CS302 OS Week3 Assignment - Report

Name: 刘仁杰
SID: 11911808

1) Explain the function of each parameter in `make qemu`

- `qemu-system-riscv64`: QEMU riscv64 emulator
- `-machine virt`: select emulated machine type, RISC-V VirtIO board
- `-nographic`: disable graphical output and redirect serial I/Os to console
- `-bios default`: set the filename for the BIOS, here default
- `-device loader,file=bin/ucore.bin,addr=0x80200000`: add device based on "Generic Loader", and sets driver properties, driver file path: `bin/ucore.bin`, driver program counter on start: `0x80200000`.

2) Explain the function of each line in `/lab3/tools/kernel.ld`

- `OUTPUT_ARCH(riscv)`: Specify a particular output machine architecture, here `riscv`.
- `ENTRY(kern_entry)`: The first instruction to execute in a program, here set to symbol `kern_entry`.
- `BASE_ADDRESS = 0x80200000;`: Set the variable `BASE_ADDRESS` to `0x80200000`.
- `SECTIONS{}`: Define the regions to set section mapping and memory layout.
- `. = BASE_ADDRESS;`: Set the location counter to be the value of `BASE_ADDRESS`.
- `.text : {*(.text.kern_entry .text .stub .text.* .gnu.linkonce.t.*)}`: Defines an output section `.text`, `*` is a wildcard which matches any file name, here maps all `.text.kern_entry`, `.text`, `.stub`, `.text.*`, `.gnu.linkonce.t.*` input sections in all input files to be placed into `.text` output section at **BASE_ADDRESS**, where `.text.*` and `.gnu.linkonce.t.*` refer to all sections beginning with `.text.` and `.gnu.linkonce.t.` in input files, also the location counter will self-increase when executing this command.
- `PROVIDE(etext = .);`: If the program defines `etext`, the linker will silently use the definition in the program. If the program references `etext` but does not define it, the linker will use the definition set as the current location counter.
- `.rodata : {*(.rodata .rodata.* .gnu.linkonce.r.*)}`: The same as above, defines an output section `.rodata` and maps all `.rodata`, `.rodata.*`, `.gnu.linkonce.r.*` input sections in all input files to be placed into `.rodata` output section at address **immediately after the `.text` output section** in memory with a **correct alignment**.
- `. = ALIGN(0x1000);`: Insert padding bytes until current location counter becomes aligned on `0x1000-byte` boundary.
- `.data : {*(.data)*(.data.*)}`: The same as above, defines an output section `.data` and maps all `.data`, `.data.*` input sections in all input files to be placed into `.data` output section at address **specified by the location counter**.
- `.sdata : {*(.sdata)*(.sdata.*)}`: The same as above, defines an output section `.sdata` and maps all `.data`, `.sdata.*` input sections in all input files to be placed into `.sdata` output section at address **specified by the location counter**.

- `PROVIDE(edata = .);`: If the program defines `edata`, the linker will silently use the definition in the program. If the program references `edata` but does not define it, the linker will use the definition set as the current location counter.
- `bss : {*(.bss)*(.bss.*)*(.sbss*)}`: The same as above, defines an output section `.bss` and maps all `.bss`, `.bss.*` and `.sbss*` input sections in all input files to be placed into `.bss` output section at address **specified by the location counter**.
- `PROVIDE(end = .);`: If the program defines `end`, the linker will silently use the definition in the program. If the program references `end` but does not define it, the linker will use the definition set as the current location counter.
- `/DISCARD/ : {*(.eh_frame .note.GNU-stack)}`: Drop the sections `.eh_frame` and `.note.GNU-stack` in all input files and thus they will not be contained in the output files.

3) Explain the function of `memset(edata, 0, end - edata);` and parameters

In `kernel.ld`, we have defined `edata` and `end` to be two memory locations. The output `bss` section is in this segment of memory. So this function is to set `end - edata` bytes (the `bss` section) to **0** starting from the memory location pointed by `edata`.

