

# Assignment2 Report

姓名: Yitong WANG(王奕童) [11910104@mail.sustech.edu.cn](mailto:11910104@mail.sustech.edu.cn)

学号: 11910104

实验课时段: 周五5-6节

实验课教师: Yun SHEN(沈昀) [sheny@mail.sustech.edu.cn](mailto:sheny@mail.sustech.edu.cn)

实验课SA:

- Yining TANG(汤怡宁) [11811237@mail.sustech.edu.cn](mailto:11811237@mail.sustech.edu.cn)
- Yushan WANG(王宇杉) [11813002@mail.sustech.edu.cn](mailto:11813002@mail.sustech.edu.cn)

## Q1 qemu指令参数

优先考虑使用 `qemu-system-riscv64 -help` 命令查看帮助文档:

```
wyt11910104@wyt11910104-virtual-machine: ~/Desktop/Assignment/A2/code_lab3/lab3$ qemu-system-riscv64 -help^C
wyt11910104@wyt11910104-virtual-machine: ~/Desktop/Assignment/A2/code_lab3/lab3$ qemu-system-riscv64 -help
QEMU emulator version 5.0.0
Copyright (c) 2003-2020 Fabrice Bellard and the QEMU Project developers
usage: qemu-system-riscv64 [options] [disk_image]

'disk_image' is a raw hard disk image for IDE hard disk 0

Standard options:
-h or -help      display this help and exit
-version        display version information and exit
-machine [type=]name[,prop[=value]][,...]
    selects emulated machine ('-machine help' for list)
    property accel=accel1[:accel2[:...]] selects accelerator
    supported accelerators are kvm, xen, hax, hvf, whpx or tcg (default: tcg)
    vmport=on|off|auto controls emulation of vmport (default: auto)
    dump-guest-core=on|off include guest memory in a core dump (default=on)
    mem-merge=on|off controls memory merge support (default: on)
    aes-key-wrap=on|off controls support for AES key wrapping (default=on)
    dea-key-wrap=on|off controls support for DEA key wrapping (default=on)
    suppress-vmdesc=on|off disables self-describing migration (default=off)
    nvdimm=on|off controls NVDIMM support (default=off)
    enforce-config-section=on|off enforce configuration section migration (default=off)
    memory-encryption=@var{ } memory encryption object to use (default=none)
    hmat=on|off controls ACPI HMAT support (default=off)
    select CPU ('-cpu help' for list)
-cpu cpu
-accel [accel=]accelerator[,prop[=value]][,...]
    select accelerator (kvm, xen, hax, hvf, whpx or tcg; use 'help' for a list)
    igd-passthru=on|off (enable Xen integrated Intel graphics passthrough, default=off)
    kernel-irqchip=on|off|split controls accelerated irqchip support (default=on)
    kvm-shadow-mem=size of KVM shadow MMU in bytes
    tb-size=n (TCG translation block cache size)
    thread=single|multi (enable multi-threaded TCG)
-smp [cpus=n[,maxcpus=cpus][,cores=cores][,threads=threads][,dies=dies][,sockets=sockets]
    set the number of CPUs to 'n' [default=1]
    maxcpus= maximum number of total cpus, including
    offline CPUs for hotplug, etc
    cores= number of CPU cores on one socket (for PC, it's on one die)
    threads= number of threads on one CPU core
    dies= number of CPU dies on one socket (for PC only)
    sockets= number of discrete sockets in the system
-numa node[,mem=size][,cpus=firstcpu[-lastcpu]][,nodeid=node][,initiator=node]
-numa node[,memdev=id][,cpus=firstcpu[-lastcpu]][,nodeid=node][,initiator=node]
-numa dist,src=source,dst=destination,val=distance
-numa cpu,node-id=node[,socket-id=x][,core-id=y][,thread-id=z]
-numa hmat-lb,initiator=node,target=node,hierarchy=memory|first-level|second-level|third-level,data-type=access-latency|read-latency|write-latency[,latency=lat][,bandwidth=bw]
-numa hmat-cache,node-id=node,size=size,level=level[,associativity=none|direct|complex][,policy=none|write-back|write-through][,line=size]
-add-fd fd=fd,set=set[,opaque=opaque]
    Add 'fd' to fd 'set'
-set group.id,arg=value
    set <arg> parameter for item <id> of type <group>
    i.e. -set drive.$fd.file=/path/to/image
-global driver.property=value
-global driver=driver,property=property,value=value
    set a global default for a driver property
-boot [order=drives][,once=drives][,menu=on|off]
    [,splash=sp_name][,splash-time=sp_time][,reboot-timeout=rb_time][,strict=on|off]
    'drives': floppy (a), hard disk (c), CD-ROM (d), network (n)
    'sp_name': the file's name that would be passed to bios as logo picture, if menu=on
    'sp_time': the period that splash picture last if menu=on, unit is ms
    'rb_timeout': the timeout before guest reboot when boot failed, unit is ms
-m [size=megs[,slots=n,maxmem=size]
    configure guest RAM
    size: initial amount of guest memory
    slots: number of hotplug slots (default: none)
```

