Assignment2

> 11911839 聂雨荷

Q1

> [20pts] `make qemu` 指令将指向 makefile 对应的 label，该指令对应于：
>
> ```
> qemu-system-riscv64 \
> -machine virt \
> -nographic \
> -bios default \
> -device loader,file=bin/ucore.bin,addr=0x80200000
> ```
>
> 请解释以上指令中每个参数的作用

- `-machine virt` : selects emulated machine `virt` (RISC-V VirtIO board)
- `-nographic` : disable graphical output and redirect serial I/Os to console
- `-bios default` : set the filename for the BIOS as default
- `-device loader,file=bin/ucore.bin,addr=0x80200000` : add misc device `loader` (call the bootloader OpenSBI), set the property `file` to be `bin/ucore.bin` and the qemu operation system starts at address `0x80200000`

Q2

> [20pts] 请查阅资料，理解并解释 `/lab/tools/kernel.ld` 文件以下片段中每一行的作用（参考：https://sourceware.org/binutils/docs/ld/Scripts.html）
>
> ```
> SECTIONS
> {
>  /* Load the kernel at this address: "." means the current address */
>  . = BASE_ADDRESS;
>
>  .text : {
>  *(.text.kern_entry)
>  *(.text .stub .text.* .gnu.linkonce.t.*)
>  }
>
>  PROVIDE(etext = .); /* Define the 'etext' symbol to this value */
>
>  .rodata : {
>  *(.rodata .rodata.* .gnu.linkonce.r.*)
>  }
>
>  /* Adjust the address for the data segment to the next page */
>  . = ALIGN(0x1000);
> ```

```
SECTIONS{
    ...
}
```

- `SECTION` command describes the memory layout of the output file

```
/* Load the kernel at this address: `.` means the current address */
. = BASE_ADDRESS;
```

- set the value of the location counter `.` to be `BASE_ADDRESS`

```
.text : {
        *(.text.kern_entry)
        *(.text .stub .text.* .gnu.linkonce.t.*)
    }
```

- define an output section `.text` (program code)
- `*(.text.kern_entry)` : add any input sections ended by `.text.kern_entry` in all input files
- `*(.text .stub .text.* .gnu.linkonce.t.*)` : add any input sections ended by `.text` / `.stub` / `.text.*` / `.gnu.linkonce.t.*`
- since the location counter is `BASE_ADDRESS`, when the output section `.text` is defined, the linker will set the address of the `.text` section in the output file to be `BASE_ADDRESS`.

```
/* Define the 'etext' symbol to this value */
PROVIDE(etext = .);
```

- define an `etext` symbol, and set the value of the symbol to be the location counter `.`

```
.rodata : {
        *(.rodata .rodata.* .gnu.linkonce.r.*)
    }
```

- define an output section `.rodata` (read-only data)
- `*(.rodata .rodata.* .gnu.linkonce.r.*)` : add any input sections ended by `.rodata` / `.rodata.*` / `.gnu.linkonce.r.*`
- since the location counter doesn't changed, the `.rodata` output section is allocated directly after the `.text` output section

```
/* Adjust the address for the data segment to the next page */
    . = ALIGN(0x1000);
```

- adjust the address for the data segment to the next page

Q3

> [10pts] 请解释 `/lab/kern/init/init.c` 中 main函数中 `memset(edata, 0, end - edata);` 的参数及语句作用。（需要读到的代码有 `init.c`，`kernel.ld`）

- `edata`: the pointer to the start block of the memory to be filled
- `0`: memory address to be set
- `end - edata`: number of bytes start from the `edata` to be set
- `edata` 和 `end` 分别记录了链接器输出的 `.bss` (read-write zero initialized data.) 的起始位置和终止位置的地址
- `memset()` 函数将 `end-edata` 个 byte，`edata` 中的也就是 `.bss` 文件的数据放入 memory 的地址 `0` 处，完成初始化