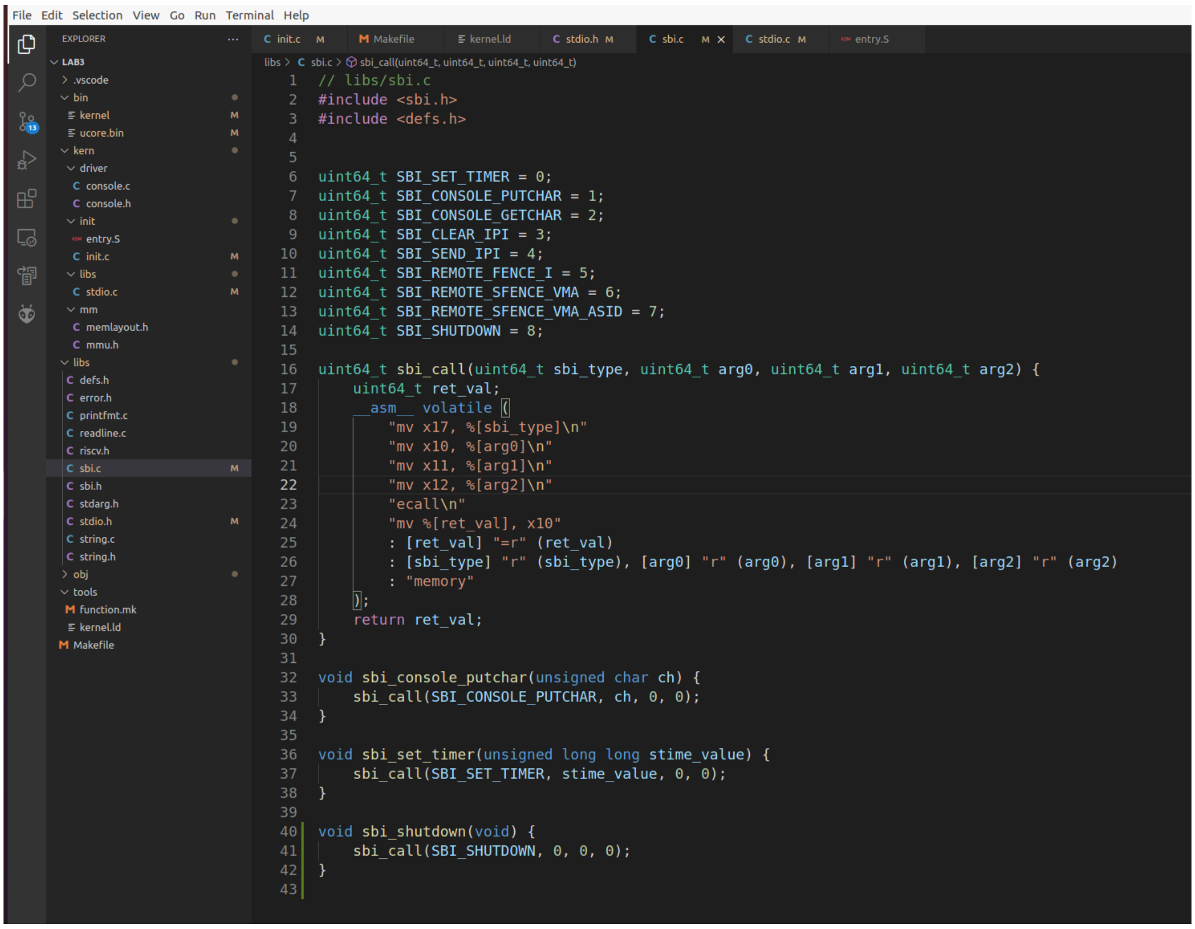
4) Describe how to call `ecall` instruction step by step after the kernel boot up

- When the kernel boot up, the first instruction to execute is set to `kern_entry` in `kernel.ld`.