## -machine virt

从文档中，可见该参数的功能是选取仿真所用的机器：

```
-machine [type=]name[,prop[=value]][,...]
    selects emulated machine ('-machine help' for list)
    property accel=accel1[:accel2[:...]] selects accelerator
    supported accelerators are kvm, xen, hax, hvf, whpx or tcg (default: tcg)
    vmport=on|off|auto controls emulation of vmport (default: auto)
    dump-guest-core=on|off include guest memory in a core dump (default=on)
    mem-merge=on|off controls memory merge support (default: on)
    aes-key-wrap=on|off controls support for AES key wrapping (default=on)
    dea-key-wrap=on|off controls support for DEA key wrapping (default=on)
    suppress-vmdesc=on|off disables self-describing migration (default=off)
    nvdimm=on|off controls NVDIMM support (default=off)
    enforce-config-section=on|off enforce configuration section migration (default=off)
    memory-encryption=@var{ } memory encryption object to use (default=none)
    hmat=on|off controls ACPI HMAT support (default=off)
```

然后考虑使用 `qemu-system-riscv64 -machine help` 获取可用的机器列表，可以发现 virt：

```
wyt11910104@wyt11910104-virtual-machine:~/Desktop/Assignment/A2/code_lab3/lab3$ qemu-system-riscv64 -machine help
Supported machines are:
none                empty machine
sifive_e            RISC-V Board compatible with SiFive E SDK
sifive_u            RISC-V Board compatible with SiFive U SDK
spike              RISC-V Spike Board (default)
spike_v1.10        RISC-V Spike Board (Privileged ISA v1.10)
spike_v1.9.1       RISC-V Spike Board (Privileged ISA v1.9.1)
virt               RISC-V VirtIO board
```

因此该参数的意义为选取仿真所用的机器为：RISC-V VirtIO board

## -nographic

从文档中，可见该参数的功能是禁用图形输出并将串行I/O重定向到控制台：

```
Display options:
-display vnc=<display>[,<optargs>]
-display none
                        select display backend type
                        The default display is equivalent to
                        "-vnc localhost:0,to=99,id=default"
-nographic             disable graphical output and redirect serial I/Os to console
-curses                shorthand for -display curses
-alt-grab              use Ctrl-Alt-Shift to grab mouse (instead of Ctrl-Alt)
```

## -bios default

从文档中，可见该参数的功能是设置 BIOS 的文件名为 default

```
-dfilter range,..      filter debug output to range of addresses (useful for
-seed number          seed the pseudo-random number generator
-L path               set the directory for the BIOS, VGA BIOS and keymaps
-bios file            set the filename for the BIOS
-enable-kvm           enable kvm full virtualization support
-xen-domid id         specify xen guest domain id
-xen-attach           attach to existing xen domain
libxl will use this when starting QEMU
```

## -device loader,file=bin/ucore.bin,addr=0x80200000

先考虑在文档中找到-device参数对应的功能：

```
-device driver[,prop[=value][,...]]
                        add device (based on driver)
                        prop=value,... sets driver properties
                        use '-device help' to print all possible drivers
                        use '-device driver,help' to print all possible properties
```

然后考虑输入 `qemu-system-riscv64 -device loader,help` 查询名为loader的driver可用的参数列表：

```
wyt11910104@wyt11910104-virtual-machine:~/Desktop/Assignment/A2/code_lab3/lab3$ qemu-system-riscv64 -device loader,help
loader options:
  addr=<uint64>          - (default: 0)
  cpu-num=<uint32>       - (default: 4294967295)
  data-be=<bool>         - (default: false)
  data-len=<uint8>       - (default: 0)
  data=<uint64>          - (default: 0)
  file=<str>             - (default: loader.bin)
  force-raw=<bool>      - (default: false)
```

在这里 bin/ucore.bin 是编译出的可执行文件 **(ELF)**，它是未压缩的，带调试信息和符号表的。这些参数的意义在于将 bin/ucore.bin 这一架构程序加载至地址 0x80200000 处运行。

## Reference

操作系统如何自学？ - 水dong方块的回答 - 知乎

<https://www.zhihu.com/question/57257819/answer/2308387292>

# Q2 kernel.ld文件中每一行的作用

```
/* Simple linker script for the ucore kernel.
   See the GNU ld 'info' manual ("info ld") to learn the syntax. */
```

