Lecture 1 Bootloader

2025.09.24

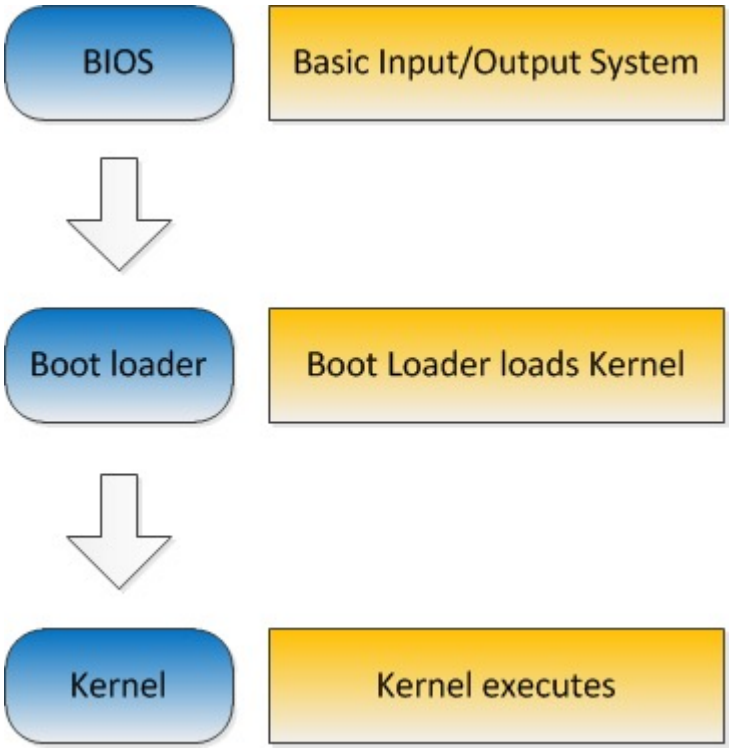
中国科学院大学
University of Chinese Academy of Sciences

Project 1 – Bootloader

- Requirements
  - Write a bootloader to start a tiny kernel, and the kernel is able to load a few user applications

Project 1 – Bootloader

- Booting procedure



Project 1 – Bootloader

- BIOS
  - Basic Input/Output System
  - Firmware used to perform hardware initialization after power-on
  - Automatically load bootloader in bootblock
- 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

- Memory layout
  - BBL (BIOS)
    - 0x5000 0000 – 0x501F FFFF
  - Bootloader will be placed at 0x5020 0000 by BIOS
  - Place your kernel at 0x5020 1000 use your own bootloader

地址范围	权限	作用
0x38000000-0x3800FFFF	ARW	clint
0x3C000000-0x3FFFFFFF	ARW	interrupt-controller
0x50000000-0x5FFFFFFF	RWXC	memory
0xE0000000-0xE0000FFF	RW	serial
0xE000B000-0xE000BFFF	RW	ethernet
0xE0100000-0xE0100FFF	RW	sdio
0xF8000000-0xF800FFFF	RW	slcr

Project 1 – Bootloader

- Memory layout
  - Makefile in start-code defines the location of you kernel in memory

```
37  BOOT_INCLUDE     = -I$(DIR_ARCH)/include
38  BOOT_CFLAGS      = $(CFLAGS) $(BOOT_INCLUDE) -Wl,--defsym=TEXT_START=$(BOOTLOADER_ENTRYPOINT) -T riscv.lds
39
40  KERNEL_INCLUDE   = -I$(DIR_ARCH)/include -Iinclude
41  KERNEL_CFLAGS    = $(CFLAGS) $(KERNEL_INCLUDE) -Wl,--defsym=TEXT_START=$(KERNEL_ENTRYPOINT) -T riscv.lds
```

```
65  BOOTLOADER_ENTRYPOINT    = 0x50200000
66  KERNEL_ENTRYPOINT        = 0x50201000
67  USER_ENTRYPOINT          = 0x52000000
```

Project 1 – Bootloader

- Tiny kernel
  - head.S
    - Define .entry_function section
    - Clear bss segment
    - Prepare runtime for kernel, such as setting stack pointer
    - Call kernel main function