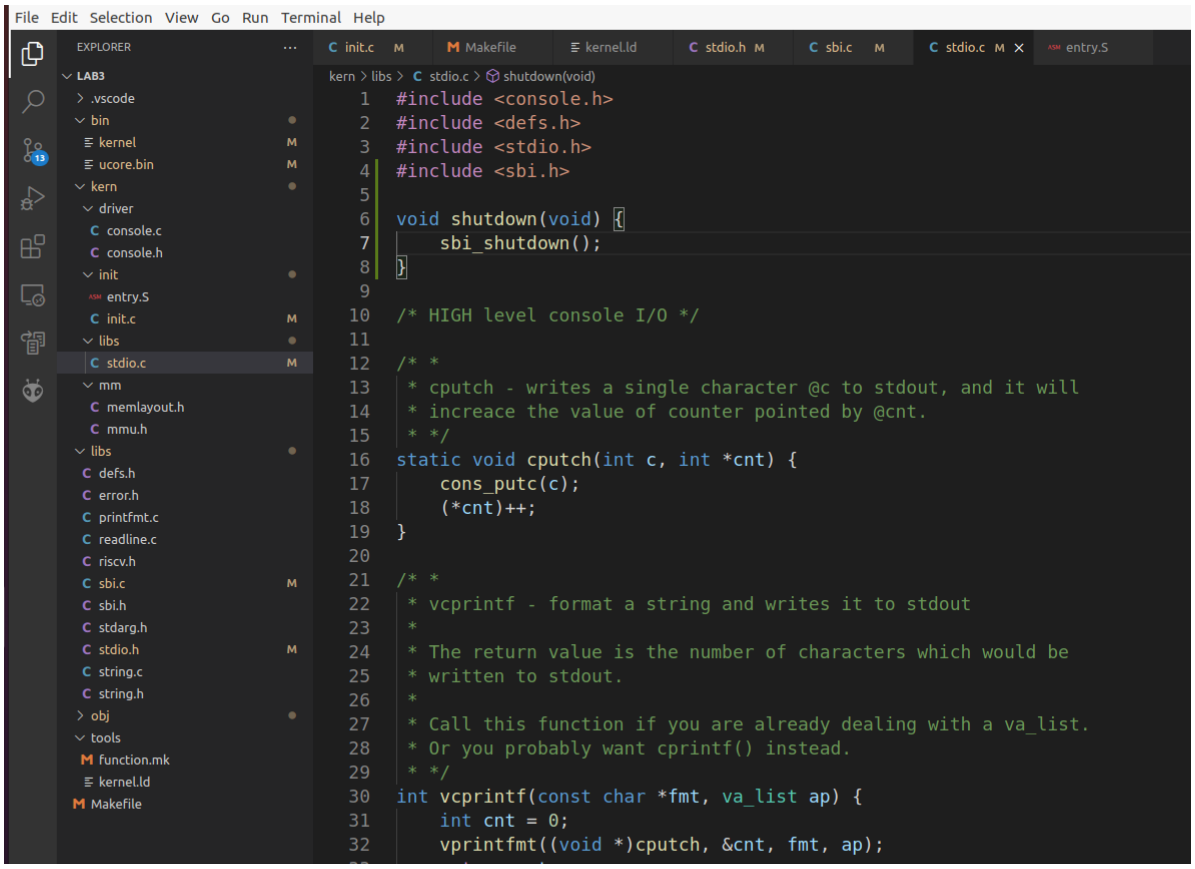
- Then in `entry.S`, the function `kern_init` in `init.c` is invoked in `kern_entry`.
- In function `kern_init`, `cputs` function is invoked.
- Then in `stdio.c`, `cputs` invokes `cputch` which further invokes `cons_putc` in `console.c`.
- In `console.c`, `cons_putc` invokes `sbi_console_putchar` in `sbi.c`.
- Finally, in `sbi.c`, `sbi_console_putchar` invokes `sbi_call` where `ecall` is called.

5) Refer to `ecall`, Implement `shutdown()` to shutdown the system.

