux process?

2 systèmes d'exploitations se concurrencent au niveau du matériel. Il faut utiliser un hyperviseur.

Comment the different options you used to start qemu.

qemu-system-i386 -serial stdio -hda hda.img

qemu-system-i386 : c'est la commande principale d'émulation d'un CPU x86

-serial stdio : Cette option permet de rediriger les entrées sortie de la machine guest vers les entrées sortie de la machine host.

-hda hda.img : On spécifie ici quelle image disque la machine guest va utiliser en tant que « disque dur »

Boot Process

How is an x86 machine booting up? What is the role of each involved parts?

Au reveil, la machine commence par executer le code en memoire (le bios)
A l'execution du Bios :

- initialisation des périphériques
- il place en mémoire le MBR situé sur le disque dur.

Le MBR fourni une description du disque, en terme de partitions et de système de fichier.

- Le MBR va ensuite être capable de charger le kernel désiré, qui se situe sur une des partions du disque.

Ensuite, le kernel, prend la main est lance ses premiers process.

How is built the disk image that you use to boot with qemu? Describe its layout in terms of sectors and the contents of those sectors.

Il y a 3 secteurs construit avec la commande dd :
Le premier secteur correspond au master boot record. Il prend un bloc de 512 octets
Le second correspond au kernel, et viens juste après le bloc précedent (seek 1)
Le troisième secteur est placé tout à la fin, au 31ème bloc de 512 octets. On y place le dossier /dev/zero qui correspond au module root.

4 . Using Eclipse to browse the sources

Explain how you configured Eclipse to be able to browse the given sources.

Installation du plugin CDT dans eclipse.Puis importation du projet ia32 dans mon workspace.
En utilisant la fenêtre (onglet) du plugin, création de 2 targets pour le Make : *make* et *make clean* qui permettent de lancer les tâche de compilation directement depuis Eclipse.

5. Master Boot Record

From what sources (.c and .S files) is the MBR built? Hint: Look at the makefile. What is the purpose of those different files?

En suivant la liste de dépendance du Makefile, le boot loader est généré à partir d'un fichier boot.elf. Ce fichier est construit avec un fichier assembleur boot.o et loader.o. Ces 2 fichiers sont respectivement obtenus à partir d'un fichier assembleur boot.s, et d'un fichier loader.c .

boot.s correspond au code qui va être directement exécuté par le bios. C'est le code situé à l'adresse 0x7c00.

loader.c est le programme qui permet de lire les secteurs du disque.

What is an ELF? (Hint: man elf, Google is your friend)

On utilise ensuite un fichier elf pour lier les deux programmes compilés. Ce fichier est utilisé avec la commande ld qui va lier nos code compilés (point d'entrée à l'adresse 0x7c00). Elf représente l'exécutable qui lie les codes compilés de boot.o et loader.o.

Why is the objcopy program used? (Hint: look in the Makefile)

On utilise la commande objcopy pour passer d'un exécutable elf à un fichier binaire Unix (qui sera notre mbr)

What kind of information is available in an ELF file?

Chaque fichier ELF est constitué d'un en-tête fixe, puis de segments et de sections. Les segments contiennent les informations nécessaires à l'exécution du programme contenu dans le fichier, alors que les sections contiennent les informations pour la résolution des liens entre fonctions et le replacement des données.

Give the ELF layout of the MBR files (hint: readelf and objdump)

