Week3-Assignment

11911609-葛兆宁

1.

qemu-system-riscv64 \

启动命令，后面才是参数

-machine virt \

选择启动的仿真机器类型，virt 指的是启动为 RISC-V VirtIO board 类型的仿真机器

-nographic \

禁止图形输出，将序列 IO 重定向为控制台

-bios default \

设定系统输入输出文件为 default

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

将 bin/ucore.bin 文件导入到内存地址为 0x80200000 的内存空间中

2. kern.ld（中文注释是我写的）：

kernel.ld

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

OUTPUT_ARCH(riscv)/*指定输出文件的系统架构为riscv*/
ENTRY(kern_entry)/*规定开始执行的第一条指令为kern_entry这条指令*/

BASE_ADDRESS = 0x80200000;/*指定基地址为0x80200000*/

SECTIONS/*表示以下部分用于描述程序输入的内存分布*/
{
    /* Load the kernel at this address: "." means the current address */
    . = BASE_ADDRESS;/*指定当前地址为基地址*/

    /*以下部分，
    .text通常用于存储代码部分与文本部分，只可读，
    .bss通常用于存放那些没有初始化的和初始化为0的全局变量和静态变量，
    .data通常用于存放那些初始化过（非零）的非const的全局变量和静态变量
    .rodata通常用于存放常量*/

    .text : {/*指定从当前地址开始,先排文本信息，将包括符合以下格式的输入文件信息合成一个section*/
        *(.text.kern_entry .text .stub .text.* .gnu.linkonce.t.*)
    }

    PROVIDE(etext = .); /* Define the 'etext' symbol to this value *//*定义etext的值为'.'，同时防止程序中定义同名的符号时链接出错*/

    .rodata : {/*排完文本后，再排rodata信息，将包括符合以下格式的输入文件信息合成一个section*/
        *(.rodata .rodata.* .gnu.linkonce.r.*)
    }

    /* Adjust the address for the data segment to the next page */
    . = ALIGN(0x1000);/*排完以上内容后进行地址对齐，将当前地址跳到对及后的地址位置*/

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

    .sdata : {/*同上*/
        *(.sdata)
        *(.sdata.*)
    }

    PROVIDE(edata = .);/*同上*/

    .bss : {/*同上*/
        *(.bss)
        *(.bss.*)
        *(.sbss*)
    }

    PROVIDE(end = .);/*同上*/

    /DISCARD/ : {/*输出时需要丢弃的section*/
        *(.eh_frame .note.GNU-stack)
    }
}
```

3. memset(edata, 0, end - edata)

参数 edata 为 void*(也就是说是地址）类型的，第二个 0 为 int 类型，第三个 end-edata 为 size_t 类型。

函数的意思是从 edata 开始到 edata+(end-edata)的地址，这一部分内存空间全设为 0。从 kernel.ld 可知 edata 与 end 之间是.bss 的 section，用于存放未初始化或初始化为 0 的变量，所以此句用于保证.bss section 部分全为 0。

4. OS 先调用 init.c 的主函数的程序，根据 init.c 的索引的 stdio.h 调用 stdio.c 中的 cputs 函数，cputs 函数根据索引的 console.h 文件调动 console.c 中的 cons_putc()函数，cons_putc()函数又根据 sbi.h 文件调用 sbi.c 中的 sbi_console_putchar()函数，sbi_console_putchar()函数调用

```
uint64_t sbi_call(uint64_t sbi_type, uint64_t arg0, uint64_t arg1, uint64_t arg2)
```

函数，sbi_call()函数根据将相应的参数存在寄存器中，然后进行 ecall 操作，之后 openSBI 收到 ecall 后根据 sbi_type 参数执行相应的操作

5.

执行结果：

```
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.

11911609JohnnyGe@johnny-Ge-WXX9:~/OS/lab3/code_lab3/lab3$
```

程序（仿照 cputs 写的）：

init.c:

```
cputs("The system will close.\n");
shutdown();
```

stdio.h 与 stdio.c:

```
void shutdown();
```

```
void shutdown()
{
    cons_shutdown();
}
```

console.h 与 console.c:

```
void cons_shutdown();
```

```
void cons_shutdown(void)
{
    sbi_shutdown();
}
```

sbi.h 与 sbi.c:

```
void sbi_shutdown(void);
```

```
void sbi_shutdown(void)
{
    sbi_call(SBI_SHUTDOWN,0,0,0);
}
```