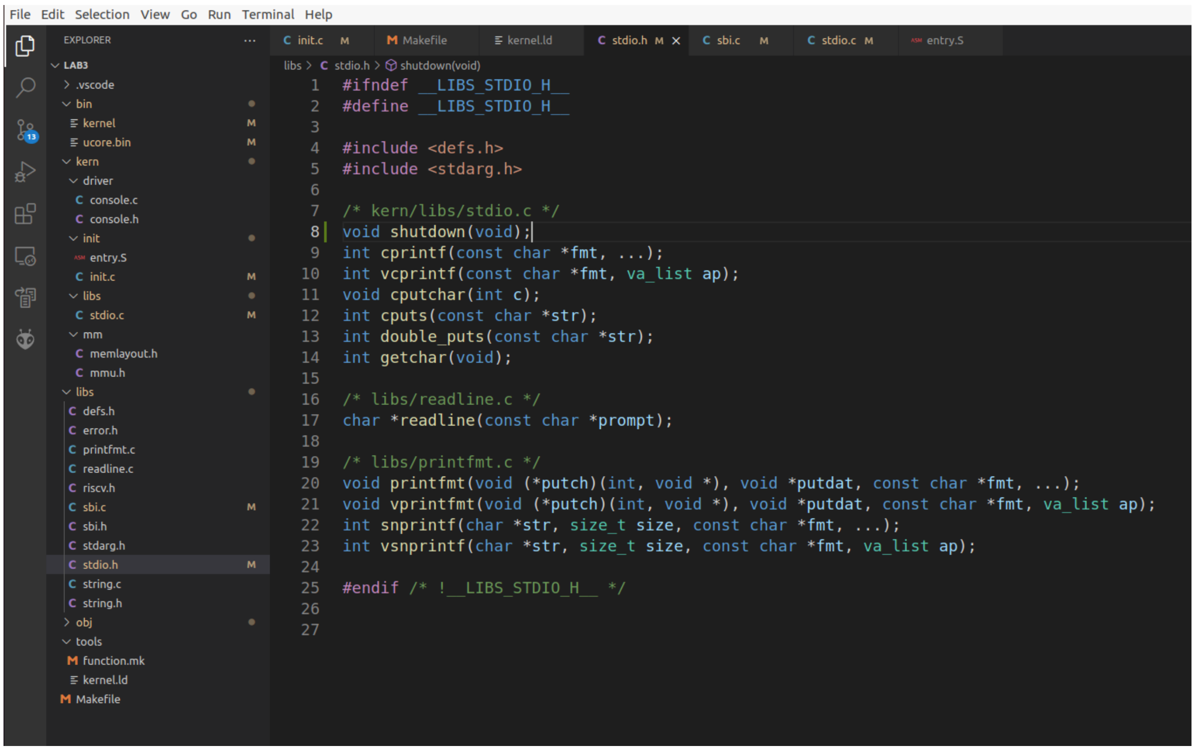
- Modified codes:



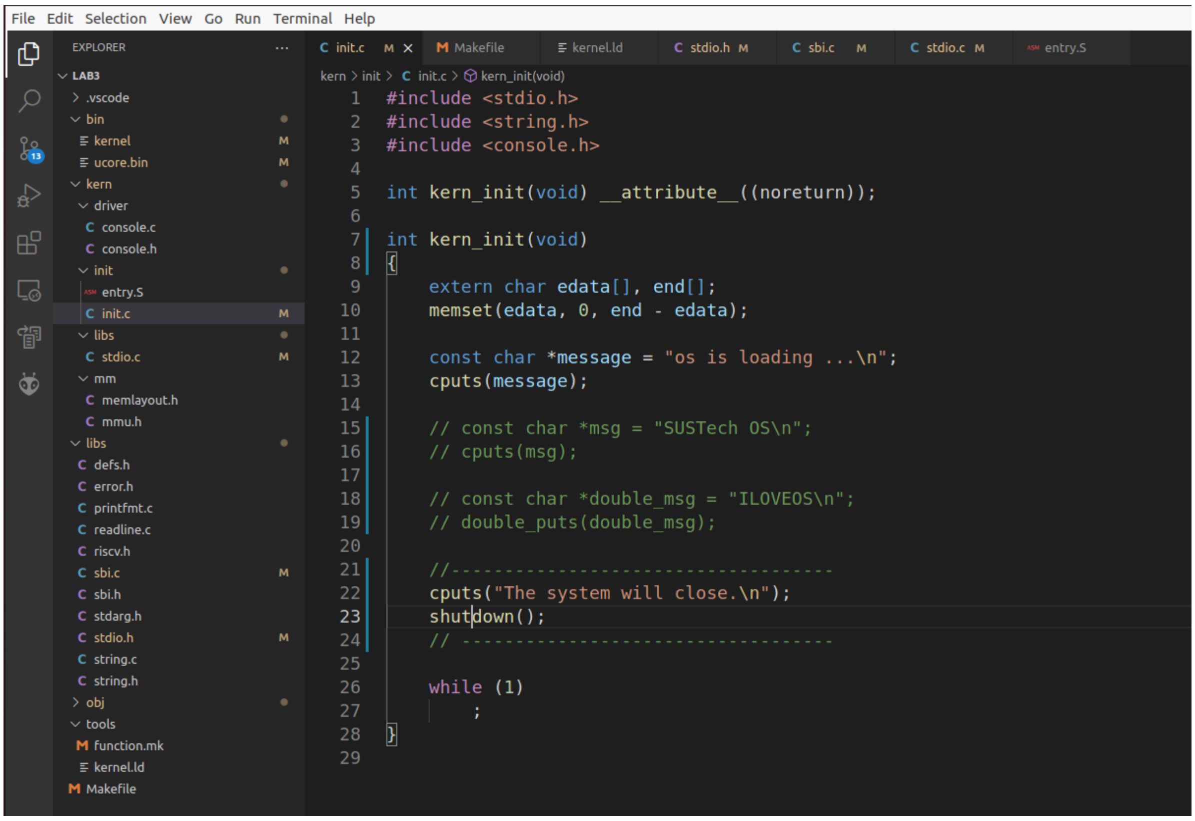
```c
// libs/sbi.c
#include <sbi.h>
#include <defs.h>


uint64_t SBI_SET_TIMER = 0;
uint64_t SBI_CONSOLE_PUTCHAR = 1;
uint64_t SBI_CONSOLE_GETCHAR = 2;
uint64_t SBI_CLEAR_IPI = 3;
uint64_t SBI_SEND_IPI = 4;
uint64_t SBI_REMOTE_FENCE_I = 5;
uint64_t SBI_REMOTE_SFENCE_VMA = 6;
uint64_t SBI_REMOTE_SFENCE_VMA_ASID = 7;
uint64_t SBI_SHUTDOWN = 8;

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"
        "ecall\n"
        "mv %[ret_val], x10"
        : [ret_val] "=r" (ret_val)
        : [sbi_type] "r" (sbi_type), [arg0] "r" (arg0), [arg1] "r" (arg1), [arg2] "r" (arg2)
        : "memory"
    );
    return ret_val;
}

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

void sbi_set_timer(unsigned long long stime_value) {
    sbi_call(SBI_SET_TIMER, stime_value, 0, 0);
}

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


File Edit Selection View Go Run Terminal Help
EXPLORER
LAB3
init.c
Makefile
kernel.ld
stdio.h
sbi.c
stdio.c
entry.S
kern > libs > stdio.c > shutdown(void)