这一段是简单的注释，编译器会无视该部分。

```
OUTPUT_ARCH(riscv)
```

这句话表示设置输出文件对应的处理器架构为RiscV。

## Reference

<https://www.cnblogs.com/lCkeeper/p/15514775.html>

```
ENTRY(kern_entry)
```

这句话表示Entry Point **(EP)**，是BIOS移动完内核后，直接跳转的地址，是程序的入口。而kern\_entry是体系相关的汇编语言实现的。

## Reference

<https://blog.csdn.net/wangyao199252/article/details/74938761>

<https://sourceware.org/binutils/docs/ld/Entry-Point.html#Entry-Point>

```
BASE_ADDRESS = 0x80200000;
```

表示基地址，是低地址，链接脚本会从基地址放置.text, .rodata等等数据段。

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

这部分是链接脚本的整体，是用于描述整个内存布局，其内有输入段，输出段等等数据段。

```
. = BASE_ADDRESS;
```

这句话表示当前地址，让设置的地址从低地址往高地址做段的放置操作。这条语句的作用是记录当前段的地址..

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

.text 段即为代码段，里面的 \*(.text.kern\_entry .text .stub .text.\* .gnu.linkonce.t.\*) 会指示将工程中所有目标文件的 .text.kern\_entry , .text , .stub , .text.\* , .gnu.linkonce.t.\* 都链接到 FLASH中。其中 \* 是通配符，可以匹配符合前缀的任意文件。 .text 表示代码段的起始地址。

## Reference

<https://www.cnblogs.com/dylancao/p/9228885.html>

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

provide 关键字用于定义一个符号，如 etext 。如果程序中再次定义了该符号，则使用程序中定义的，否则则使用链接器脚本中的定义。这里是将 . 所代表的的地址值赋值给 etext 。

## Reference

<https://blog.csdn.net/x13015851932/article/details/48253695>

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

.rodata 字段用于定义**read-only data**，用于保存只读数据。里面的 \*(.rodata .rodata.\* .gnu.linkonce.r.\*) 会指示将工程中所有目标文件的 .rodata , .rodata.\* , .gnu.linkonce.r.\* 文件都链接到FLASH中。其中 \* 是通配符，可以匹配符合前缀的任意文件。

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

如之前所述，这条语句的作用是重设当前段的地址。其中 `ALIGN(0x1000)` 表示按照指定的边界进行排列，里面的参数必须是2的倍数。

```
/* The data segment */  
.data : {  
    *(.data)  
    *(.data.*)  
}
```

这里前一句是注释，编译器无视之。后面是用于定义 `data` 字段（数据段），里面会指示将工程中所有目标文件的 `.data`，`.data.*` 文件都链接到FLASH中。其中要注意：`data` 字段是用于保存初始化的全局数据。`.data` 是一个地址，表示代码段的结束地址，也是数据段的起始地址。

```
.sdata : {  
    *(.sdata)  
    *(.sdata.*)  
}
```

这里是用于定义 `sdata` 字段，它包含初始化的全局小数据，里面会指示将工程中所有目标文件的 `.sdata`，`.sdata.*` 文件都链接到FLASH中。

```
PROVIDE(edata = .);
```

如之前所介绍，这里是定义 `edata` 符号，并将 `.` 所代表的地址值赋值给 `edata`。`edata` 是 `bss` 段的开始地址。

```
.bss : {  
    *(.bss)  
    *(.bss.*)  
    *(.sbss*)  
}
```

这里是用于定义 `bss` 字段，它包含未初始化的全局数据，里面会指示将工程中所有目标文件的 `.bss`，`.bss.*`，`.sbss*` 文件都链接到FLASH中。`.bss` 是一个地址，表示数据段的结束地址和BSS段的起始地址。

## Reference

<https://www.wenjiangs.com/doc/b5owlkja>

```
PROVIDE(end = .);
```

如之前所介绍，这里是定义 `end` 符号，并将 `.` 所代表的地址值赋值给 `end`。`end` 表示BSS段的结束地址。

```
/DISCARD/ : {  
    *.eh_frame .note.GNU-stack  
}
```

`/DISCARD/` 关键字的作用是舍弃指定段，不会出现在输出文件中。在这里是舍弃掉 `.eh_frame` 和 ```.note.GNU-stack``` 段。

## Reference

<https://www.iteye.com/blog/yefzhu-1561933>

## Q3 `memset(edata, 0, end - edata);` 的参数及语句作用

注意Q2中的代码语句：

```
PROVIDE(edata = .);
```

```
.bss : {  
    *.bss  
    *.bss.*  
    *.sbss*  
}
```

```
PROVIDE(end = .);
```

