操作系统研讨课
Course: B0911011Y

蒋德钧

Fall Term 2019-2020

email: jiangdejun@ict.ac.cn
office phone: 62601007

中国科学院大学
University of Chinese Academy of Sciences

Lecture 0 Introduction

2019.09.02

中国科学院大学
University of Chinese Academy of Sciences

Lecture 0: Introduction

- Overview
  - Course introduction
  - Course administration

Lecture 0: Introduction

- Course objectives
  - Practice what you learn in OS theory course
  - Obtain capabilities of system programming, e.g. hardware-software co-designing
  - Establish full stack view

Lecture 0: Introduction

- Course contents
  - Build a simple operating system
    - Bootloader
    - Kernel supporting multitasking
    - Process communication and management
    - Device driver
    - Virtual memory management
    - File system

Lecture 0: Introduction

- Course administration
  - Classrooms：教 205（机房） & 221（机房）
  - Schedule (may re-schedule according to project progress)

周次	课次	时间	内容	Project
1		2019年8月26日	No class	
2	1	2019年9月2日	P1 start	bootloader
3	2	2019年9月9日	P1 design review	
4	3	2019年9月16日	P1 due, P2 start	
5	4	2019年9月23日	P2 design review	simple kernel
6	5	2019年9月30日	P2 1st due	
7		2019年10月7日	No class	
8	6	2019年10月14日	P2 2nd due, P3 start	
9	7	2019年10月21日	P3 design review	IPC
10		2019年10月28日	No class	
11	8	2019年11月4日	P3 due, P4 start	
12	9	2019年11月11日	P4 design review	virtual memory
13	10	2019年11月18日	P4 due, P5 start	
14	11	2019年11月25日	P5 design review	device driver
15	12	2019年12月2日	P5 due, P6 start	
16	13	2019年12月9日	P6 design review	file system
17	14	2019年12月16日	P6 due	
18		2019年12月23日	No class	
19	15	2019年12月30日	Final due	Final due
20		2020年1月6日	No class	

Lecture 0: Introduction

- Course administration
  - Lecturer
    - 蒋德钧：jiangdejun@ict.ac.cn
  - Teaching assistants
    - 卢天越：lutianyue@ict.ac.cn
    - 王盈： wangying01@ict.ac.cn
    - 王鹿鸣：wangluming@ict.ac.cn
    - 韩书楷：hanshukai@ict.ac.cn
    - 覃晓婉：qinxiaowan@ict.ac.cn
    - 杨依涵：yangyihan16@mails.ucas.ac.cn
  - Office hour
    - Make appointment