```c
#include <console.h>
#include <defs.h>
#include <stdio.h>
#include <sbi.h>

void shutdown(void) {
    sbi_shutdown();
}

/* HIGH level console I/O */

/* *
 * 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);
    (*cnt)++;
}

/* *
 * vcprintf - format a string and writes it to stdout
 *
 * The return value is the number of characters which would be
 * written to stdout.
 *
 * Call this function if you are already dealing with a va_list.
 * Or you probably want cprintf() instead.
 * */
int vcprintf(const char *fmt, va_list ap) {
    int cnt = 0;
    vprintfmt((void *)cputch, &cnt, fmt, ap);
```

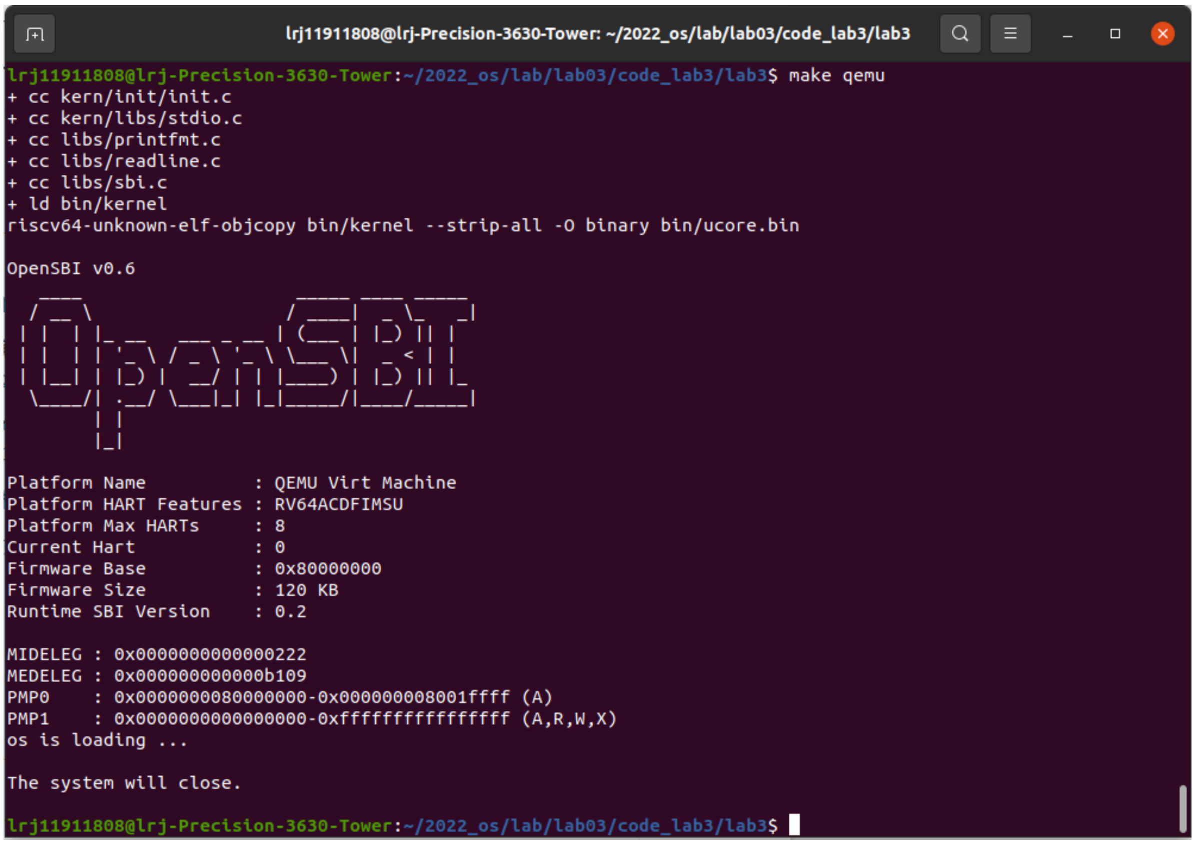

File Edit Selection View Go Run Terminal Help
EXPLORER
init.c M
Makefile
kernel.ld
stdio.h M
sbi.c M
stdio.c M
entry.S
libs > stdio.h > shutdown(void)

