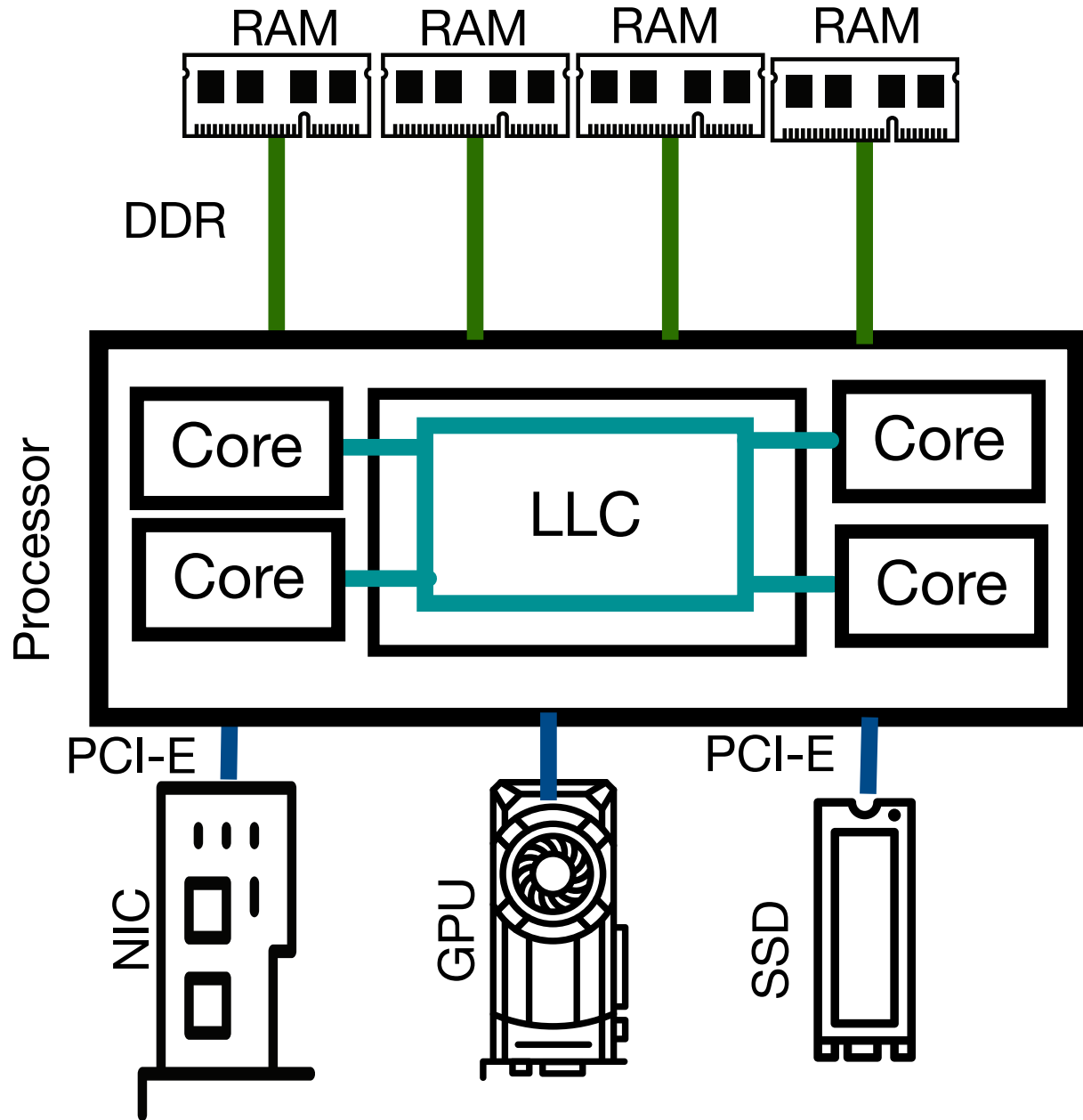


Machine



Mar 26, 24 22:44

handout10-2.txt

Page 1/5

```

1 CS 202, Spring 2023
2 Handout 9 (Class 16)
3
4 1. Example use of I/O instructions: boot loader
5
6     Below is the WeensyOS boot loader
7
8     It may be helpful to understand the overall picture
9
10    This code demonstrates I/O, specifically with the disk: the
11    bootloader reads in the kernel from the disk.
12
13    See the functions boot_waitdisk() and boot_readsect(). Compare to Figures 36
14    .5 and 36.6 in OSTEP.
15
16    /* boot.c */
17    #include "x86-64.h"
18    #include "elf.h"
19
20    // boot.c
21    //
22    // WeensyOS boot loader. Loads the kernel at address 0x40000 from
23    // the first IDE hard disk.
24    //
25    // A BOOT LOADER is a tiny program that loads an operating system into
26    // memory. It has to be tiny because it can contain no more than 512 bytes
27    // of instructions: it is stored in the disk's first 512-byte sector.
28    //
29    // When the CPU boots it loads the BIOS into memory and executes it. The
30    // BIOS initializes devices and CPU state, reads the first 512-byte sector of
31    // the boot device (hard drive) into memory at address 0x7C00, and jumps to
32    // that address.
33    //
34    // The boot loader is contained in bootstart.S and boot.c. Control starts
35    // in bootstart.S, which initializes the CPU and sets up a stack, then
36    // transfers here. This code reads in the kernel image and calls the
37    // kernel.
38    //
39    // The main kernel is stored as an ELF executable image starting in the
40    // disk's sector 1.
41
42    #define SECTORSIZE      512
43    #define ELFHDR         ((elf_header*) 0x10000) // scratch space
44
45    void boot(void) __attribute__((noreturn));
46    static void boot_readsect(uintptr_t dst, uint32_t src_sect);
47    static void boot_readseg(uintptr_t dst, uint32_t src_sect,
48                             size_t filesz, size_t memsz);
49
50    // boot
51    // Load the kernel and jump to it.
52    void boot(void) {
53        // read 1st page off disk (should include programs as well as header)
54        // and check validity
55        boot_readseg((uintptr_t) ELFHDR, 1, PAGESIZE, PAGESIZE);
56        while (ELFHDR->e_magic != ELF_MAGIC) {
57            /* do nothing */
58        }
59
60        // load each program segment