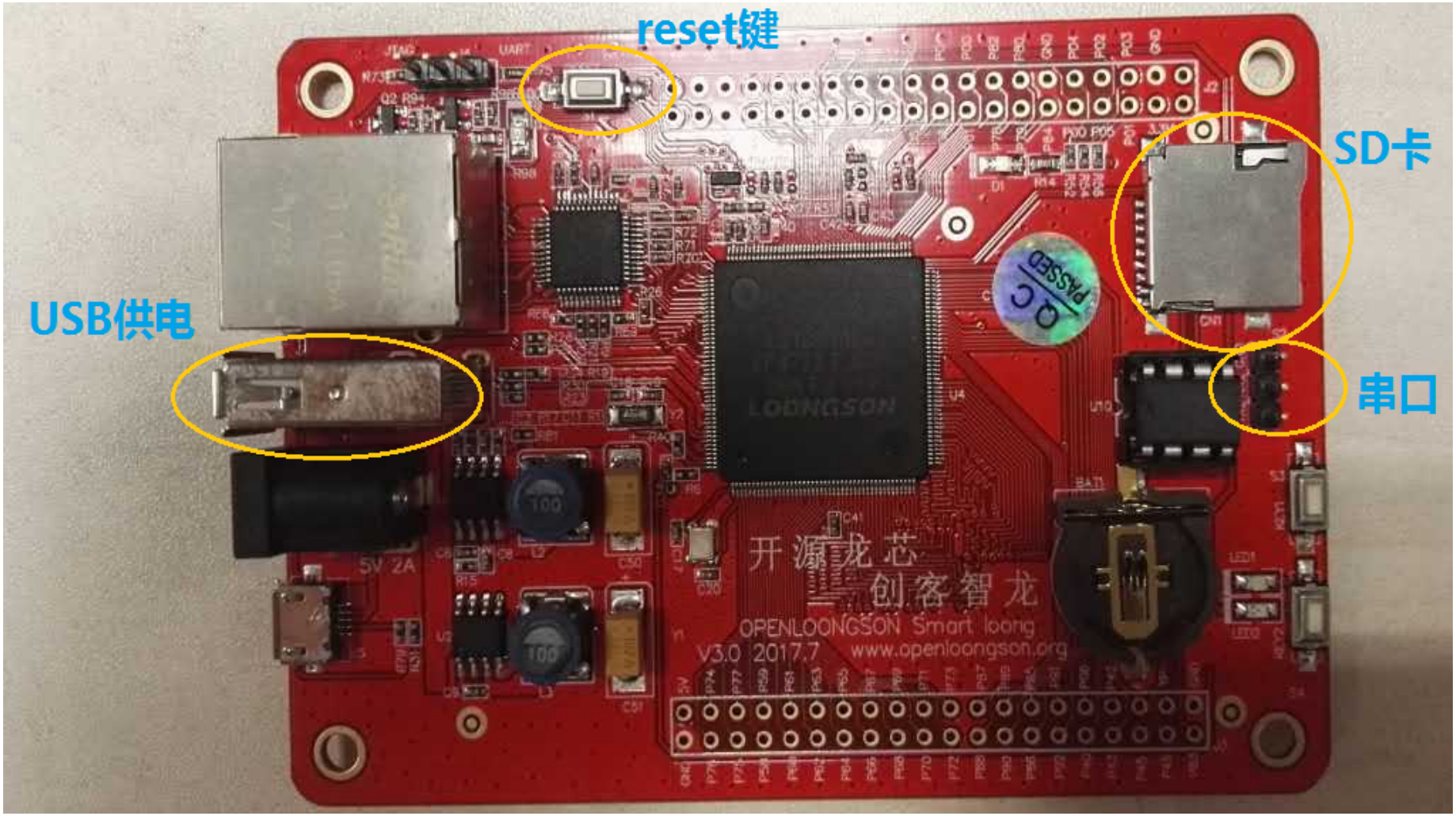
Lecture 0: Introduction

- Development environment
  - Software
    - Providing a virtual machine with VirtualBox
    - Ubuntu 12.04, kernel 3.11
    - Pre-installed MIPS compiling environment