Project 1 – Bootloader

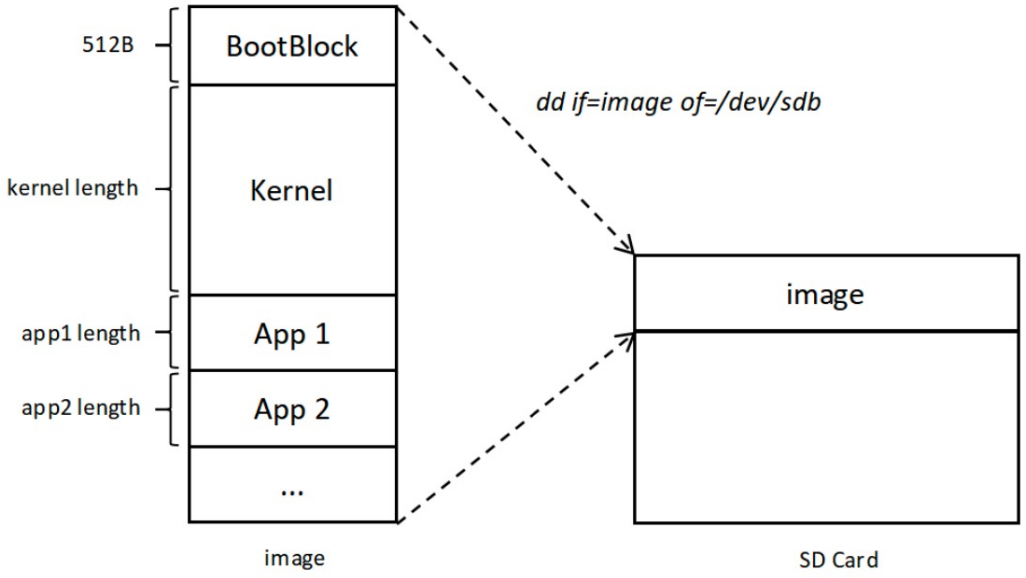
- Tiny kernel
  - main.c
    - Check bss segment
    - Initialize jump table for calling lower-level BIOS functions
    - Print welcome message
    - Initialize test applications
    - Wait for input
      - Echo input
      - Load and execute test applications

Project 1 – Bootloader

- Jump table for BIOS functions
  - BBL provides SBI(Supervisor Binary Interface) to support reading SD card, putting/getting char to/from console
  - You do not need to modify them, you can read source code for your interests
    - arch/riscv/bios/common.c
    - arch/riscv/include/asm/biosdef.h

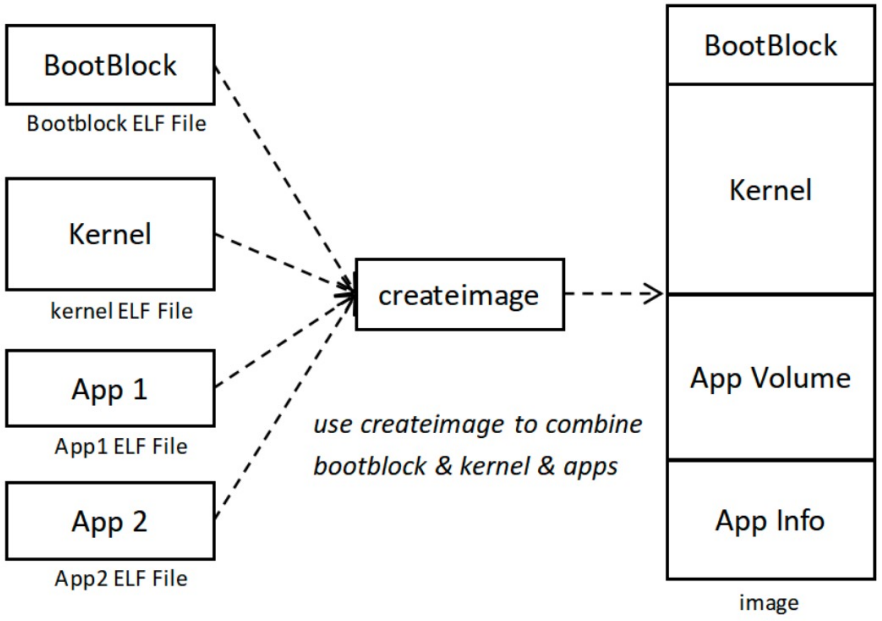
Project 1 – Bootloader

- Kernel image
  - Executable file
    - bootloader
    - Kernel
    - User applications



