Gabriele **Proietti Mattia** Roberto **Beraldi**

Advanced Operating Systems and Virtualization

[0] Introduction

Department of Computer, Control and Management Engineering "A. Ruberti", Sapienza University of Rome

gpm.name · proiettimattia@diag.uniroma1.it

A.Y. 2020/2021 · v1

Course Information

Teachers

- Prof. Roberto **Beraldi**
  The course holder

  Personal website: http://www.dis.uniroma1.it/~beraldi/
  Contact email: beraldi@diag.uniroma1.it

- Ing. Gabriele **Proietti Mattia**
  Contact for any information about the topics, the exam and the course in general.

  Personal website: https://gpm.name
  Contact email: proiettimattia@diag.uniroma1.it

Prerequisites

For correctly understanding the topics of the course, I will take for granted the following abilities:

- C (C89/C99) programming
- basic knowledge of the assembly language, in particular the x86 ISA
- basic knowledge of computer internal architecture (e.g. interrupts, I/O)
- basic knowledge of operating systems

Anyway, you are always invited to ask questions if something is not clear

Materials and Exams

For attending the course you need:

- to join the Google Classroom of the course with your student account, using the invite code: **`saeuagm`**
- to create a GitHub account, if you did not have one already (use/add your Sapienza G Suite email). We will use GitHub classroom for the project, more details will be given
- keep always an eye to the course's website: gpm.name/teaching/2021-aosv

Lecture Schedule

The course starts today **24th Feb, 2021** and will end on **28th May, 2021**. During the week we will have two lectures:

- Wednesday 17.00 - 19.00 (2h)
- Friday 08.00 - 11:00 (3h)

The Friday's lecture lasts 3 hours, therefore in the last hour we will usually see some practical examples that you can follow with your PC. I suggest you to prepare a Linux environment (in a VM if Linux is your daily driver). I will use Ubuntu as reference.

Lecture will be held both in presence and via Google Meet. The course slides will be available on the course website.

Office Hours

Due to the COVID restriction office hours are no more in presence. Please send an email to proiettimattia@diag.uniroma1.it we can arrange a Google Meet.

Exam

The Exam consists of two parts:

1. **Project** (⅖ of the final mark)
2. **Written Test** (⅗ of the final mark)

The final mark will derived from the weighted average of the two parts. Particularly outstanding works can also receive *30 cum laude*.

Project

The project definition and rules will come later in the course after doing about ½ of the course outline. The purpose of the project is to dirty your hands in implementing or modifying functionalities of the Linux Kernel. For the project we will use **GitHub Classroom**.

Written Test

Will consists only of open questions about the course topics, due to the COVID things may change until June, therefore we cannot know if it will be in presence, online or mixed.

Materials

Materials for the course are essentially the slides and the notes that you take at lecture.

Topics are taken from :

- Love, Robert. *Linux kernel development*. Pearson Education, 2010.
- Bovet, Daniel P., and Marco Cesati. *Understanding the Linux Kernel: from I/O ports to process management*. " O'Reilly Media, Inc.", 2005.
- Gorman, Mel. *Understanding the Linux virtual memory manager*. Upper Saddle River: Prentice Hall, 2004.
- … and others that will be referenced in the slides

There is not a single official book.

Warning: you do not need to buy any book for the course as slides are self-contained

Acknowledgments

The content of this course and the material is freely inspired and also based on the past editions of the course, held by Prof. Alessandro Pellegrini and Prof. Francesco Quaglia. You can check their websites for further resources.

Course Outline

Purpose of the course

The purpose of the course is to gain a deep knowledge of the internal components of an operating system, which are the main architectural details in order to address the principal objectives of an OS (in particular of the kernel), like process and memory management, interrupts management, system calls, scheduling and many others. Non disregarding the actual code that implements the functionalities.

One of the most used kernel is fortunately open source, for this reason our main study objective will be the Linux Kernel, in particular in its x86 (i386) and x86_64 (amd64) declinations. But there also will be some hints to other kernels.