Lecture 0: Introduction

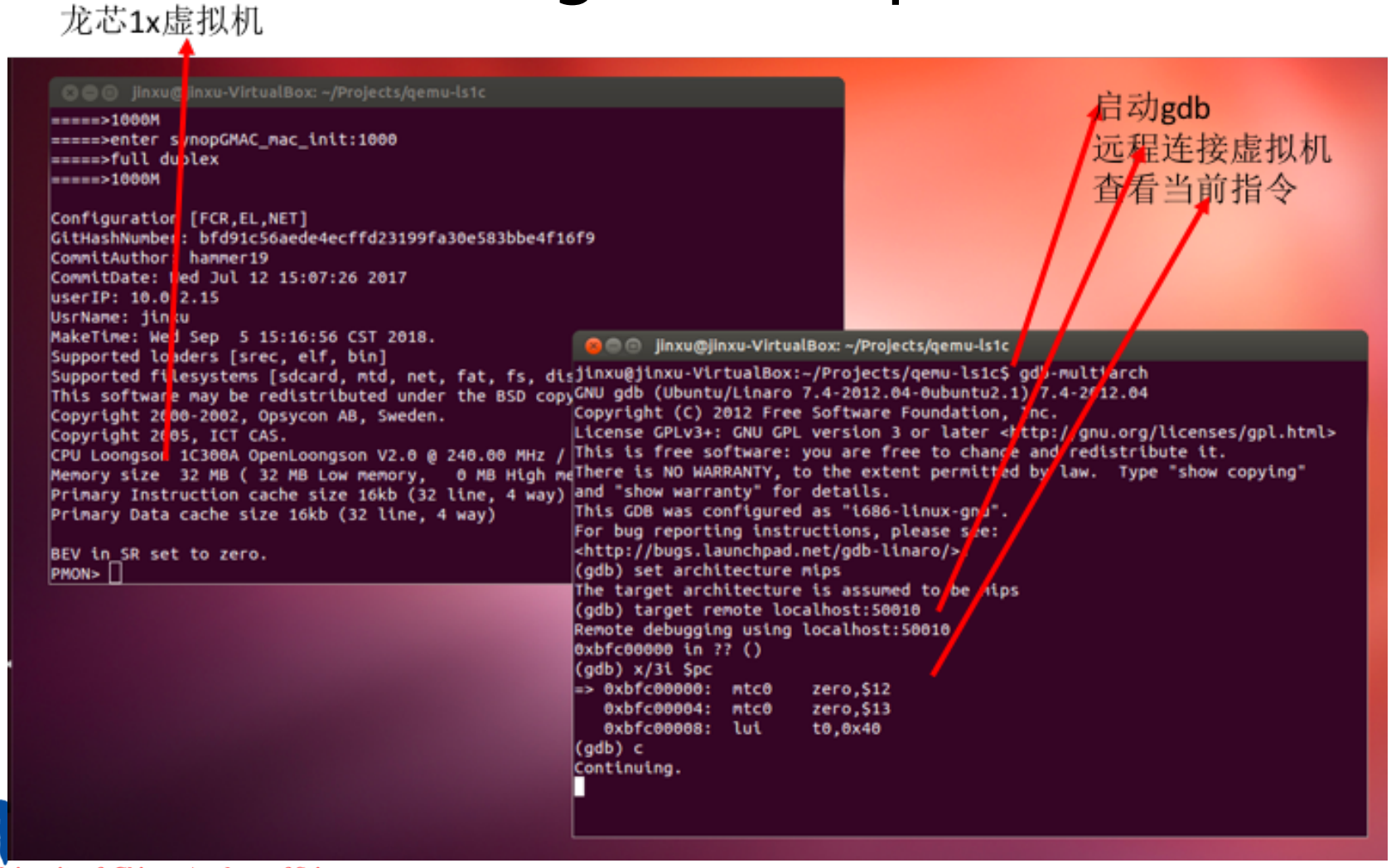
- Development environment
  - Hardware – MIPS-core board
    - One piece of Openloongson SoC board
    - One USB cable
    - One serial port cable
    - One SSD card and one card reader
    - Protection package(Optinal)

