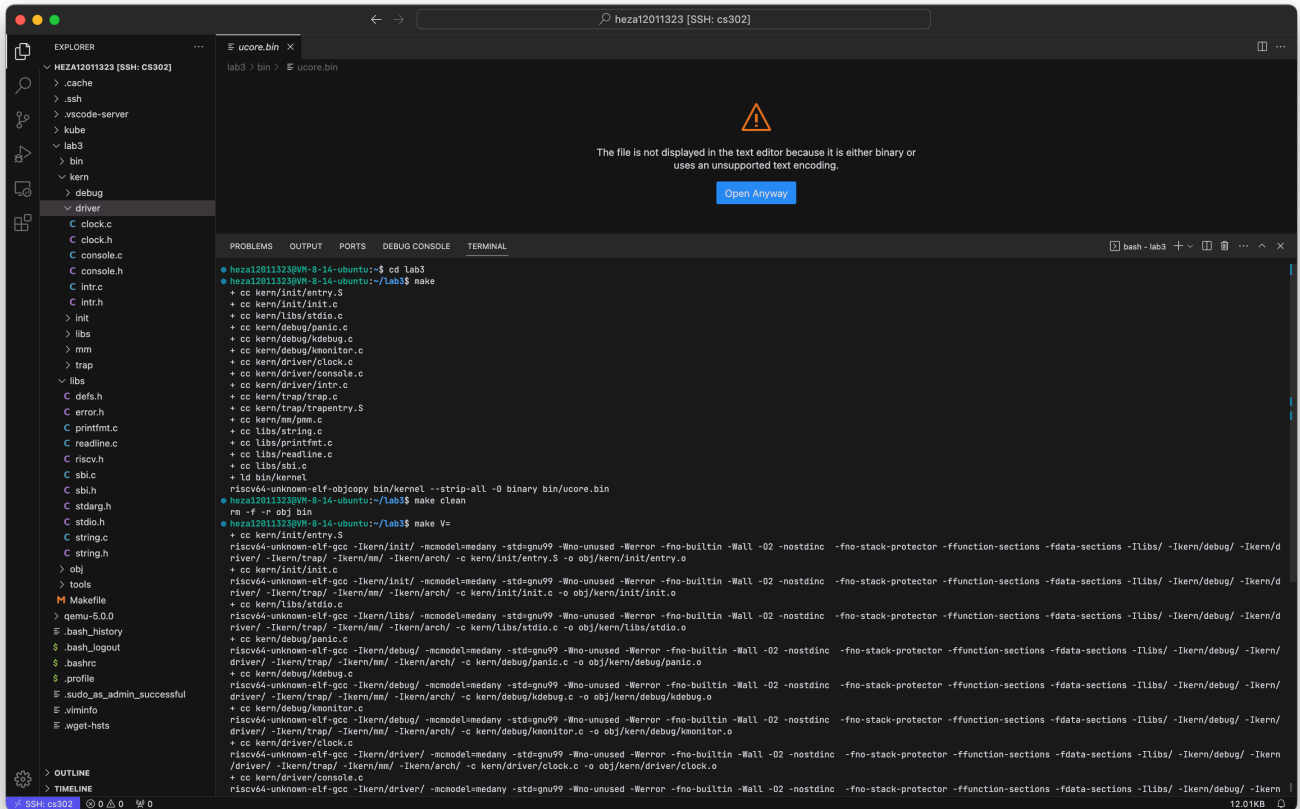


# CS302 Lab3 Report

何泽安 12011323

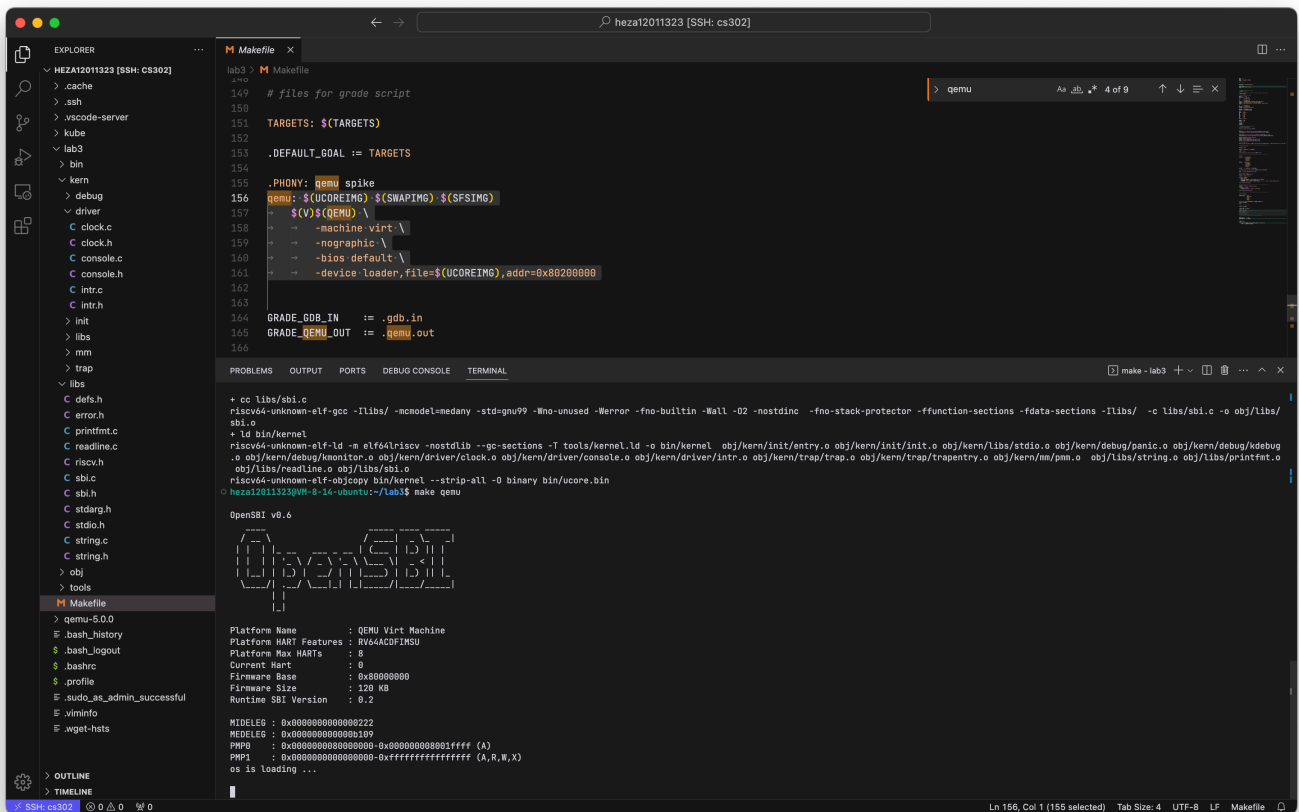
2023.3.1

1. First upload the related code (UCore) into server and extract it. Then directly use the `make` command to generate the virtual disk using the well config Makefile -- related binary files are generated under the `bin` folder.



The screenshot shows a VS Code editor window with the Explorer sidebar on the left displaying the project structure of 'HEZAI2011323 [SSH: CS302]'. The main editor area shows the 'ucore.bin' file, which is a binary file and thus not displayed in the text editor. Below the editor, the 'TERMINAL' tab is active, showing the execution of the 'make' command in the 'lab3' directory. The terminal output lists the source files being compiled, including kernel/init/entry.S, kernel/init/init.c, kernel/lib/string.c, kernel/lib/stdio.c, kernel/lib/panic.c, kernel/lib/kdebug.c, kernel/lib/kmonitor.c, kernel/lib/clock.c, kernel/lib/console.c, kernel/lib/intr.c, kernel/lib/mm.c, kernel/lib/trap.c, kernel/lib/sbi.c, kernel/lib/stdio.c, kernel/lib/printk.c, kernel/lib/readline.c, kernel/lib/sbi.c, kernel/lib/kernel.c, and kernel/lib/panic.c. The output also shows the compilation of these files into the 'bin' directory, resulting in the 'ucore.bin' file.

Then we use the `make qemu` target to start the VM.



In summary, only three commands are needed:

```
cd /path/to/src
make
make qemu
```

2. A **ELF** file (Executable and Linking Format) has a rather complex structure, including an ELF header, some rudendent debug info, etc, it contains the memory layout in the ELF header, which needs to be parsed by something like a OS, but without that, say, in OpenSBI firmware, the ELF file cannot be directly executed.