61        elf_program* ph = (elf_program*) ((uint8_t*) ELFHDR + ELFHDR->e_phoff);
62        elf_program* eph = ph + ELFHDR->e_phnum;
63        for (; ph < eph; ++ph) {
64            boot_readseg(ph->p_va, ph->p_offset / SECTORSIZE + 1,
65                        ph->p_filesz, ph->p_memsz);
66        }
67
68        // jump to the kernel
69        typedef void (*kernel_entry_t)(void) __attribute__((noreturn));
70        kernel_entry_t kernel_entry = (kernel_entry_t) ELFHDR->e_entry;
71        kernel_entry();
72    }

```

Tuesday March 26, 2024

handout10-2.txt

Mar 26, 24 22:44

handout10-2.txt

Page 2/5

```

73
74
75 // boot_readseg(dst, src_sect, filesz, memsz)
76 // Load an ELF segment at virtual address 'dst' from the IDE disk's sector
77 // 'src_sect'. Copies 'filesz' bytes into memory at 'dst' from sectors
78 // 'src_sect' and up, then clears memory in the range
79 // '[dst+filesz, dst+memsz)'.
80 static void boot_readseg(uintptr_t ptr, uint32_t src_sect,
81                          size_t filesz, size_t memsz) {
82     uintptr_t end_ptr = ptr + filesz;
83     memsz += ptr;
84
85     // round down to sector boundary
86     ptr &= ~(SECTORSIZE - 1);
87
88     // read sectors
89     for (; ptr < end_ptr; ptr += SECTORSIZE, ++src_sect) {
90         boot_readsect(ptr, src_sect);
91     }
92
93     // clear bss segment
94     for (; end_ptr < memsz; ++end_ptr) {
95         *(uint8_t*) end_ptr = 0;
96     }
97 }
98
99 // boot_waitdisk
100 // Wait for the disk to be ready.
101 static void boot_waitdisk(void) {
102     // Wait until the ATA status register says ready (0x40 is on)
103     // & not busy (0x80 is off)
104     while ((inb(0x1F7) & 0xC0) != 0x40) {
105         /* do nothing */
106     }
107 }
108
109 // boot_readsect(dst, src_sect)
110 // Read disk sector number 'src_sect' into address 'dst'.
111 static void boot_readsect(uintptr_t dst, uint32_t src_sect) {
112     // programmed I/O for "read sector"
113     boot_waitdisk();
114     outb(0x1F2, 1); // send 'count = 1' as an ATA argument
115     outb(0x1F3, src_sect); // send 'src_sect', the sector number
116     outb(0x1F4, src_sect >> 8);
117     outb(0x1F5, src_sect >> 16);
118     outb(0x1F6, (src_sect >> 24) | 0xE0);
119     outb(0x1F7, 0x20); // send the command: 0x20 = read sectors
120
121     // then move the data into memory
122     boot_waitdisk();
123     insl(0x1F0, (void*) dst, SECTORSIZE/4); // read 128 words from the disk
124 }
125
126
127
128