Lecture 0: Introduction



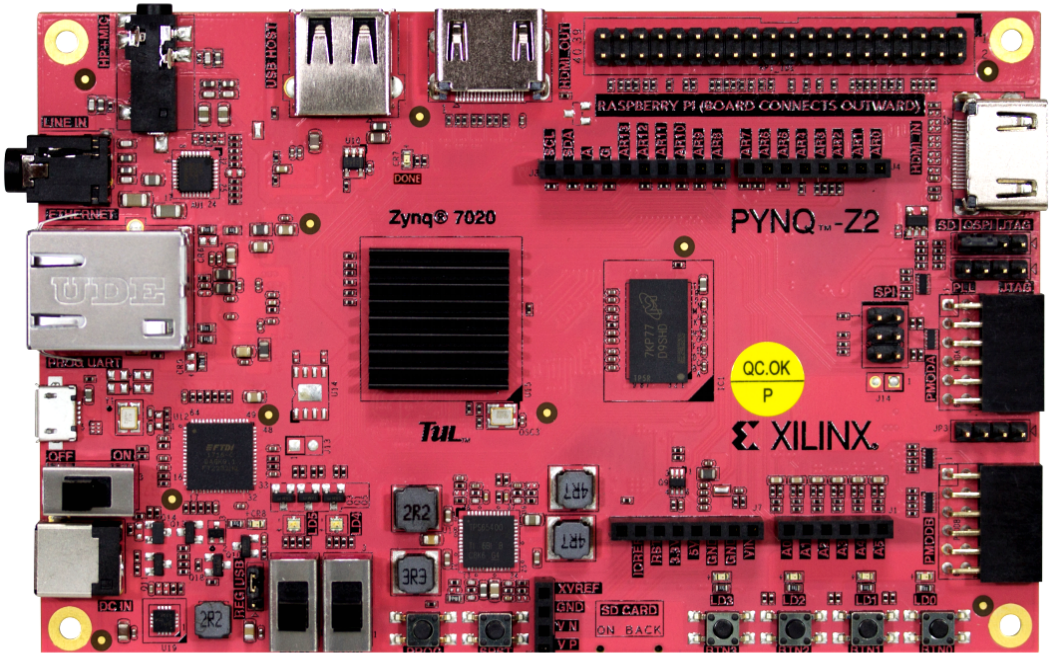
Lecture 0: Introduction

- Additional debug tools - qemu



Lecture 0: Introduction

- Development environment
  - Hardware – RISC-V core board



Lecture 0: Introduction

- Development environment
  - About RISC-V core board
    - We encourage students who want to challenge yourself to take RISC-V cord board as your testbed
    - We plan to have at most 15 students involving in OS developing on RISC-V board
      - Tell us based on your own willingness
      - A test will be given to you before you can take OS-on-RISCV experiments

Lecture 0: Introduction

- Grouping with different teachers
  - P1: group I
  - P2: grouping II
  - P3 ~ P4: grouping III
  - P5 ~ P6: grouping IV
  - Group students randomly
  - Group presentation + individual submission

Lecture 0: Introduction

- Project submission
  - Design documents
  - Source code + README
  - Submission site: course web site
    - http://sep.ucas.ac.cn/

Lecture 0: Introduction

- Grading
  - Grading per project
    - design review: 40 points
    - code development: 60 points
  - Final grading (normal case)
    - Final grades = Basic * 0.9 + Bonus * 0.1
    - Basic

P1	P2 (op1)/P2(op2)	P3	P4	P5	P6
10%	10%/10%	10%	20%	20%	20%

    - Bonus: depends on projects

Lecture 0: Introduction

- Final grading (exception case I)
  - Note that you will be able to get 60 points as the final grade once you
    - Finish P1, P2(op1), and P3
    - And finish the first task of P4, P5, and P6 respectively

Lecture 0: Introduction

- Final grading (exception case II)
  - Note that you will get 100 points as the final grade if you
    - Finish the BIG BONUS: a working operating system as a whole

Lecture 0: Introduction

- Grading
  - Grading individually depends on
    - group presentation and Q&A
    - project submission
  - Submit your project on time: 100%
  - Submit within one week after deadline: -30%
  - Submit within two weeks after deadline: -50%

Lecture 0: Introduction

- Grading
  - Copying others' code is ABSOLUTELY prohibited
    - NO points will be given

Lecture 0:

- Daily Q&A
  - WeChat

Lecture 0: Introduction

- Suggestions on accomplish this course
  - Think before coding
  - Start as possible as you can, DON'T be deadline driven
  - Prepare the first version of your design before design review, and discuss with teachers
  - Learn to use tools, e.g. gdb
  - Group working
  - XV6 may help

Lecture 0: Introduction

- Any question?