`edata` 数据段是bss段的开始，而 `end` 数据段是bss段的结束。又考虑bss段的作用是存放初始化为0的可读写数据，因此这句语句的作用就是将bss段的数据内容全部初始化为0。

## Q4 `ecall`指令的调度流程

根据lab3，我们可知打印字符是在 `/kern/init/init.c` 文件中调用 `cputs` 以实现的：

```
init.c
~/Desktop/Assignment/A2/code_lab3/lab3/kern/init

1 #include <stdio.h>
2 #include <string.h>
3 #include <console.h>
4
5 int kern_init(void) __attribute__((noreturn));
6
7 int kern_init(void) {
8     extern char edata[], end[];
9     memset(edata, 0, end - edata);
10
11     const char *message = "os is loading ...\n";
12     cputs(message);
13
14     while (1)
15         ;
16 }
```

然后考虑顺藤摸瓜，找到 /kern/libs/stdio.c 中对 cputs 的定义，发现它调用了同文件中的 cputch 方法：

```
stdio.c
~/Desktop/Assignment/A2/code_lab3/lab3/kern/libs

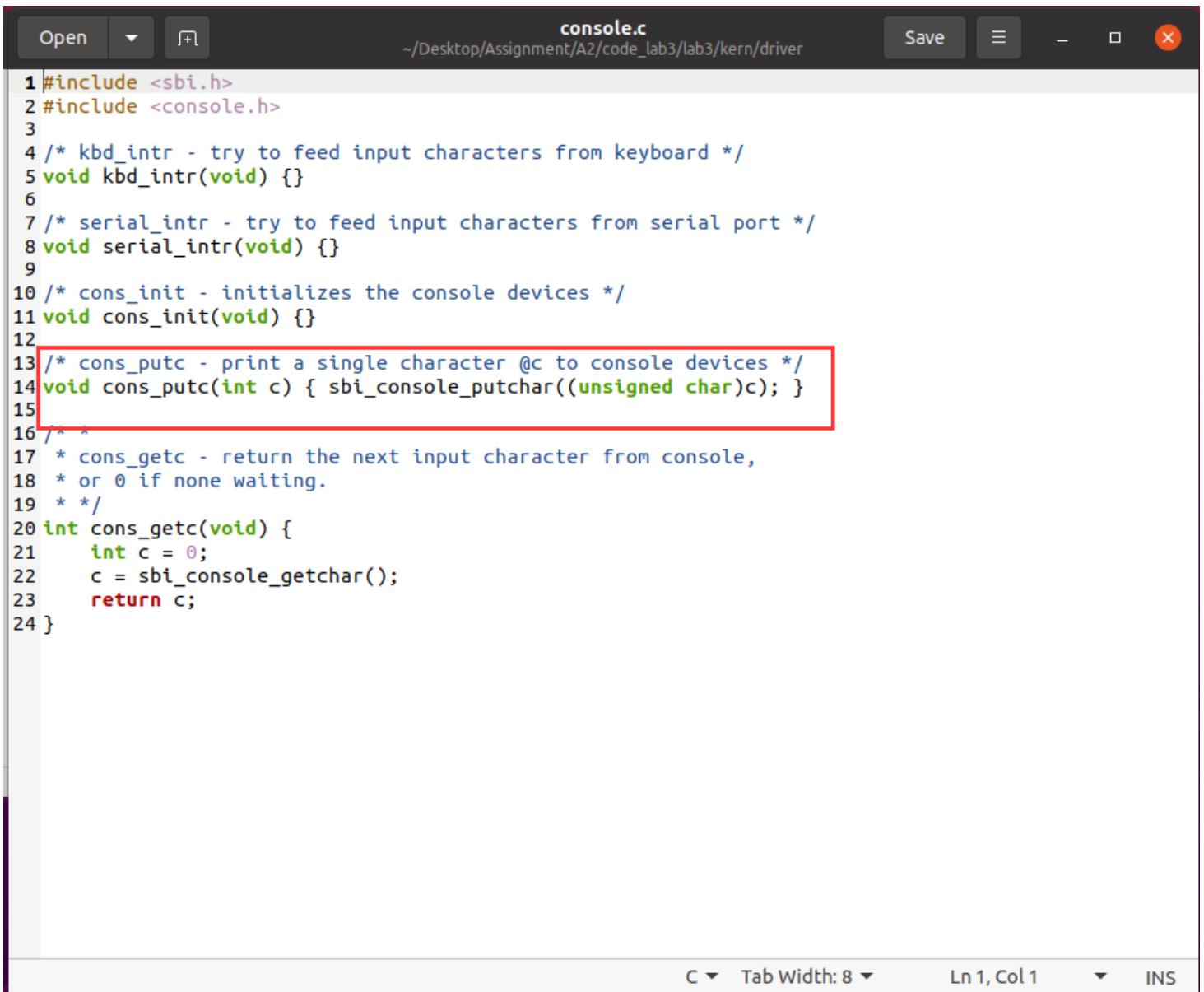
31 /*
32  * cprintf - formats a string and writes it to stdout
33  *
34  * The return value is the number of characters which would be
35  * written to stdout.
36  */
37 int cprintf(const char *fmt, ...) {
38     va_list ap;
39     int cnt;
40     va_start(ap, fmt);
41     cnt = vcprintf(fmt, ap);
42     va_end(ap);
43     return cnt;
44 }
45
46 /* cputchar - writes a single character to stdout */
47 void cputchar(int c) { cons_putc(c); }
48
49 /*
50  * cputs- writes the string pointed by @str to stdout and
51  * appends a newline character.
52  */
53 int cputs(const char *str) {
54     int cnt = 0;
55     char c;
56     while ((c = *str++) != '\0') {
57         cputch(c, &cnt);