```c
#ifndef __LIBS_STDIO_H__
#define __LIBS_STDIO_H__

#include <defs.h>
#include <stdarg.h>

/* kern/libs/stdio.c */
void shutdown(void);
int cprintf(const char *fmt, ...);
int vcprintf(const char *fmt, va_list ap);
void cputchar(int c);
int cputs(const char *str);
int double_puts(const char *str);
int getchar(void);

/* libs/readline.c */
char *readline(const char *prompt);

/* libs/printfmt.c */
void printfmt(void (*putch)(int, void *), void *putdat, const char *fmt, ...);
void vprintfmt(void (*putch)(int, void *), void *putdat, const char *fmt, va_list ap);
int snprintf(char *str, size_t size, const char *fmt, ...);
int vsnprintf(char *str, size_t size, const char *fmt, va_list ap);

#endif /* !__LIBS_STDIO_H__ */
```


File Edit Selection View Go Run Terminal Help
EXPLORER
init.c M
Makefile
kernel.ld
stdio.h M
sbi.c M
stdio.c M
entry.S
kern > init > init.c > kern_init(void)

```c
#include <stdio.h>
#include <string.h>
#include <console.h>

int kern_init(void) __attribute__((noreturn));

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

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

    // const char *msg = "SUSTech OS\n";
    // cputs(msg);

    // const char *double_msg = "ILOVEOS\n";
    // double_puts(double_msg);

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

    while (1)
        ;
}
```

- Result:


lrj11911808@lrj-Precision-3630-Tower: ~/2022_os/lab/lab03/code_lab3/lab3

```
lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab03/code_lab3/lab3$ make qemu
+ cc kern/init/init.c
+ cc kern/libs/stdio.c
+ cc libs/printfmt.c
+ cc libs/readline.c
+ cc libs/sbi.c
+ ld bin/kernel
riscv64-unknown-elf-objcopy bin/kernel --strip-all -O binary bin/ucore.bin

OpenSBI v0.6
   ____                    _____ ____ _____
  / __ \                  / ____|  _ \_   _|
 | |  | |_ __   ___ _ __ | (___ | |_) || |
 | |  | | '_ \ / _ \ '_ \ \___ \|  _ < | |
 | |__| | |_) |  __/ | | |____) | |_) || |_
  \____/| .__/ \___|_| |_|_____/|____/_____|
        | |
        |_|

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.

lrj11911808@lrj-Precision-3630-Tower:~/2022_os/lab/lab03/code_lab3/lab3$
```