Lecture 1 Bootloader

2018.09.12

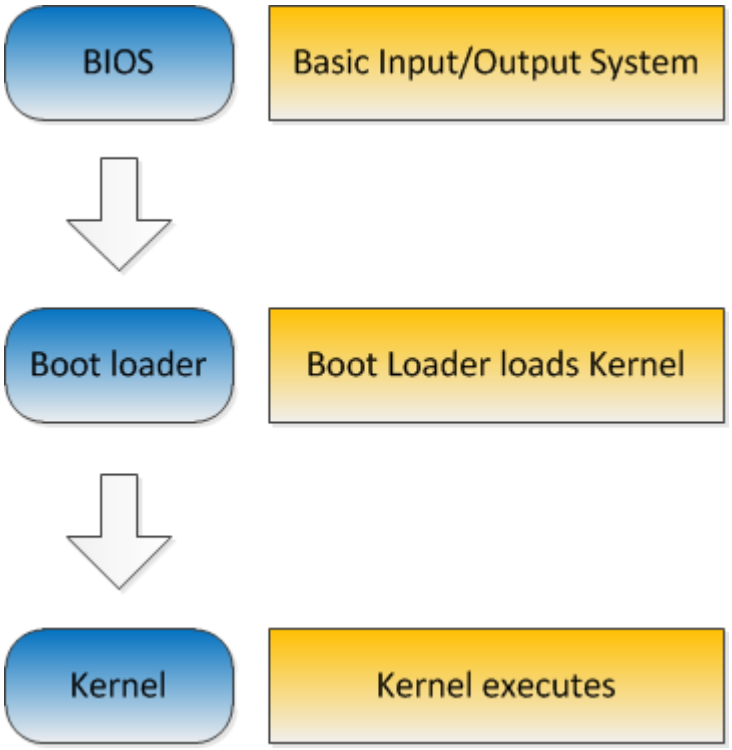
中国科学院大学
University of Chinese Academy of Sciences

Project 1 – Bootloader

- Requirements
  - Write a bootloader to start a tiny kernel based on Openloongson SoC board
    - bootblock.s
    - kernel.c
    - createimage.c

Project 1 – Bootloader

- Booting procedure



Project 1 – Bootloader

- BIOS
  - Basic Input/Output System
  - Firmware used to perform hardware initialization after power-on
  - Load bootloader
- Bootblock
  - Loaded by BIOS
  - Located in the first sector on hard disk

Project 1 – Bootloader

- Bootloader
  - A small program to enable operating system
    - Load the kernel
    - Switch control to the kernel

Project 1 – Bootloader

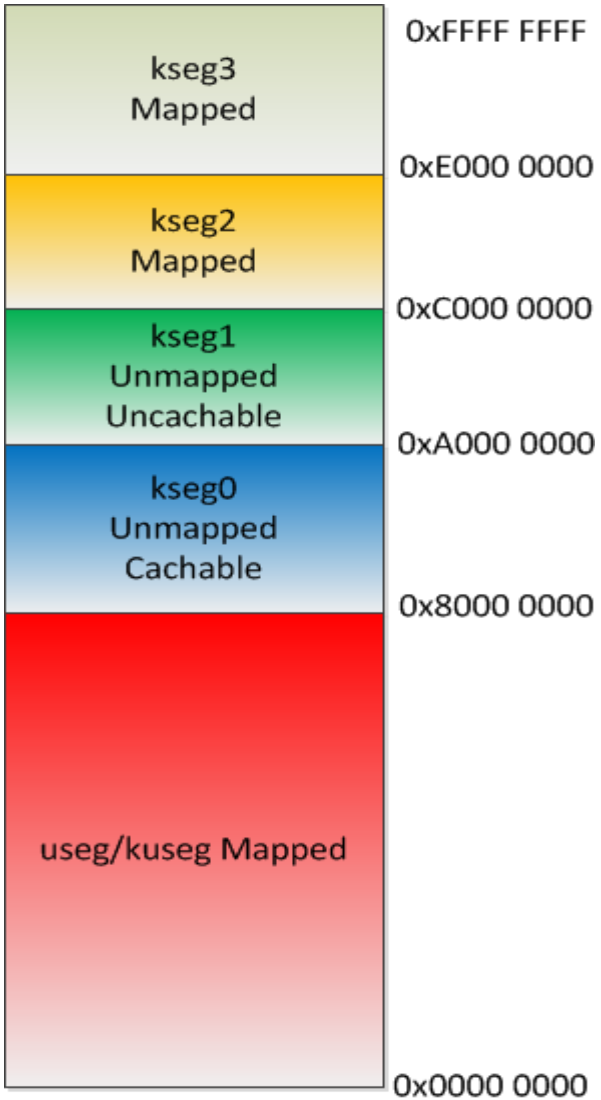
- Tiny kernel
  - A tiny piece of code
    - Use __attribute__ to control the section of ***_start*** function in kernel
    - Add code in your kernel

kernel.c

```
void __attribute__((section(".entry_function"))) _start(void)
{
    // Call PMON BIOS printstr to print message "Hello OS!"
    return;
}
```