58     }
59     cputch('\n', &cnt);
60     return cnt;
61 }
62
63 /* getchar - reads a single non-zero character from stdin */
64 int getchar(void) {
65     int c;
66     while ((c = cons_getc()) == 0) /* do nothing */;
67     return c;
68 }
```



```
Open  ▾  [icon]  stdio.c  ~/Desktop/Assignment/A2/code_lab3/lab3/kern/libs  Save  ≡  -  □  ✕

1 #include <console.h>
2 #include <defs.h>
3 #include <stdio.h>
4
5 /* HIGH level console I/O */
6
7 /* *
8  * cputch - writes a single character @c to stdout, and it will
9  * increase the value of counter pointed by @cnt.
10  * */
11 static void cputch(int c, int *cnt) {
12     cons_putc(c);
13     (*cnt)++;
14 }
15
16 /* *
17  * vprintf - format a string and writes it to stdout
18  *
19  * The return value is the number of characters which would be
20  * written to stdout.
21  *
22  * Call this function if you are already dealing with a va_list.
23  * Or you probably want printf() instead.
24  * */
25 int vprintf(const char *fmt, va_list ap) {
26     int cnt = 0;
27     vprintfmt((void *)cputch, &cnt, fmt, ap);
28     return cnt;
29 }
30
31 /* *
32  * printf - formats a string and writes it to stdout
33  *
34  * The return value is the number of characters which would be
35  * written to stdout.
36  * */
37 int printf(const char *fmt, ...) {
38     va_list ap;
39     ap = va_start(ap, fmt);
40     vprintf(fmt, ap);
41     va_end(ap);
42 }
```