While **BIN** files are much simpler, with raw binary machine codes, it is the final way that the memory looks before the CPU starts executing it, however, since it cannot represent info in the ELF way, it may be much larger than ELF files. Also, bin files can be directly executed by firmwares.

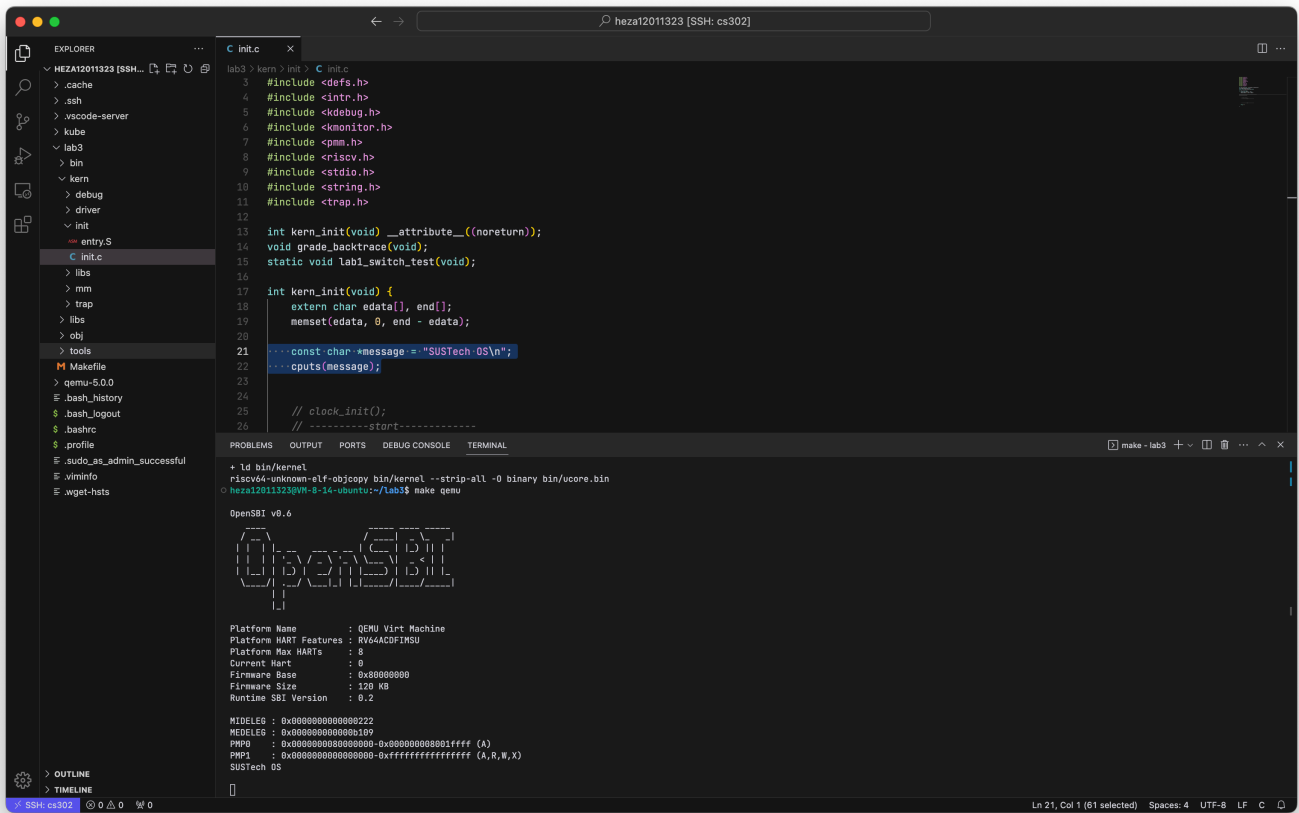
ELF is a cut-up/compressed version of that, which the CPU/MCU thus can't run directly. [StackOverflow](#)

Bin file is purely, binary files with bits and bytes which go and locate at the particular address.

[StackOverflow](#)

3. A *linker* helps linking input files (**.o**) into the output file (**elf**). During this step, the linker will face many sections in the input/output files, the *linker script* hereby helps descripting the way to map sections from input files to output files (merging the sections of the target files, relocate the starting/ending addresses of each section), it also rules the memory layout of these sections.