```

1/3

Mar 26, 24 22:44

handout10-2.txt

Page 3/5

```

129 2. Two more examples of I/O instructions
130
131     (a) Reading keyboard input
132
133     The code below is an excerpt from WeensyOS's k-hardware.c
134
135     This reads a character typed at the keyboard (which shows up on the
136     "keyboard data port" (KEYBOARD_DATAREG)).
137
138     /* Excerpt from WeensyOS x86-64.h */
139     // Keyboard programmed I/O
140     #define KEYBOARD_STATUSREG    0x64
141     #define KEYBOARD_STATUS_READY 0x01
142     #define KEYBOARD_DATAREG     0x60
143
144     int keyboard_readc(void) {
145         static uint8_t modifiers;
146         static uint8_t last_escape;
147
148         if ((inb(KEYBOARD_STATUSREG) & KEYBOARD_STATUS_READY) == 0) {
149             return -1;
150         }
151
152         uint8_t data = inb(KEYBOARD_DATAREG);
153         uint8_t escape = last_escape;
154         last_escape = 0;
155
156         if (data == 0xE0) { // mode shift
157             last_escape = 0x80;
158             return 0;
159         } else if (data & 0x80) { // key release: matters only for modifier ke
160             ys
161                 int ch = keymap[(data & 0x7F) | escape];
162                 if (ch >= KEY_SHIFT && ch < KEY_CAPSLOCK) {
163                     modifiers &= ~(1 << (ch - KEY_SHIFT));
164                 }
165                 return 0;
166             }
167
168             int ch = (unsigned char) keymap[data | escape];
169
170             if (ch >= 'a' && ch <= 'z') {
171                 if (modifiers & MOD_CONTROL) {
172                     ch -= 0x60;
173                 } else if (!(modifiers & MOD_SHIFT) != !(modifiers & MOD_CAPSLOCK))
174                 {
175                     ch -= 0x20;
176                 }
177             } else if (ch >= KEY_CAPSLOCK) {
178                 modifiers ^= 1 << (ch - KEY_SHIFT);
179                 ch = 0;
180             } else if (ch >= KEY_SHIFT) {
181                 modifiers |= 1 << (ch - KEY_SHIFT);
182                 ch = 0;
183             } else if (ch >= CKEY(0) && ch <= CKEY(21)) {
184                 ch = complex_keymap[ch - CKEY(0)].map[modifiers & 3];
185             } else if (ch < 0x80 && (modifiers & MOD_CONTROL)) {
186                 ch = 0;
187             }
188
189             return ch;
190         }