Q4

> [20pts] 请描述 `cputs()` 指令是如何通过 sbi 打印字符的。

函数顺序如下：

```
/* *
 * cputs- writes the string pointed by @str to stdout and
 * appends a newline character.
 * */
int cputs(const char *str) {
    int cnt = 0;
    char c;
    while ((c = *str++) != '\0') {
        cputch(c, &cnt);
    }
    cputch('\n', &cnt);
    return cnt;
}

/* *
 * cputch - writes a single character @c to stdout, and it will
 * increace the value of counter pointed by @cnt.
 * */
static void cputch(int c, int *cnt) {
    cons_putc(c);
```

```c
    (*cnt)++;
}

/* cons_putc - print a single character @c to console devices */
void cons_putc(int c) {
    sbi_console_putchar((unsigned char)c);
}

void sbi_console_putchar(unsigned char ch) {
    sbi_call(SBI_CONSOLE_PUTCHAR, ch, 0, 0);
}

uint64_t sbi_call(uint64_t sbi_type, uint64_t arg0, uint64_t arg1, uint64_t
arg2) {
    uint64_t ret_val;
    __asm__ volatile (
        "mv x17, %[sbi_type]\n"
        "mv x10, %[arg0]\n"
        "mv x11, %[arg1]\n"
        "mv x12, %[arg2]\n" //mv操作把参数的数值放到寄存器里
        "ecall\n" //参数放好之后，通过ecall，交给OpenSBI来执行
        "mv %[ret_val], x10"
        //OpenSBI按照riscv的calling convention,把返回值放到x10寄存器里
        //我们还需要自己通过内联汇编把返回值拿到我们的变量里
        : [ret_val] "=r" (ret_val)
        : [sbi_type] "r" (sbi_type), [arg0] "r" (arg0), [arg1] "r" (arg1),
        [arg2] "r" (arg2)
        : "memory"
    );
    return ret_val;
}

```

- 传入字符串指针 `*str`，每一个字符串以 `\0` 作为结尾
- 遍历字符串中每一个字符，对于每一个字符，调用 `cputch()`，将单个字符写入stdout，它将增加由 cnt 指向的 counter 的值
- `cputch()` 是对于 `sbi_console_putchar()` 的简单封装，字符继续向下传入
- `sbi_console_putchar()` 向下调用 `sbi_ecall()`，向 OpenSBI 传入 ecall 的指令是 `SBI_CONSOLE_PUTCHAR`，并且传递字符
- `sbi_call()` 使用内联汇编，将包括 `sbi_type` 和字符放对应位置后，调用 `ecall` 指令，启动调用 OpenSBI 进入 M 态处理