En-tête ELF:

```
  Magique:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Classe:                            ELF32
  Données:                           complément à 2, système à octets de poids faible d'abord (little endian)
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  Version ABI:                       0
  Type:                              EXEC (fichier exécutable)
  Machine:                           Intel 80386
  Version:                           0x1
  Adresse du point d'entrée:         0x7c00
  Début des en-têtes de programme :  52 (octets dans le fichier)
  Début des en-têtes de section :    5080 (octets dans le fichier)
  Fanions:                           0x0
  Taille de cet en-tête:             52 (octets)
  Taille de l'en-tête du programme:  32 (octets)
  Nombre d'en-tête du programme:     2
  Taille des en-têtes de section:    40 (octets)
  Nombre d'en-têtes de section:      14
  Table d'indexes des chaînes d'en-tête de section: 11

En-têtes de section :
  [Nr] Nom              Type            Adr      Décala.Taille ES Fan LN Inf Al
  [ 0]                  NULL            00000000 000000 000000 00      0   0  0
  [ 1] .text            PROGBITS        00007c00 000074 0001cb 00 WAX  0   0  4
  [ 2] .comment         PROGBITS        00000000 00023f 00002b 01  MS  0   0  1
  [ 3] .debug_aranges   PROGBITS        00000000 000270 000040 00      0   0  8
  [ 4] .debug_info      PROGBITS        00000000 0002b0 00063a 00      0   0  1
  [ 5] .debug_abbrev    PROGBITS        00000000 0008ea 000233 00      0   0  1
  [ 6] .debug_line      PROGBITS        00000000 000b1d 00015b 00      0   0  1
  [ 7] .debug_frame     PROGBITS        00000000 000c78 000088 00      0   0  4
  [ 8] .debug_str       PROGBITS        00000000 000d00 0002a0 01  MS  0   0  1
  [ 9] .debug_loc       PROGBITS        00000000 000fa0 000308 00      0   0  1
```

```
  [10] .debug_ranges     PROGBITS        00000000 0012a8 0000a0 00      0   0  1
  [11] .shstrtab         STRTAB          00000000 001348 000090 00      0   0  1
  [12] .symtab           SYMTAB          00000000 001608 000210 10     13  27  4
  [13] .strtab           STRTAB          00000000 001818 0000c1 00      0   0  1
Clé des fanions :
  W (écriture), A (allocation), X (exécution), M (fusion), S (chaînes)
  I (info), L (ordre des liens), G (groupe), T (TLS), E (exclu), x (inconnu)
  O (traitement additionnel requis pour l'OS) o (spécifique à l'OS), p (spécifique au processeur)

Il n'y a pas de groupe de section dans ce fichier.

En-têtes de programme :
  Type           Décalage Adr. vir.  Adr.phys.  T.Fich. T.Mém.  Fan Alignement
  LOAD           0x000074 0x00007c00 0x00007c00 0x001cb 0x001cb RWE 0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x10

 Correspondance section/segment :
  Sections de segment...
   00     .text
   01

Il n'y a pas de section dynamique dans ce fichier.

Il n'y a pas de réadressage dans ce fichier.

The decoding of unwind sections for machine type Intel 80386 is not currently supported.

Table de symboles « .symtab » contient 33 entrées :
   Num:    Valeur Tail Type    Lien   Vis      Ndx Nom
     0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00007c00     0 SECTION LOCAL  DEFAULT    1
     2: 00000000     0 SECTION LOCAL  DEFAULT    2
     3: 00000000     0 SECTION LOCAL  DEFAULT    3
     4: 00000000     0 SECTION LOCAL  DEFAULT    4
     5: 00000000     0 SECTION LOCAL  DEFAULT    5
     6: 00000000     0 SECTION LOCAL  DEFAULT    6
     7: 00000000     0 SECTION LOCAL  DEFAULT    7
     8: 00000000     0 SECTION LOCAL  DEFAULT    8
     9: 00000000     0 SECTION LOCAL  DEFAULT    9
    10: 00000000     0 SECTION LOCAL  DEFAULT   10
    11: 00000000     0 FILE    LOCAL  DEFAULT  ABS boot.o
    12: 00000008     0 NOTYPE  LOCAL  DEFAULT  ABS PROT_MODE_CSEG
    13: 00000010     0 NOTYPE  LOCAL  DEFAULT  ABS PROT_MODE_DSEG
    14: 00000001     0 NOTYPE  LOCAL  DEFAULT  ABS CR0_PE_ON
    15: 00007c0d     0 NOTYPE  LOCAL  DEFAULT    1 seta20.1
    16: 00007c17     0 NOTYPE  LOCAL  DEFAULT    1 seta20.2
    17: 00007c3d     0 NOTYPE  LOCAL  DEFAULT    1 nobiosmem
    18: 00007c41     0 NOTYPE  LOCAL  DEFAULT    1 memdetected
    19: 00007c41     0 NOTYPE  LOCAL  DEFAULT    1 real_to_prot
    20: 00007c98     0 NOTYPE  LOCAL  DEFAULT    1 gdtdesc
    21: 00007c56     0 NOTYPE  LOCAL  DEFAULT    1 protcseg
    22: 00007c7d     0 NOTYPE  LOCAL  DEFAULT    1 spin
    23: 00007c80     0 NOTYPE  LOCAL  DEFAULT    1 gdt
    24: 00000000     0 FILE    LOCAL  DEFAULT  ABS loader.c
    25: 00007c9e    92 FUNC    LOCAL  DEFAULT    1 readsect
    26: 00000000     0 FILE    LOCAL  DEFAULT  ABS
```

```
27: 00007db5    22 FUNC    GLOBAL DEFAULT    1 diskboot
28: 00007cfa   187 FUNC    GLOBAL DEFAULT    1 load_elf
29: 00007dcb     0 NOTYPE  GLOBAL DEFAULT    1 __bss_start
30: 00007dcb     0 NOTYPE  GLOBAL DEFAULT    1 _edata
31: 00007dcc     0 NOTYPE  GLOBAL DEFAULT    1 _end
32: 00007c00     0 NOTYPE  GLOBAL DEFAULT    1 start
```

Look at the code in loader.c and understand it. What are the function waitdisk, readsect, and readseg doing?

waitdisk permet d'attendre que le disque soit disponible avant une lecture.
readsect lis un secteur.
readseg permet de lire l'ensemble des secteur d'un segment.