At the end of the course you will obviously not become kernel masters, not only because the kernel is gigantic program but also because it changes every day. Instead, you will gain a deep knowledge of the kernel internal architecture, thanks to the theory that we will do at the lectures and also of programming within the context of a specific kernel subsystem, thanks to the project.

Course Outline

Theory (Tentative)

- x86 Boot Process
- Memory Management
- System Calls
- Interrupts Management
- Concurrency and Synchronization
- Virtual Filesystem
- Process Scheduling
- Time Management
- Virtualization
- Security
- Hot Patching

Course Outline

Lab/Hands-on (Tentative)

- A glance on Git
- Building the kernel
- Debugging the kernel
- Linux Kernel Modules (LKMs)
- Using inline ASM
- Kernel Data Structures
- … more to come

A first glance on the boot process

DIAG

Boot sequence

1. **BIOS/UEFI**
   Actual hardware setup
2. **Bootloader Stage 1**
   Executes the stage 2 bootloader (skipped for UEFI)
3. **Bootloader Stage 2**
   Loads and starts the kernel
4. **Kernel**
   Takes control and initializes the machine (machine-dependent operations)
5. **Init (or systemd)**
   First process: basic environment initialization
6. **Runlevels/Targets**
   Initializes the user environment

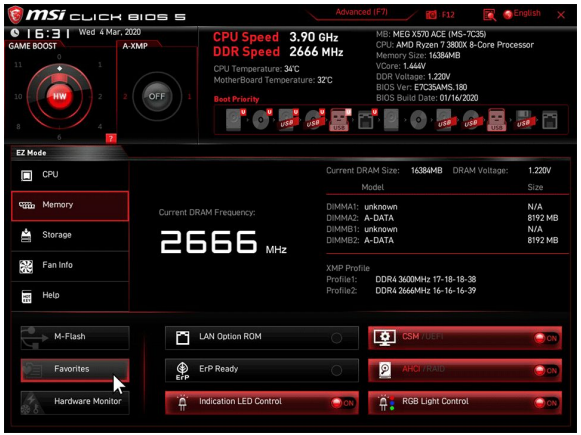
Boot Sequence

1. BIOS/UEFI

Traditionally the **BIOS** (**Basic Input/Output System**) is the first program (actually a firmware) that is executed after booting the PC. It usually resides in a chip that can be soldered or removable. The role of the BIOS is to perform hardware check and initialization for starting the operating system.

Nowadays, motherboards are no more shipped with BIOS but with **UEFI** (**Unified Extensible Firmware Interface**) that basically performs the same operations but in a more modular, extendible and efficient way. Most UEFI firmwares also have a legacy support to BIOS services.

AMIBIOS chip on a motherboard



Configuration panel in a modern UEFI motherboard

Boot Sequence

2. Bootloader Stage 1

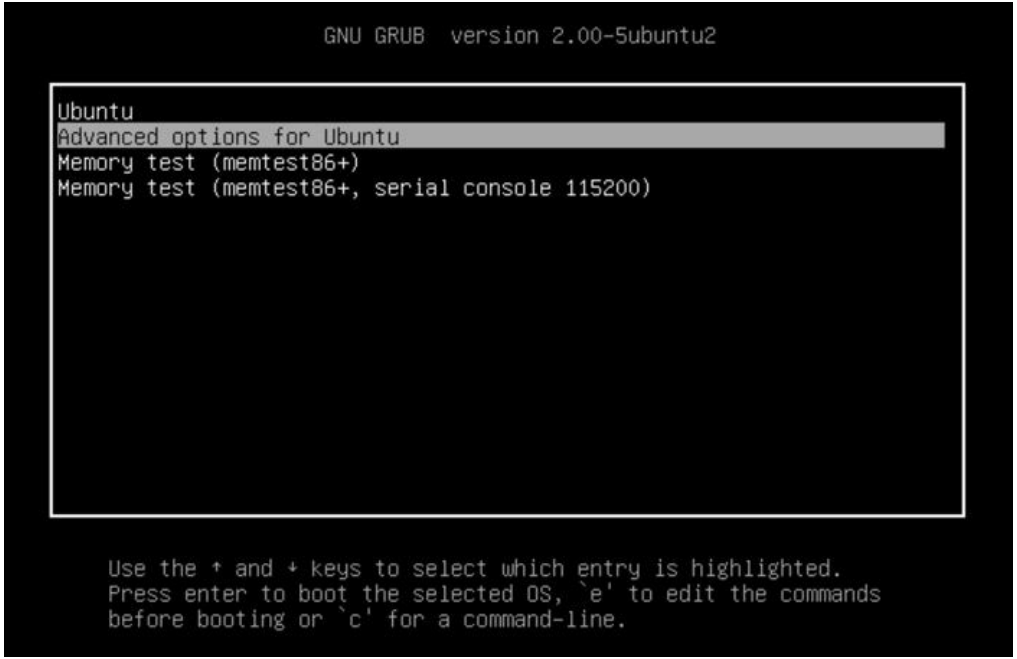
The Stage 1 bootloader is present only for BIOS and it is stored in the Master Boot Record (MBR) a very small portion of the disk (512 bytes). The role of this bootloader is only to run the Stage 2 Bootloader because it has not enough space for directly launching the kernel. A UEFI system directly launches the kernel.