Project 1 – Bootloader

- Create image
  - Use createimage tool to generate the image
  - Parse ELF files and combine their segments

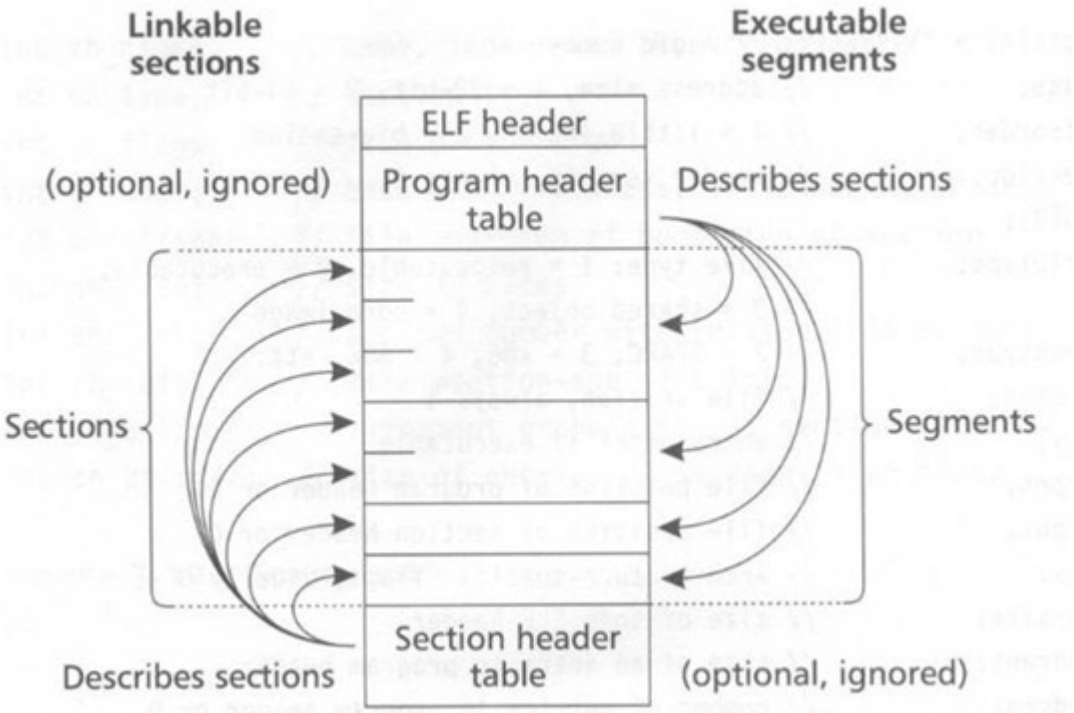


Project 1 – Bootloader

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

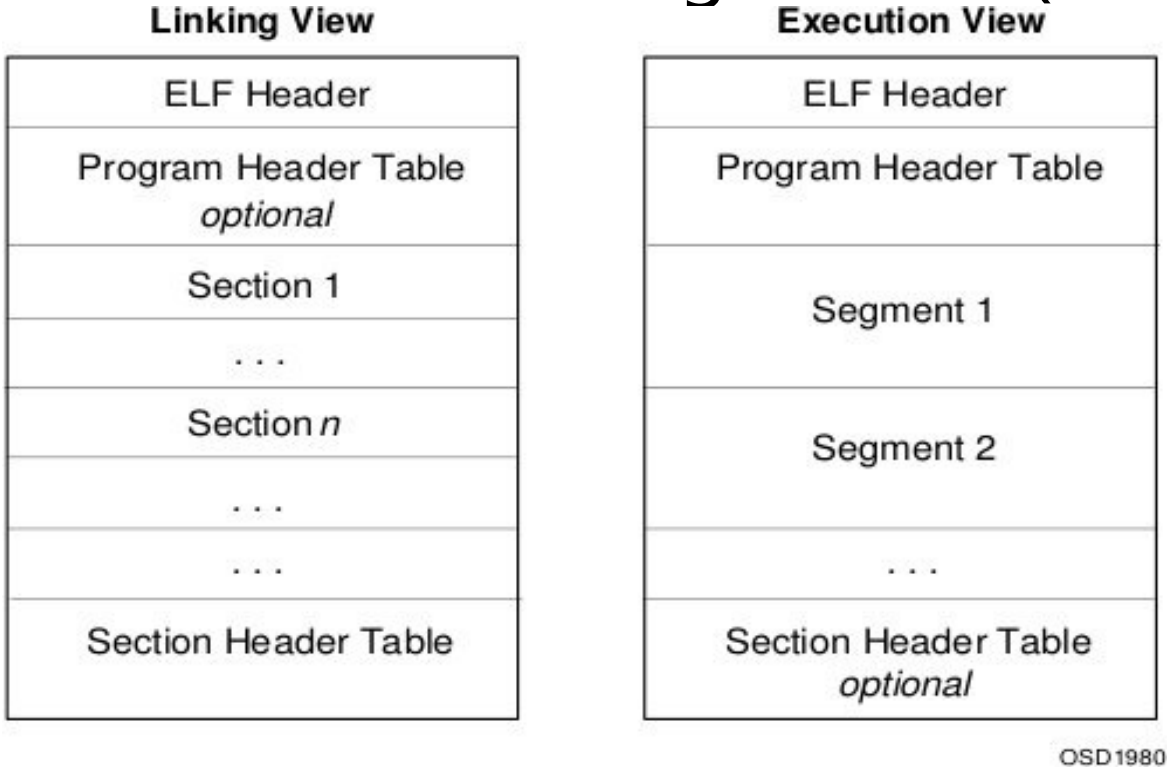
Project 1 – Bootloader

- 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;
```