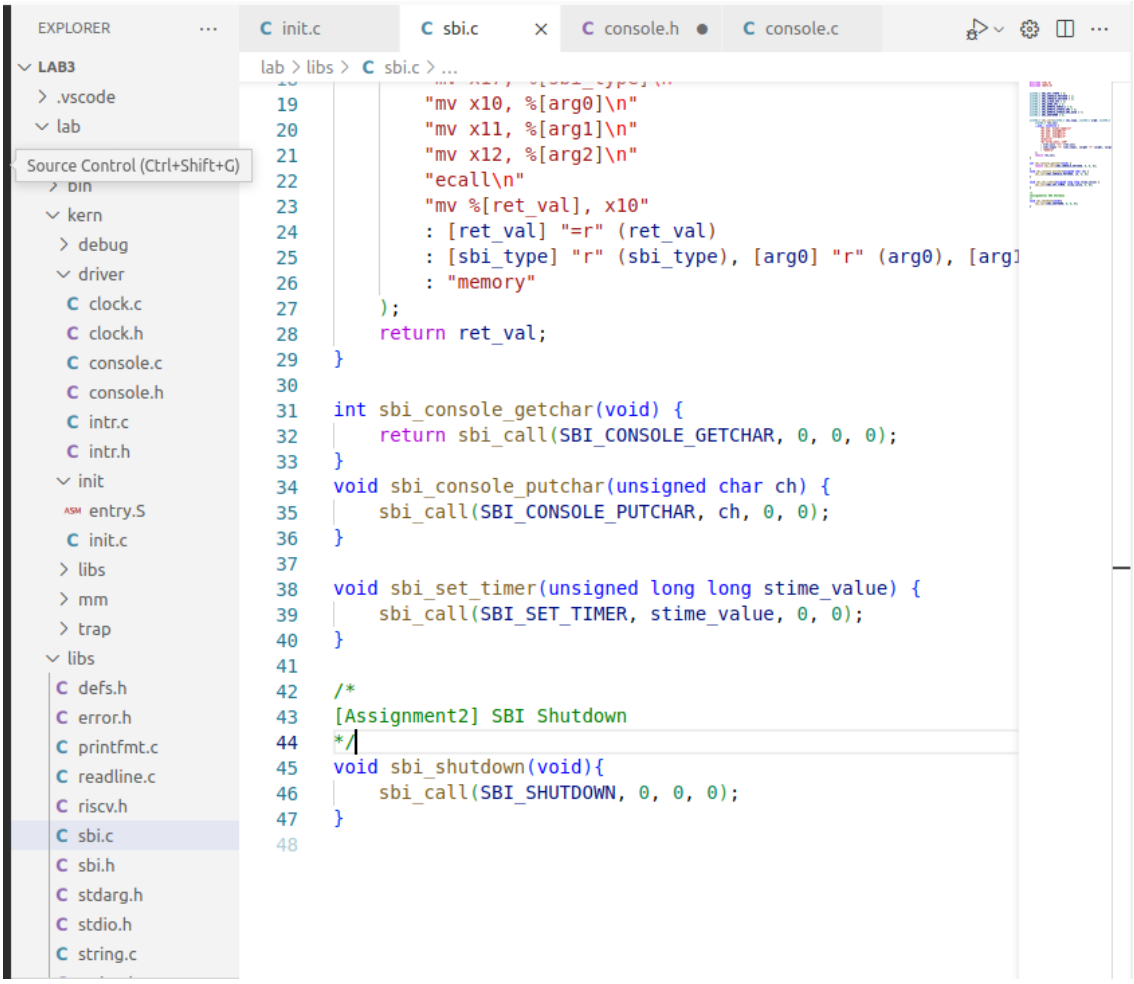
Q5

> [30pts] 编程题 请在第三周 lab.zip 代码包的基础上，理解使用 ecall 打印字符的原理，实现一个 `shutdown()` 关机函数。（所有修改到的代码请截图和运行结果截图一起放在报告中）
>
> 在 while(1); 前面填加：

```c
//init.c
//-----------------------------------
cputs("The system will close.\n");
shutdown();
// -----------------------------------
while (1)
    ;
```

实现效果(代码不会执行到while语句)：

参考资料：https://github.com/riscv-non-isa/riscv-sbi-doc/blob/master/riscv-sbi.adoc#410-function-listing



EXPLORER

LAB3

lab
- .vscode
- bin
- kern
  - debug
  - driver
    - clock.c
    - clock.h
    - console.c
    - console.h
    - intr.c
    - intr.h

init.c | sbi.c | console.h | console.c

lab > kern > driver > console.h > shutdown(void)

```c
#ifndef __KERN_DRIVER_CONSOLE_H__
#define __KERN_DRIVER_CONSOLE_H__

void cons_init(void);
void cons_putc(int c);
int cons_getc(void);
void serial_intr(void);
void kbd_intr(void);
void shutdown(void); // Assignment2

#endif /* !__KERN_DRIVER_CONSOLE_H__ */
```

EXPLORER

LAB3

.vscode
lab
- .vscode
- bin
- kern
  - debug
  - driver
    - clock.c
    - clock.h
    - console.c
    - console.h
    - intr.c
    - intr.h
  - init
    - entry.S
    - init.c
  - libs
  - mm
  - trap
- libs
  - defs.h
  - error.h

init.c | sbi.c | console.h | console.c

lab > kern > driver > console.c > ...

```c

/* serial_intr - try to feed input characters from serial por
void serial_intr(void) {}

/* cons_init - initializes the console devices */
void cons_init(void) {}

/* cons_putc - print a single character @c to console devices
void cons_putc(int c) { sbi_console_putchar((unsigned char)c)

/* *
 * cons_getc - return the next input character from console,
 * or 0 if none waiting.
 * */
int cons_getc(void) {
    int c = 0;
    c = sbi_console_getchar();
    return c;
}

/*
[Assignment2] shutdown
*/
void shutdown(void){ sbi_shutdown();}
```

.

EXPLORER

LAB3

- lab
  - .vscode
  - bin
  - kern
    - debug
    - driver
      - clock.c
      - clock.h
      - console.c
      - console.h
      - intr.c
      - intr.h
    - init
      - entry.S
      - init.c
    - libs
    - mm
    - trap
  - libs
    - defs.h
    - error.h
    - printfmt.c
    - readline.c
    - riscv.h
    - sbi.c
    - sbi.h
    - stdarg.h
    - stdio.h
    - string.c

OUTLINE

TIMELINE

init.c | sbi.c | console.h | console.c

lab > kern > init > init.c > kern_init(void)

```c
#include <kmonitor.h>
#include <pmm.h>
#include <riscv.h>
#include <stdio.h>
#include <string.h>
#include <trap.h>

int kern_init(void) __attribute__((noreturn));
void grade_backtrace(void);
static void lab1_switch_test(void);

int kern_init(void) {
    extern char edata[], end[];
    memset(edata, 0, end - edata);

    const char *message = "os is loading ...\n";
    cputs(message);

    // ----------start-------------
    cputs("The system will close.\n");
    shutdown();
    // -----------end--------------

    while (1)
        ;
}
```

nyh11911839@nyh-virtual-machine: ~/OSlab/lab3/lab

```
 | |  | |_ __   ___ _.__ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

Platform Name          : QEMU Virt Machine
Platform HART Features : RV64ACDFIMSU
Platform Max HARTs     : 8
Current Hart           : 0
Firmware Base          : 0x80000000
Firmware Size          : 120 KB
Runtime SBI Version    : 0.2

MIDELEG : 0x0000000000000222
MEDELEG : 0x000000000000b109
PMP0    : 0x0000000080000000-0x000000008001ffff (A)
PMP1    : 0x0000000000000000-0xffffffffffffffff (A,R,W,X)
os is loading ...

The system will close.

nyh11911839@nyh-virtual-machine:~/OSlab/lab3/lab$
```