```

Mar 26, 24 22:44

handout10-2.txt

Page 4/5

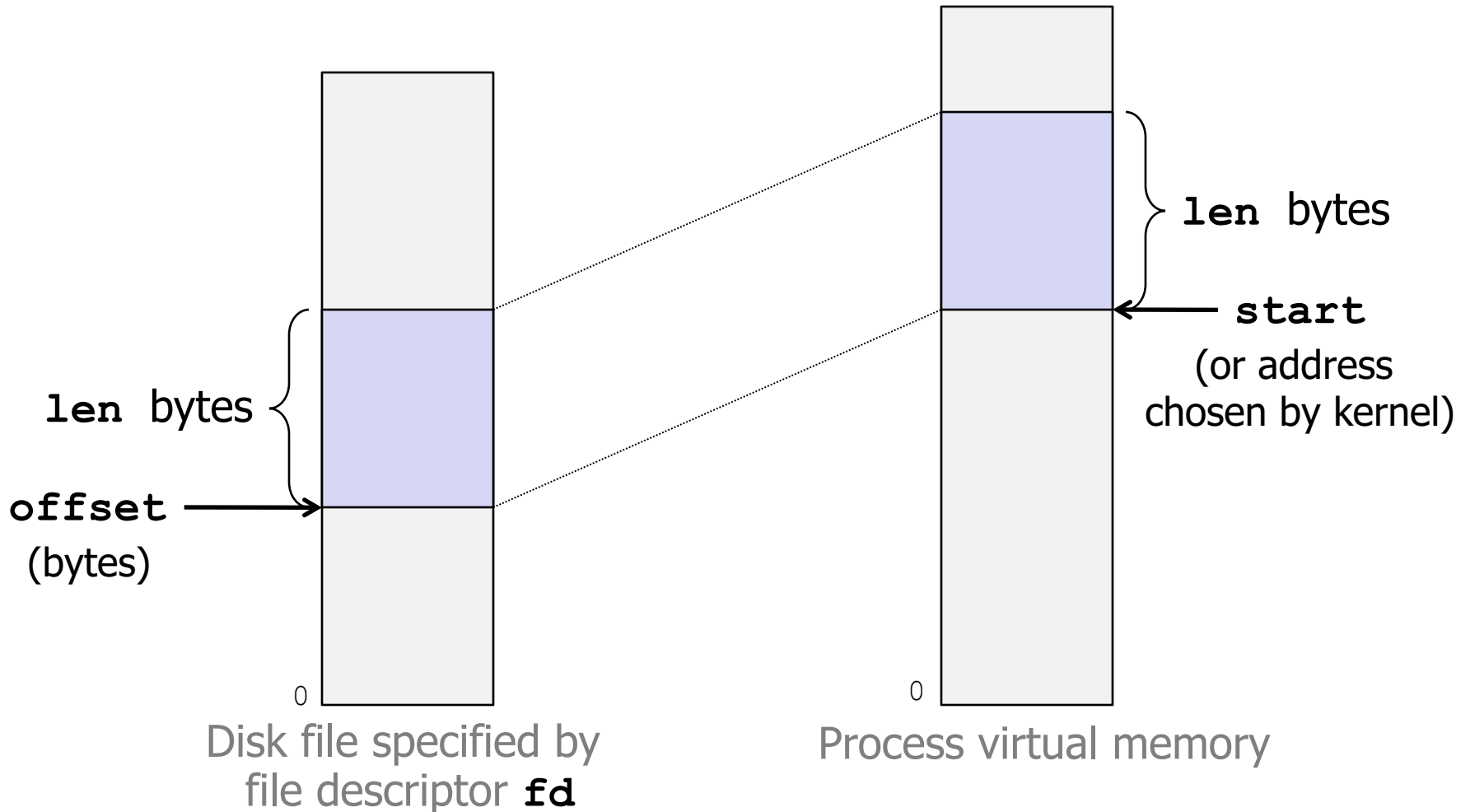
```

190
191     (b) Setting the cursor position
192
193     The code below is also excerpted from WeensyOS's k-hardware.c. It
194     uses I/O instructions to set a blinking cursor somewhere on a 25 x 80
195     screen.
196
197     // console_show_cursor(cpos)
198     //     Move the console cursor to position 'cpos', which should be between 0
199     //     and 80 * 25.
200
201     void console_show_cursor(int cpos) {
202         if (cpos < 0 || cpos > CONSOLE_ROWS * CONSOLE_COLUMNS) {
203             cpos = 0;
204         }
205         outb(0x3D4, 14); // Command 14 = upper byte of position
206         outb(0x3D5, cpos / 256);
207         outb(0x3D4, 15); // Command 15 = lower byte of position
208         outb(0x3D5, cpos % 256);
209     }
210
211
212
213
214

```


User-Level Memory Mapping

```
void *mmap(void *start, int len,  
           int prot, int flags, int fd, int offset)
```



Mar 26, 24 22:43

copyout.c

Page 1/1

```
1 #include <fcntl.h>
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include <sys/mman.h>
5 #include <sys/stat.h>
6 #include <sys/types.h>
7 #include <unistd.h>
8
9 void mmapcopy(int fd, int size);
10
11 int main(int argc, char **argv) {
12     struct stat stat;
13     int fd;
14
15     /* Check for required cmd line arg */
16     if (argc != 2) {
17         printf("usage: %s <filename>\n", argv[0]);
18         exit(0);
19     }
20
21     /* Copy input file to stdout */
22     if ((fd = open(argv[1], O_RDONLY, 0)) < 0)
23         perror("open");
24
25     fstat(fd, &stat);
26     mmapcopy(fd, stat.st_size);
27
28     close(fd);
29
30     return 0;
31 }
32
33 void mmapcopy(int fd, int size) {
34     /* Ptr to memory mapped area */
35     char *bufp;
36
37     bufp = mmap(NULL, size, PROT_READ, MAP_PRIVATE, fd, 0);
38
39     write(STDOUT_FILENO, bufp, size);
40
41     return;
42 }
43 }
```