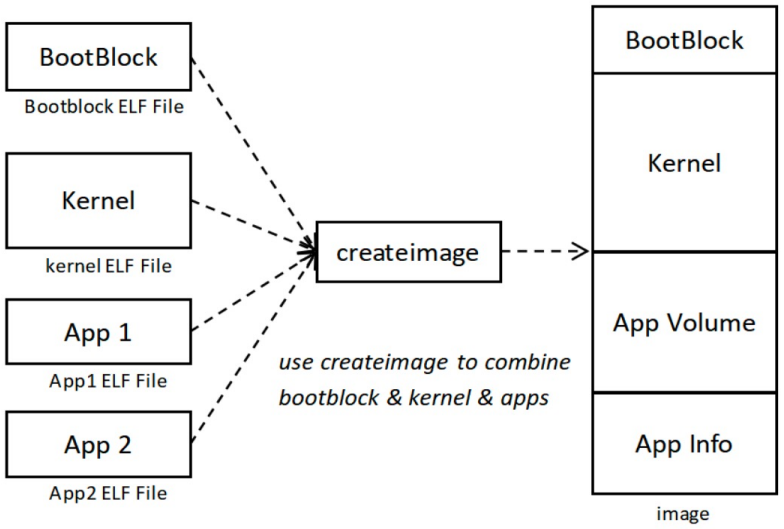
Project 1 – Bootloader

- ELF object file format
  - 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;
```

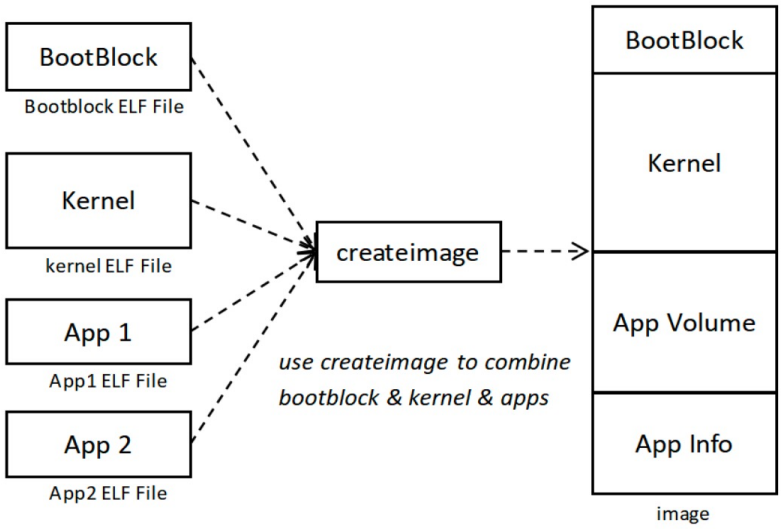
Project 1 – Bootloader

- App volume
  - Segments of user applications
  - You can use fix-sized sectors to store applications with padding
  - You can store applications in a compacted layout without padding