然后发现 cputch 方法中调用了 cons\_putc 方法，这是一个具体实现在 kern/driver/console.c 的方法：



```
1 #include <sbi.h>
2 #include <console.h>
3
4 /* kbd_intr - try to feed input characters from keyboard */
5 void kbd_intr(void) {}
6
7 /* serial_intr - try to feed input characters from serial port */
8 void serial_intr(void) {}
9
10 /* cons_init - initializes the console devices */
11 void cons_init(void) {}
12
13 /* cons_putc - print a single character @c to console devices */
14 void cons_putc(int c) { sbi_console_putchar((unsigned char)c); }
15
16 /*
17  * cons_getc - return the next input character from console,
18  * or 0 if none waiting.
19  */
20 int cons_getc(void) {
21     int c = 0;
22     c = sbi_console_getchar();
23     return c;
24 }
```

The screenshot shows a code editor window titled "console.c" with the file path "~/Desktop/Assignment/A2/code\_lab3/lab3/kern/driver". The code is C language, defining several functions for console input/output. A red rectangular box highlights the definition of the `cons_putc` function on lines 13 and 14. The status bar at the bottom indicates "C", "Tab Width: 8", "Ln 1, Col 1", and "INS".

这里面调用的 `sbi_console_putchar` 方法实现在 `libs/sbi.c` 文件中，通过对其中的方法调用的分析，即可寻找到 `ecall` 指令的调用位置，这里面是通过内联汇编实现的调用：

```
sbi.c
~/Desktop/Assignment/A2/code_lab3/lab3/libs
Save

1 // ttps://sbi.c
2 #include <sbi.h>
3 #include <defs.h>
4
5
6 uint64_t SBI_SET_TIMER = 0;
7 uint64_t SBI_CONSOLE_PUTCHAR = 1;
8 uint64_t SBI_CONSOLE_GETCHAR = 2;
9 uint64_t SBI_CLEAR_IPI = 3;
10 uint64_t SBI_SEND_IPI = 4;
11 uint64_t SBI_REMOTE_FENCE_I = 5;
12 uint64_t SBI_REMOTE_SFENCE_VMA = 6;
13 uint64_t SBI_REMOTE_SFENCE_VMA_ASID = 7;
14 uint64_t SBI_SHUTDOWN = 8;
15
16 uint64_t sbi_call(uint64_t sbi_type, uint64_t arg0, uint64_t arg1, uint64_t arg2) {
17     uint64_t ret_val;
18     __asm__ volatile (
19         "mv x17, %[sbi_type]\n"
20         "mv x10, %[arg0]\n"
21         "mv x11, %[arg1]\n"
22         "mv x12, %[arg2]\n"
23         "ecall\n"
24         "mv %[ret_val], x10"
25         : [ret_val] "=r" (ret_val)
26         : [sbi_type] "r" (sbi_type), [arg0] "r" (arg0), [arg1] "r" (arg1), [arg2] "r" (arg2)
27         : "memory"
28     );
29     return ret_val;
30 }
31
32 void sbi_console_putchar(unsigned char ch) {
33     sbi_call(SBI_CONSOLE_PUTCHAR, ch, 0, 0);
34 }
35
36 void sbi_set_timer(unsigned long long stime_value) {
37     sbi_call(SBI_SET_TIMER, stime_value, 0, 0);
38 }
```

## Q5 shutdown关机函数的实现

首先截图运行结果：

```
wyt11910104@wyt11910104-virtual-machine:~/Desktop/Assignment/A2/code_lab3/lab3$ make clean
rm -f -r obj bin
wyt11910104@wyt11910104-virtual-machine:~/Desktop/Assignment/A2/code_lab3/lab3$ make
+ cc kern/init/entry.S
+ cc kern/init/init.c
+ cc kern/libs/stdio.c
+ cc kern/driver/console.c
+ cc libs/string.c
+ cc libs/printfmt.c
+ cc libs/readline.c
+ cc libs/sbi.c
m+ ld bin/kernel
riscv64-unknown-elf-objcopy bin/kernel --strip-all -O binary bin/ucore.bin
wyt11910104@wyt11910104-virtual-machine:~/Desktop/Assignment/A2/code_lab3/lab3$ make qemu
```

OpenSBI v0.6

# OpenSIS

```
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 : 0x00000000000000222
MEDELEG : 0x0000000000000b109
PMP0    : 0x0000000080000000-0x000000008001ffff (A)
PMP1    : 0x0000000000000000-0xffffffffffffffff (A,R,W,X)

os is loading ...
```

The system will close.

然后列出修改的代码区段:

- `libs/sbi.h` 中增加核心关机函数的函数声明, `libs/sbi.c` 中增加核心关机函数的函数具体实现, 这里考虑直接调用对应的参数 `SBI_SHUTDOWN` 和 `sbi_call` 方法:

```
sbi.h
~/Desktop/Assignment/A2/code_lab3/lab3/libs
Save

sbi.c
x
sbi.h
x

1 #ifndef _ASM_RISCV_SBI_H
2 #define _ASM_RISCV_SBI_H
3
4 typedef struct {
5     unsigned long base;
6     unsigned long size;
7     unsigned long node_id;
8 } memory_block_info;
9
10 unsigned long sbi_query_memory(unsigned long id, memory_block_info *p);
11
12 void sbi_set_timer(unsigned long long stime_value);
13 void sbi_send_ipi(unsigned long hart_id);
14 unsigned long sbi_clear_ipi(void);
15 void sbi_shutdown(void);
16
17 void sbi_console_putchar(unsigned char ch);
18 int sbi_console_getchar(void);
19
20 #endif
```