Project 1 – Bootloader

- Memory mapping
  - Bootloader will be placed at 0xa080 0000 by BIOS
  - Place you kernel at 0xa080 0200 use your own bootloader



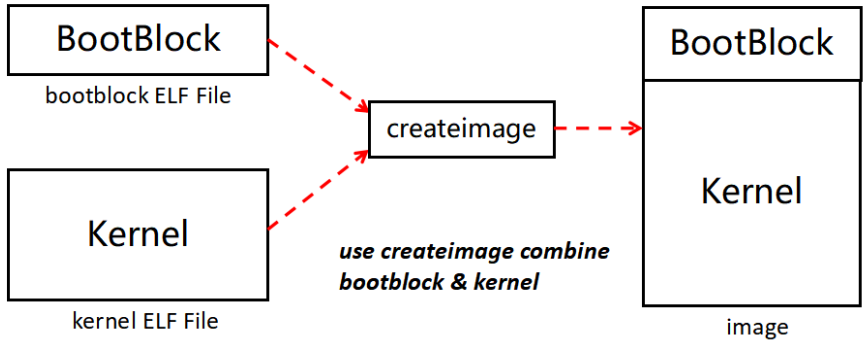
Project 1 – Bootloader

- Memory mapping
  - Makefile in start-code decides the location of you kernel in memory

```
kernel: kernel.c
    ${CC} -G 0 -O2  -fno-pic -mno-abicalls -fno-builtin -nostdinc -mips3 -Ttext=0xffffffffa0800200
```

Project 1 – Bootloader

- Createimage
  - Executable file
    - bootloader
    - kernel
  - Bootable OS image
    - Use createimage tool to generate the image
    - Parse ELF files and combine their segments