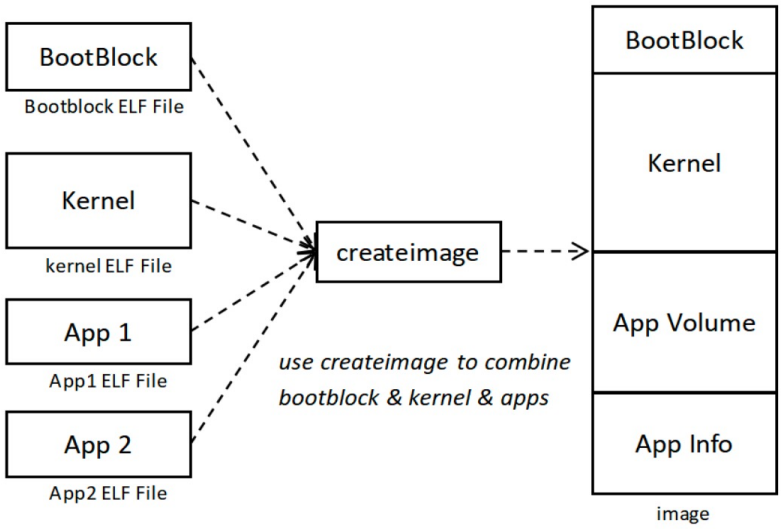
Project 1 – Bootloader

- App Info
  - A simple index to tell kernel
    - The sector offset and size of an application in the image
    - The memory location of the entry point of an application
    - Application name



Project 1 – Bootloader

- App Info
  - The app info is used by kernel to load an application
  - You need to write app info into image file, and let bootloader load it into memory when loading kernel



Project 1 – Bootloader

- Step by step – Task1
  - Develop a simple bootloader to only print characters
  - Help to understand how the bootloader work and how to invoke BIOS functions (using assembly programming)

Project 1 – Bootloader

- Step by step – Task 2
  - Develop bootblock.S to load kernel
  - Develop head.S to clear bss, set stack, and jump to kernel main function
  - Develop main.c to get console input and echo the inputted char
  - Use the given createimage to generate kernel image

Project 1 – Bootloader

- Step by step – Task 3
  - Develop createimage.c (write_img_info function) to add info into kernel image file, such as kernel size, user application number and location
    - Kernel, applications occupy the fix-sized sectors
  - Develop main.c to allow user input task ID for launching applications
  - Develop loader.c to support load user applications
  - Develop crt0.S to initialize C runtime for application

Project 1 – Bootloader

- Step by step – Task 4
  - Do not use fix-sized sectors to store kernel and applications. Instead, use their actual size when storing them into image file without padding
  - Use application name to launch an application

Project 1 – Bootloader

- Step by step – Task 5 (C-Core)
  - Implement batch processing to process multiple tasks（four tasks）one by one
    - The four tasks are described in the guidebook
  - Write a batch processing file to define the processing sequence. The sequence can be changed when re-writing the file
  - Implement three commands
    - List all applications' names
    - Write the batch processing file
    - Execute the batch processing

Project 1 – Bootloader

- Step by step – Task 5 (C-Core)
  - Two requirements
    - After a task finishes executing, the control flow should return to the kernel
    - The output of a previous task is the input of the latter one. You can prepare a fixed memory buffer for storing the ouput/input.

Project 1 – Bootloader

- Requirement for design review
  - 介绍P1的代码框架，以及和P1任务相关的代码文件和目录
  - 介绍bootBlock和内核分别如何调用BIOS提供的功能
  - 介绍内核如何加载一个应用，并启动该应用
  - 介绍image文件的组成和内部布局
  - 其他任何想交流讨论的问题

Project 1 – Bootloader

- Requirement for S/A/C-Core

Core type	Task requirements
S-Core	Tasks 1, 2, 3
A-Core	Tasks 1, 2, 3, 4
C-Core	Tasks 1, 2, 3, 4, 5

Tips

- Pls. first use QEMU for test. You will get the Pynq board at P1 due
- Learn to work on Linux
  - Read the outputs carefully
- Read the task handbook 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
    - Pls. do google searching/GPT querying before you ask questions
  - Discuss with your classmate

Schdule

- P1 design review
  - Oct. 11
- P1 due
  - Oct. 15