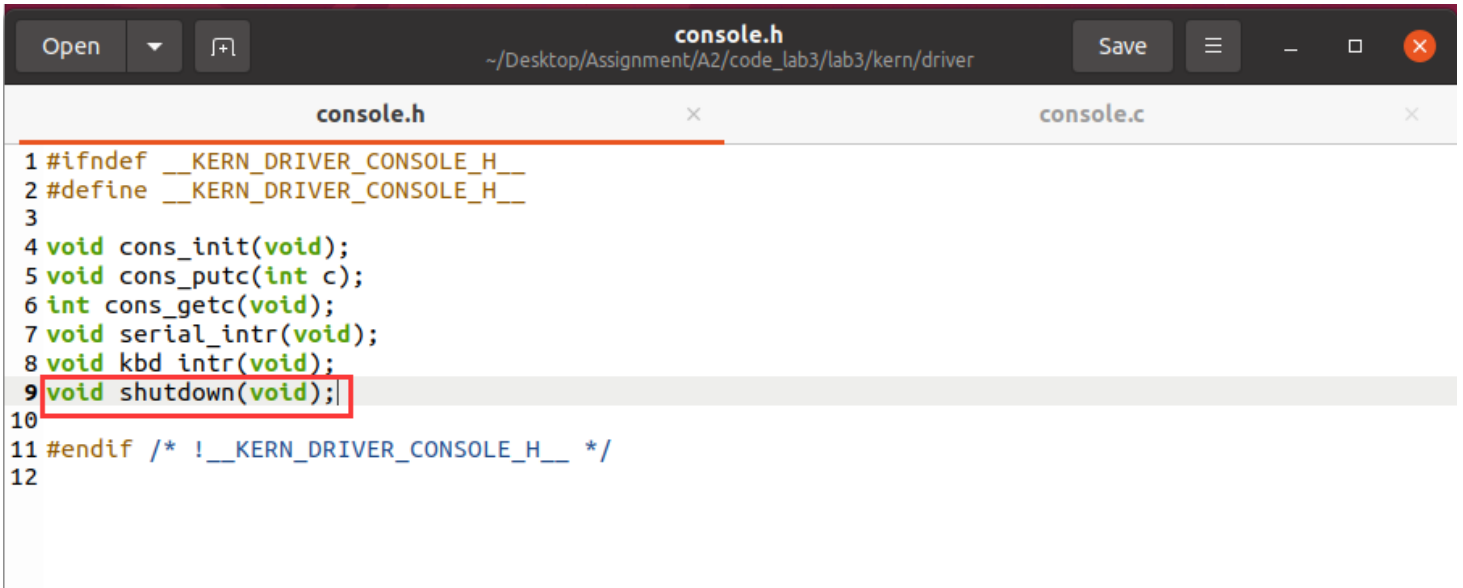
```
sbi.c
~/Desktop/Assignment/A2/code_lab3/lab3/libs
Save

5
6 uint64_t SBI_SET_TIMER = 0;
7 uint64_t SBI_CONSOLE_PUTCHAR = 1;
8 uint64_t SBI_CONSOLE_GETCHAR = 2;
9 uint64_t SBI_CLEAR_IPI = 3;
10 uint64_t SBI_SEND_IPI = 4;
11 uint64_t SBI_REMOTE_FENCE_I = 5;
12 uint64_t SBI_REMOTE_SFENCE_VMA = 6;
13 uint64_t SBI_REMOTE_SFENCE_VMA_ASID = 7;
14 uint64_t SBI_SHUTDOWN = 8;
15
16 uint64_t sbi_call(uint64_t sbi_type, uint64_t arg0, uint64_t arg1, uint64_t arg2) {
17     uint64_t ret_val;
18     __asm__ volatile (
19         "mv x17, %[sbi_type]\n"
20         "mv x10, %[arg0]\n"
21         "mv x11, %[arg1]\n"
22         "mv x12, %[arg2]\n"
23         "ecall\n"
24         "mv %[ret_val], x10"
25         : [ret_val] "=r" (ret_val)
26         : [sbi_type] "r" (sbi_type), [arg0] "r" (arg0), [arg1] "r" (arg1), [arg2] "r" (arg2)
27         : "memory"
28     );
29     return ret_val;
30 }
31
32 void sbi_shutdown(){
33     sbi_call(SBI_SHUTDOWN, 0, 0, 0);
34 }
35
36 void sbi_console_putchar(unsigned char ch) {
37     sbi_call(SBI_CONSOLE_PUTCHAR, ch, 0, 0);
38 }
39
40 void sbi_set_timer(unsigned long long stime_value) {
41     sbi_call(SBI_SET_TIMER, stime_value, 0, 0);
42 }
```

