Exercise 1. NULL

Exercise 2.

- What the bios do should be to find and initialize devices as well as configure them, it set up something like interrupt descriptor table to make all the devices work correctly and findable by the OS. Specifically, it will initialize the DMA controller, which will enable the guest device to visit main memory. Then it will jump to the bootloader part.

Exercise 3.

- The processor start executing 32-bit code firstly at the instruction “ljmp $PROT_MODE_CSEG, $protcseg”. The previous instruction of it, which is “movl %eax, %cr0”, set the PE bit of cr0, and then the processor enter 32-bit protected mode.
- The last instruction of the boot loader is “call *0x10018”, the first instruction of the kernel is “movw $0x1234, 0x472”
- The boot loader will read the first page in the disk to memory, and use it as the ELF header, in the header the information of each segment include its offset and size will be included.

Exercise 4. NULL

Exercise 5.

- Before kernel is loaded, the region of memory is 0; after kernel is loaded, this region has some values.
- It’s the kernel binary that is loaded while booting.

Exercise 6.

- If we change the link address of the boot loader, it will still be loaded at 0x7c00, however, while compiling, the compiler think that the boot loader will begin at another value. So when there occurs a jump instruction which jump to a wrong place. Also, while loading GDTt, the wrong GDT will be loaded. All these things will cause the machine to hang somewhere. To handle the executing fail, the machine will reboot and start executing the first instruction of BIOS again.

Exercise 7.

- The first instruction after the new mapping is established that would fail to work properly if the ole mapping were still in place is “jmp *%eax”.

Exercise 8. NULL

Exercise 9.

- console.c provides an interface “cons_putc”, which will print a char in serial port, parallel port and vga. printf.c provide “putch” to output a char using functions in console.c and “cprintf” to print a formated string.

- If the screen is full, it will copy all lines in the screen one line forward, and clear the last line of the screen.
-
    - fmt points to "x %d, y %x, z %d\n", ap points to the arg list of x, y, z.
    - “cons_puts” will be called every time a char is printed, “va_arg” will be called when a arg is in need, “va_arg” will return the first item in the arg list and then move the arg cursor one step forward. “vcprintf” is only called once, and the args are “x %d, y %x, z %d\n” and a va_list of x, y and z.
- It output “He110 World”. The hex of 57616 is e110, x86 arch is little endian, so the integer i is stored just the same as “rld\0”. If x86 is big-endian, i should be set as 0x726c6400. There’s no need to change 57616.
- The value above 3 in the stark will be printed out. cprintf does not know how many args exactly this function call have, it just follow the program to execute the offset of the args, so it will print the value store above 3.
- Just let cprintf declare like this:
  int cprintf(..., const char *fmt) {...}
  However, this may need to modify the compiler.
  Or we can send the number of variables as the last argument, thus we can get it and pass the value to vcprintf

Exercise 10. NULL
Exercise 11. NULL

Exercise 12.

- In entry.S, the program setup ebp and use “movl $(bootstacktop),%esp” to setup the stack.
- The stack is located at KSTKSIZE in virtual memory.
- KSTKSIZE

Exercise 13.

- Before each call of test_backtrace, 2 32-bit words will be pushed into the stack. The previous one is the argument and the other one is return address.

Exercise 15.

- Stabs are from the .stab segment. The compiler will store these information in a specific format in this segment while compiling the kernel. Then in kdebug.c stab_binsearch will search for these information and find the value in need.

Exercise 16. NULL
Exercise 17. NULL