4. Find the file under `kern/init/init.c`, edit the c-string `message`, save, clean the last compiled files, recompile (`make`) and run vm (`make qemu`)



The screenshot shows a VS Code editor interface with the following components:

- EXPLORER:** A file tree on the left showing the project structure. The `init.c` file under `kern/init` is selected.
- Editor:** The main window displays the contents of `init.c`. The file includes headers like `<defs.h>`, `<intr.h>`, `<kdebug.h>`, `<monitor.h>`, `<pmm.h>`, `<riscv.h>`, `<stdio.h>`, `<string.h>`, and `<trap.h>`. It defines `kern_init` and `grade_backtrace`. A line of code is highlighted: `const char *message = "SUSTech OS\n";`.
- TERMINAL:** The bottom panel shows the output of the `make` command. It displays the compilation of `bin/kernel` and the execution of `make qemu`. The output shows the OpenSBI v0.6 logo and system information for the QEMU Virt Machine, including platform name, HART features, and memory layout.

5. We need to first find the header file `libs/stdio.h` and declare the function `int double_puts(const char **str*)`, then implement the function in `kern/libs/stdio.c`. After that we are able to call this function in `init.c`.

```
heza12011323 [SSH: cs302]

C init.c x
lab3 > kern > init > C init.c
11 #include <trap.h>
12
13 int kern_init(void) __attribute__((noreturn));
14 void grade_backtrace(void);
15 static void lab1_switch_test(void);
16
17 int kern_init(void) {
18     extern char edata[], end[];
19     memset(edata, 0, end - edata);
20
21     const char *message = "SUSTech OS\n";
22     cputs(message);
23
24     const char *iloveos = "ILOVEOS";
25     double_puts(iloveos);
26
27     // clock_init();
28     // -----start-----
29
30
31
32     // -----end-----
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100

C stdio.c x
lab3 > kern > libs > C stdio.c
55 char c;
56 while ((c = *str++) != '\0') {
57     cputch(c, &cnt);
58 }
59 cputch('\n', &cnt);
60 return cnt;
61
62 // add
63 int double_puts(const char *str) {
64     int cnt = 0;
65     char c;
66     while ((c = *str++) != '\0') {
67         cputch(c, &cnt);
68         cputch(c, &cnt);
69     }
70     cputch('\n', &cnt);
71     return cnt;
72 }
73
74 /* getchar - reads a single non-zero character from sta
75 int getchar(void) {
76     int c;
77     while ((c = cons_getc()) == 0) /* do nothing */;
78 }

C stdio.h x
lab3 > libs > C stdio.h
1 #ifndef __LIBS_STDIO_H__
2 #define __LIBS_STDIO_H__
3
4 #include <defs.h>
5 #include <stdarg.h>
6
7 /* kern/libs/stdio.c */
8 int cprintf(const char *fmt, ...);
9 int vprintf(const char *fmt, va_list ap);
10 void cputchar(int c);
11 int cputs(const char *str);
12 int double_puts(const char *str); // add
13 int getchar(void);
14
15 /* kern/libs/readline.c */
16 char *readline(const char *prompt);
17
18 /* libs/printf.c */
19 void printfmt(void (*putch)(int, void *), void *putdat,
20 void vprintfmt(void (*putch)(int, void *), void *putdat,
21 int snprintf(char *str, size_t size, const char *fmt, .
22 int vsnprintf(char *str, size_t size, const char *fmt,
23
24 #endif /* ! LIBS_STDIO_H__ */

PROBLEMS OUTPUT PORTS DEBUG CONSOLE TERMINAL
riscv64-unknown-elf-objcopy bin/kernel --strip-all -O binary bin/ucore.bin
heza12011323@vm-8-14-ubuntu:~/lab3$ make qemu

OpenSBI v0.6

Platform Name      : QEMU Virt Machine
Platform HART Features : RV64ACDFIMSU
Platform Max HARTs  : 8
Current Hart       : 0
Firmware Base      : 0x80000000
Firmware Size      : 128 KB
Runtime SBI Version : 0.2

MIDELEG : 0x00000000000000222
MEDELEG : 0x00000000000000109
PMP0    : 0x0000000000000000-0x0000000000001ffff (A)
PMP1    : 0x0000000000000000-0xffffffffffffff (A,R,W,X)
SUSTech OS

IILL00VVE00SS

SSH: cs302 0 0 0 0 Ln 22, Col 20 Spaces: 4 UTF-8 LF C
```