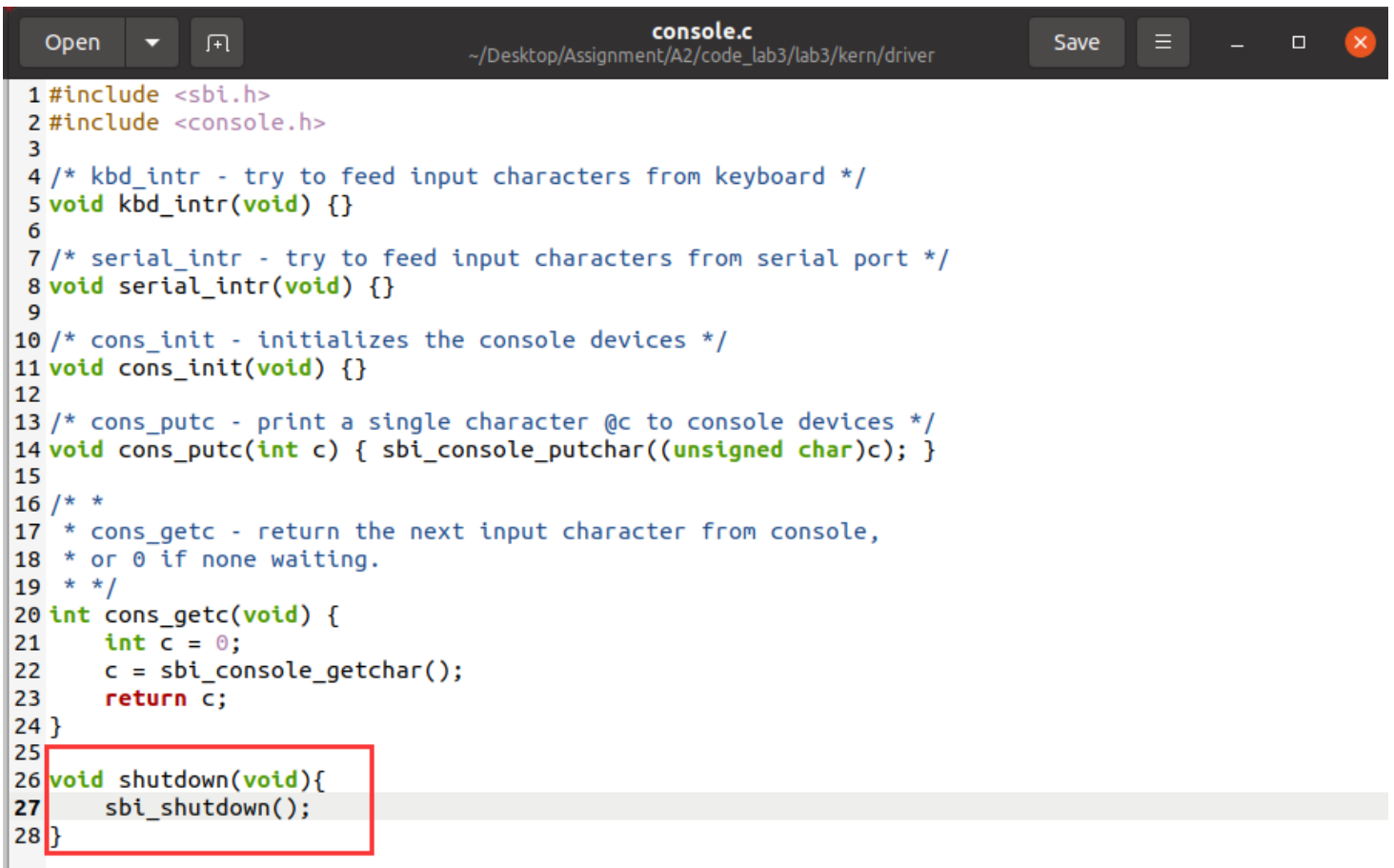
Bracket match found on line: 33

C Tab Width: 8 Ln 33, Col 35 INS

- 在 kern/driver/console.h 中增加中间调用关机函数的函数声明，并在 kern/driver/console.h 中增加对应的具体实现：




```
1 #ifndef __KERN_DRIVER_CONSOLE_H__
2 #define __KERN_DRIVER_CONSOLE_H__
3
4 void cons_init(void);
5 void cons_putc(int c);
6 int cons_getc(void);
7 void serial_intr(void);
8 void kbd_intr(void);
9 void shutdown(void);
10
11 #endif /* !__KERN_DRIVER_CONSOLE_H__ */
12
```



```
1 #include <sbi.h>
2 #include <console.h>
3
4 /* kbd_intr - try to feed input characters from keyboard */
5 void kbd_intr(void) {}
6
7 /* serial_intr - try to feed input characters from serial port */
8 void serial_intr(void) {}
9
10 /* cons_init - initializes the console devices */
11 void cons_init(void) {}
12
13 /* cons_putc - print a single character @c to console devices */
14 void cons_putc(int c) { sbi_console_putchar((unsigned char)c); }
15
16 /* *
17  * cons_getc - return the next input character from console,
18  * or 0 if none waiting.
19  * */
20 int cons_getc(void) {
21     int c = 0;
22     c = sbi_console_getchar();
23     return c;
24 }
25
26 void shutdown(void){
27     sbi_shutdown();
28 }
```

- 最后在 /kern/init/init.c 中调用对应的 shutdown 函数即可。

Open ▾  init.c ~/Desktop/Assignment/A2/code\_lab3/lab3/kern/init Save   

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <console.h>
4
5 int kern_init(void) __attribute__((noreturn));|
6
7 int kern_init(void) {
8     extern char edata[], end[];
9     memset(edata, 0, end - edata);
10
11     const char *message = "os is loading ...\n";
12     cputs(message);
13
14     cputs("The system will close.\n");
15     shutdown();
16
17     while (1)
18         ;
19 }
```

C ▾ Tab Width: 8 ▾ Ln 5, Col 47 ▾ INS