CS 444/544 OS II
Lab Tutorial #2 (part 1)

Booting and GDB Practice

Acknowledgement: Slides drawn heavily from Yeongjin Jiang
Lab Setup Check
Contents

• Following the Booting Sequence

• Exercise 3-6

• Coding Style
Exercise 3

Set a breakpoint at address 0x7c00, which is where the boot sector will be loaded. Continue execution until that breakpoint. Trace through the code in `boot/boot.S`, using the source code and the disassembly file `obj/boot/boot.asm` to keep track of where you are. Also use the x/i command in GDB to disassemble sequences of instructions in the boot loader, and compare the original boot loader source code with both the disassembly in `obj/boot/boot.asm` and GDB.
Exercise 3: How?

• Use tmux, open boot/boot.S and gdb at the same time.

Use ni, si, breakpoint
Exercise 3: Enabling Protected Mode

PROT_MODE_CSEG 0x8

Bootmain() is in boot/main.c

```
.lgdt gdtdesc
movl %cr0, %eax
orl $CR0_PE_ON, %eax
movl %eax, %cr0

# Jump to next instruction, but in 32-bit code segment.
# Switches processor into 32-bit mode.
ljmp PROT_MODE_CSEG, protcseg

.code32 # Assemble for 32-bit mode
protcseg:

# Set up the protected-mode data segment registers
movw PROT_MODE_DSEG, %ax # Our data segment selector
movw %ax, %ds # --> DS: Data Segment
movw %ax, %es # --> ES: Extra Segment
movw %ax, %fs # --> FS
movw %ax, %gs # --> GS
movw %ax, %ss # --> SS: Stack Segment

# Set up the stack pointer and call into C.
movl $start, %esp
call bootmain
```
Exercise 3: bootmain

In `boot/main.c`
```c
# Set up the stack pointer and call into C.
movl $start, %esp
call bootmain

void bootmain(void)
{
    struct Proghdr *ph, *eph;

    // read 1st page off disk
    readseg((uint32_t) ELFHDR, SECTSIZE*8, 0);

    // is this a valid ELF?
    if (ELFHDR->e_magic != ELF_MAGIC)
        goto bad;

    // load each program segment (ignores ph flags)
    ph = (struct Proghdr *) (uint8_t *) ELFHDR + ELFHDR->e_phoff;
    eph = ph + ELFHDR->e_phnum;
    for (; ph < eph; ph++)
        // p_pa is the load address of this segment (as well
        // as the physical address)
        readseg(ph->p_pa, ph->p_memsz, ph->p_offset);

    // call the entry point from the ELF header
    // note: does not return!
    ((void (*)(void)) (ELFHDR->e_entry))();
}
```
Exercise 4-6

- Ex 4: Understand why pointer.c works like that and read about ELF header and the ELF file.

- Ex 5: Use si, ni to follow the instructions after changing 0x7c00 to others, e.g., 0x6b00 or something else.

```
$(OBJDIR)/boot/boot: $(BOOT_OBJS)
  @echo + ld boot/boot
  $(V)$LD $(LDFLAGS) -N -e start -Ttext 0x7C00 -o @.out $^  
  $(V)$OBJDUMP -S @.out >@@.asm
  $(V)$OBJCOPY -S -O binary -j .text @.out @
  $(V)perl boot/sign.pl $(OBJDIR)/boot/boot
```

- Ex 6: practice gdb commands
GDB Command for Reading Memory

- **x/100wx** [address or register]
  - Examine
  - **100** values
  - sized as word (w, 4 bytes)
    - b – byte
    - g – 8 bytes
  - **In hexadecimal (x)**
    - d - decimal
Coding Convention (CODING)

• No space after a function name in a call
  • cprintf("asdf")  GOOD
  • cprintf ("asdf")  NO

• One space after if/for/while/switch
  • if (a == 1) {  GOOD
  • if(a==1) {     NO

• function_and_variable_names_look_like_this
  • NoCamelCase

• Macros are ALL UPPERCASE
  • e.g., SEG()
Coding Convention (CODING)

• Pointer types includes a space before *
  • (uint32_t *) GOOD
  • (uint32_t*) NO

• Use ‘//’ for your comment
  • All imported comments are /**/ , so we can distinguish yours from those
  • FYI, Linux Kernel uses /**/...

• Function with no args
  • f(void), not f();
Coding Convention (CODING)

• Function definition
  • Insert newline between the return type and function name
  • This will make finding function definition easy
  • E.g., find the definition of mon_kerninfo would be:
  • ‘^mon_kerninfo’ in regexp.

```c
int mon_kerninfo(int argc, char **argv, struct Trapframe *tf)
{
```

```bash
os2 ~/cs444/s21/os2-lab1-Rogersyp 106% grep -nr ^mon_kerninfo
kern/monitor.c:43:mon_kerninfo(int argc, char **argv, struct Trapframe *tf)
os2 ~/cs444/s21/os2-lab1-Rogersyp 107%
```