---

9. Explain the dialog with the disk controller. (Hint: in/out functions).

On vérifie la disponibilité du diskcontroller à l'adresse 0x1F7.
La vérification se fait en lisant le contenu du premier bit de la valeur obtenue avec inb(0x1F7) & 0xC0) != 0x40
cette partie nous indique si le premier bit du diskcontroller est à 0 (c'est à dire s'il est non utilisé)

---

10. What can you say about the concepts at the software-hardware frontier?

L'utilisation des fonctions in* et out* dépende énormément du hardware, puisque le numéro de port fait partie de leur paramètre. Un changement de hardware impliquerait une incompatibilité du software si les ports sont différents.

6 Master Boot Record Debugging

Use gdb to step through our bootloader. Single stepping through a program is often the best way to understand how it works. It is a skill that you must have. You have a README-GDB with everything you need to know about GDB. You also have the Beej's quick tutorial to GDB.

Hints:

- Look at the dbg target in the Makefile.
- Look at the .gdbinit file.
- Use source layout in gdb.
- Use emacs as a front end.

List and explain the various gdb commands you use.

La principale commande que j'ai utilisé pour débugger est celle-ci :

gdb -tui -iex "set auto-load safe-path /home/thomas/M2PGi-M2M/M2PGi-M2M/Step1/bootloader/ia32" kernel.elf

Elle me permet d'ouvrir gdb avec un chargement automatique du .gdbinit dans mon répertoire. Et de précharger la table des symboles souhaitée.

Contenu du .gdbinit :

```
set non-stop off
symbol kernel.elf
br kmain
br write_string
```

J'ai donc placé 2 breakpoints sur les principales fonctions du programme.

7 Our mini Kernel

Explain in your own words what is the mini kernel.

---

1. What is the code in crt0.S doing?

Ce fichier assembleur permet d'initialiser le programme en initialisant ce qui est nécessaire avant l'appel de la fonction kmain. Il initialise la pile et nettoie les flags.

---

2. What are the function in/out for at this level?

Les in et out sont utilisées pour faire des lectures et ecritures sur les ports des terminaux (Qemu ou Terminal d'exécution du programme)

---

3. What are the inline attributes for?

L'appel au fonction par gcc se fait plus rapidement.

---

4. Explain why is your fan ramping up when you launch qemu with: $ make clean ; make run

5. Explain what is the relationship between the qemu option (-serial stdio) and the COM1 concept in the program.

Cette option permet de « brancher » les entrées de qemu sur l'entrée de COM1 (ce qu'il émule). Il en va de même pour les sorties.

6. Explain what is COM1 versus the console?

Il s'agit de la sortie du logiciel émulé par Qemu.

Kernel Extensions

IT IS MANDATORY TO USE THE DEBUGGER TO DEBUG YOUR CODE. You will be asked to demonstrate that skill.

9.1 Echo on the screen

This extension is to have the input from the UART be echoed on the console screen (the greenish output). Do not forget that you have only 25 lines and you will need to implement scrolling.

Cette partie a été implémentée, echo + scrolling des lignes. De plus, j'ai la possibilité de déplacer un curseur et d'écrire au milieu d'une ligne en cours de rédaction. Il est aussi possible de supprimer les caractères (touche DEL).

Je n'ai pas eu le temps d'implémenter l'historique des commandes.

Pas implémentés

9.2 History and line editing

This extension is to have a history of typed lines. A line is added to the history when the return key is pressed. The arrow up and down allow you to scroll up and down in the history. The arrow left and write allow you to move left and right in the current line. The backspace and delete allow you delete characters.

9.3 Echo on COM2

This extension is to have the ability to have a printf-like capability on COM2. The code is in the kprintf.c file.
Hints:

1. Look at the target run2 in the Makefile to know how to setup COM2. 2. Add the kprintf.c file to your kernel

10 Laboratory Log

A section for your personnal notes on the experience, such as the tools you used, Linux shell commands, etc.

Most of you tend to use the shell history as a log... But a shell can crash and the history lost. The history is also limited. The history just gives you the command, not the meaning.

You will notice that you spend quite some time in this course figuring out Linux/Shell commands or tools and their corky syntax. We strongly suggests that you mention the commands/tools here with a decent explanation of how you used them.

Trust me, in 6 months, you will be happy not to go through the man/goodle waste of time...