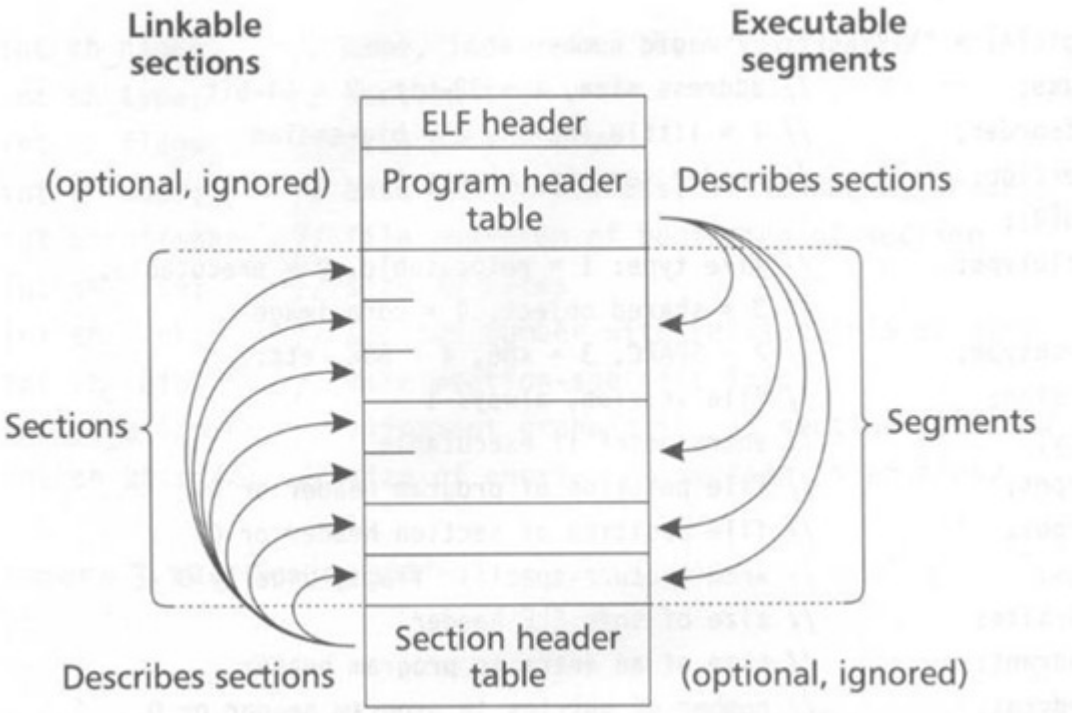


Project 1 – Bootloader

- ELF object file format
  - Executable and Linking Format (ELF)
  - Object file
    - Binary representation of programs
  - Created by assembler and link editor

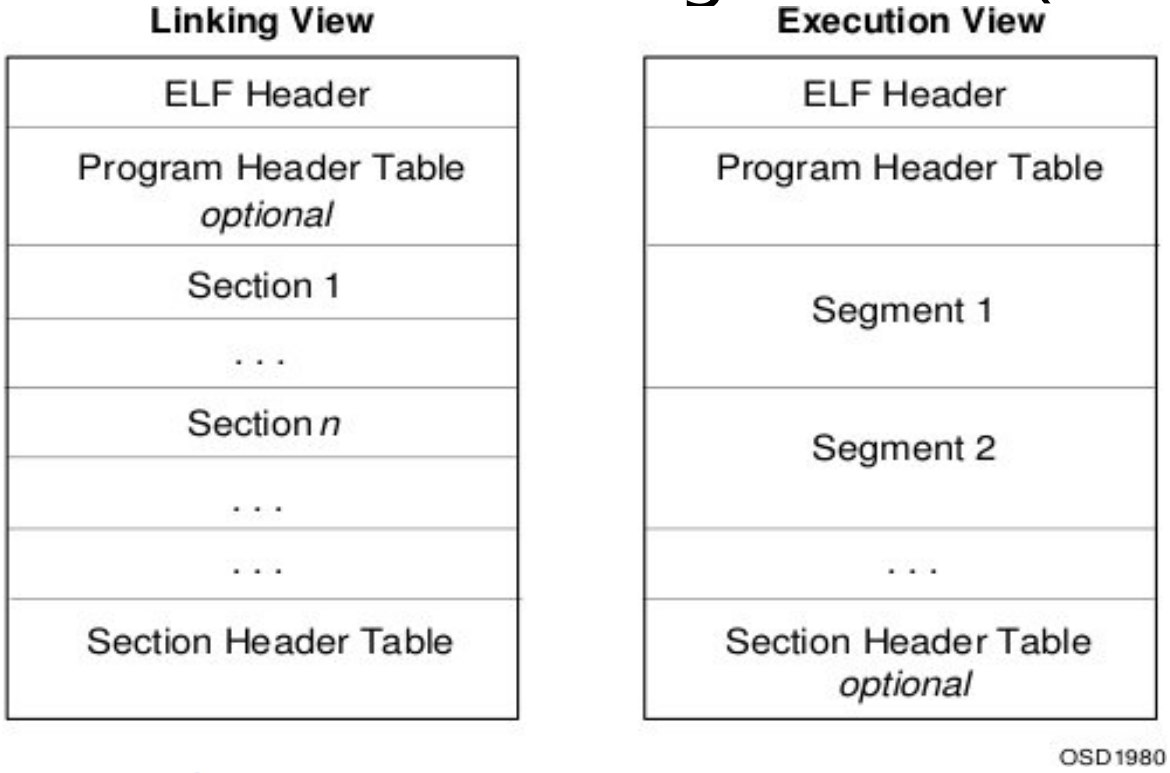
Project 1 – Bootloader

- ELF object file format
  - Executable and Linking Format (ELF)



Project 1 – Bootloader

- ELF object file format
  - Executable and Linking Format (ELF)