The MBR is roughly divided in the following way

Boot Sequence

3. Bootloader Stage 2

This bootloader is concretized with GRUB (previously called LILO) and its role is the one of launching the kernel. Moreover it usually allows the selection of kernel image to be run.



Boot Sequence

4. Kernel

Once the kernel is launched, there are many operations to perform before passing the control to the end user. The kernel indeed:

- Configures the underlying **hardware** environment
- Mounts the root **filesystem**
- Configures the internal **data structures**
- Spawns the first process, called **`init`**

```
[    0.000000] Linux version 3.13.0-24-generic (buildd@roseapple) (gcc version 4.8.2 (Ubuntu 4.8.2-19ubuntu1) ) #46-Ubuntu SMP Th
 Apr 10 19:08:14 UTC 2014 (Ubuntu 3.13.0-24.46-generic 3.13.9)
[    0.000000] KERNEL supported cpus:
[    0.000000]   Intel GenuineIntel
[    0.000000]   AMD AuthenticAMD
[    0.000000]   NSC Geode by NSC
[    0.000000]   Cyrix CyrixInstead
[    0.000000]   Centaur CentaurHauls
[    0.000000]   Transmeta GenuineTMx86
[    0.000000]   Transmeta TransmetaCPU
[    0.000000]   UMC UMC UMC UMC
[    0.000000] e820: BIOS-provided physical RAM map:
[    0.000000] BIOS-e820: [mem 0x0000000000000000-0x000000000009fbff] usable
[    0.000000] BIOS-e820: [mem 0x000000000009fc00-0x000000000009ffff] reserved
[    0.000000] BIOS-e820: [mem 0x00000000000f0000-0x00000000000fffff] reserved
```

The first lines of `dmesg` log when the kernel starts

Boot Sequence

5. Init/systemd

The `init` process (today mostly replaced with `systemd` in many distributions) is the first process started. It starts as a daemon and it configures the **software environment**, loads the default **runlevel** (systemd) and spawns all the other processes. In Linux indeed, the only way for creating and starting a process is using the fork function, therefore every process is a fork of `init`.

Boot Sequence

6. Targets/Runlevels

The **targets** represents the state of the the machine within the context of `systemd`. Before the introduction of `systemd` (with System V) they were also called **runlevels** and they were identified by numbers. They are the following:

- Run level 0 is matched by `poweroff.target`
- Run level 1 is matched by `rescue.target`
- Run level 3 is emulated by `multi-user.target`
- Run level 5 is emulated by `graphical.target`
- Run level 6 is emulated by `reboot.target`
- Emergency is matched by `emergency.target`

Every target has associated a set of programs or services that needs to be launched.

Advanced Operating Systems and Virtualization

[0] Introduction

LECTURERS

Gabriele **Proietti Mattia**
Roberto **Beraldi**

gpm.name · proiettimattia@diag.uniroma1.it

DIAG