Project 1 – Bootloader

- ELF object file format
  - ELF header

```
typedef struct
{
  unsigned char e_ident[EI_NIDENT];   /* Magic number and other info */
  Elf32_Half    e_type;               /* Object file type */
  Elf32_Half    e_machine;            /* Architecture */
  Elf32_Word    e_version;            /* Object file version */
  Elf32_Addr    e_entry;              /* Entry point virtual address */
  Elf32_Off     e_phoff;              /* Program header table file offset */
  Elf32_Off     e_shoff;              /* Section header table file offset */
  Elf32_Word    e_flags;              /* Processor-specific flags */
  Elf32_Half    e_ehsize;             /* ELF header size in bytes */
  Elf32_Half    e_phentsize;          /* Program header table entry size */
  Elf32_Half    e_phnum;              /* Program header table entry count */
  Elf32_Half    e_shentsize;          /* Section header table entry size */
  Elf32_Half    e_shnum;              /* Section header table entry count */
  Elf32_Half    e_shstrndx;           /* Section header string table index */
} Elf32_Ehdr;
```

Project 1 – Bootloader

- ELF object file format
  - Section header

```
typedef struct
{
  elf32_word    sh_name;        /* Section name (string tbl index) */
  elf32_word    sh_type;        /* Section type */
  elf32_word    sh_flags;       /* Section flags */
  elf32_addr    sh_addr;        /* Section virtual addr at execution */
  elf32_off     sh_offset;      /* Section file offset */
  elf32_word    sh_size;        /* Section size in bytes */
  elf32_word    sh_link;        /* Link to another section */
  elf32_word    sh_info;        /* Additional section information */
  elf32_word    sh_addralign;   /* Section alignment */
  elf32_word    sh_entsize;     /* Entry size if section holds table */
} elf32_shdr;
```

Project 1 – Bootloader

- ELF object file format
  - Program header

```
typedef struct
{
  Elf32_Word    p_type;         /* Segment type */
  Elf32_Off     p_offset;       /* Segment file offset */
  Elf32_Addr    p_vaddr;        /* Segment virtual address */
  Elf32_Addr    p_paddr;        /* Segment physical address */
  Elf32_Word    p_filesz;       /* Segment size in file */
  Elf32_Word    p_memsz;        /* Segment size in memory */
  Elf32_Word    p_flags;        /* Segment flags */
  Elf32_Word    p_align;        /* Segment alignment */
} Elf32_Phdr;
```

Project 1 – Bootloader

- BIOS functions
  - printch(ch)
    - Print character to serial port
    - address: 0x80011140
  - printstr(str)
    - Print string to serial port
    - address: 0x80011100
  - read_sd_card(address, offset, size)
    - Read SSD card
    - address: 0x80011000
  - Learn to invoke BIOS functions using MIPS code

Project 1 – Bootloader

- Step by step
  - Task 1: setup the environment (Project 0)
    - Need to be done today
  - Task 2: develop a simple bootloader to only print characters
    - Help to understand how the bootloader work and how to invoke BIOS functions

Project 1 – Bootloader

- Step by step
  - Task 3: given createimage, develop kernel.c and bootblock.s to start a kernel
    - Use BIOS function read_sd_card to load kernel
  - Task 4: develop your own createimage.c
    - Understand parsing ELF file
    - Understand constructing image

Project 1 – Bootloader

- Requirement for design review
  - Answer following questions
    - Show the in-memory layout of your bootblock and kernel
    - How do you invoke BIOS function? Show example code to invoke read_sd_card
    - How do you combine kernel.o and bootloader into an bootable image ?

Project 1 – Bootloader

- Requirement for developing
  - Finish following codes
    - Using read_sd_card function to place kernel image into memory: 15
    - Give control to the kernel: 15
    - Create image: 20
    - Extended flag: 5
    - Kernel runs: 5

Project 1 – Bootloader

- Bonus (1 point)
  - Imagine you have limited memory space, after executing bootloader, you need to reuse the memory space occupied by the bootloader starting from the address 0xa080 0000, please support such scenario to save your memory space.

Tips

- Learn to work on Linux
  - Watch out the outputs
- Read the task assignments carefully
- Pay attention to the memory address when you place kernel images

Tips

- About asking questions
  - Think and try to describe your problem clearly
    - Pls. do not just show us a screenshot
  - Discuss with your groupmate/classmate
  - Google search is a good way to help you