

```

1 CS5600, Handout week11.a
2
3 /* file: mmap.c */
4
5 #include <fcntl.h>
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <sys/mman.h>
9 #include <sys/stat.h>
10 #include <sys/types.h>
11 #include <unistd.h>
12
13 void mmapwrite(int fd, int size);
14 void normalwrite(int fd, int size);
15
16 int main(int argc, char **argv) {
17     struct stat stat;
18     int fd;
19
20     if (argc != 2) { // Check for required cmd line arg
21         printf("usage: %s <filename>\n", argv[0]);
22         exit(0);
23     }
24
25     /* Copy input file to stdout */
26     if ((fd = open(argv[1], O_RDONLY, 0)) < 0)
27         perror("open");
28
29     fstat(fd, &stat);
30
31     // option 1
32     mmapwrite(fd, stat.st_size);
33
34     /* // option 2
35      * normalwrite(fd, stat.st_size);
36      */
37
38     close(fd);
39
40     return 0;
41 }
42
43 void mmapwrite(int fd, int size) {
44
45     /* Ptr to memory mapped area */
46     char *bufp;
47
48     bufp = mmap(NULL, size, PROT_READ, MAP_PRIVATE, fd, 0);
49
50     write(STDOUT_FILENO, bufp, size);
51
52     return;
53 }
54
55
56 void normalwrite(int fd, int size) {
57
58     char *buf = malloc(size);
59
60     read(fd, buf, size);
61
62     write(STDOUT_FILENO, buf, size);
63
64     return;
65 }

```

## Question:

Which runs faster, option 1 or option 2? by how much?

## Exercise:

Try to run both options by yourself:

```

$ cat /dev/urandom | head -c 100000000 > 1G.file
$ make mmap
$ time ./mmap 1G.file > /dev/null

```

```

$ vim mmap.c
// switch to option 2
$ make mmap
$ time ./mmap 1G.file > /dev/null

```

```

1 CS5600 Week11.b
2
3 1. Two examples of I/O instructions
4
5 (a) Reading keyboard input
6
7 The code below is an excerpt from WeensyOS.
8 (details in PS/2 controller: https://wiki.osdev.org/%22804%22\_PS/2\_Controller)
9 This reads a character typed at the keyboard (which shows up on the
10 "keyboard data port" (KEYBOARD_DATAREG)).
11
12 /* Excerpt from WeensyOS x86-64.h and k-hardware.cc */
13 // Keyboard programmed I/O
14 #define KEYBOARD_STATUSREG 0x64
15 #define KEYBOARD_STATUS_READY 0x01
16 #define KEYBOARD_DATAREG 0x60
17
18 int keyboard_read() {
19     static uint8_t modifiers;
20     static uint8_t last_escape;
21
22     if ((inb(KEYBOARD_STATUSREG) & KEYBOARD_STATUS_READY) == 0) {
23         return -1;
24     }
25
26     uint8_t data = inb(KEYBOARD_DATAREG);
27     uint8_t escape = last_escape;
28     last_escape = 0;
29
30     if (data == 0xE0) { // mode shift
31         last_escape = 0x80;
32         return 0;
33     } else if (data & 0x80) { // key release: matters only
34         // for modifier keys
35         int ch = keymap[(data & 0x7F) | escape];
36         if (ch >= KEY_SHIFT && ch < KEY_CAPSLOCK) {
37             modifiers &= ~(1 << (ch - KEY_SHIFT));
38         }
39         return 0;
40     }
41
42     int ch = (unsigned char) keymap[data | escape];
43
44     if (ch >= 'a' && ch <= 'z') {
45         if (modifiers & MOD_CONTROL) {
46             ch -= 0x60;
47         } else if (!(modifiers & MOD_SHIFT) != \
48                 !(modifiers & MOD_CAPSLOCK)) {
49             ch -= 0x20;
50         }
51     } else if (ch >= KEY_CAPSLOCK) {
52         modifiers ^= 1 << (ch - KEY_SHIFT);
53         ch = 0;
54     } else if (ch >= KEY_SHIFT) {
55         modifiers |= 1 << (ch - KEY_SHIFT);
56         ch = 0;
57     } else if (ch >= CKEY(0) && ch <= CKEY(21)) {
58         ch = complex_keymap[ch - CKEY(0)].map[modifiers & 3];
59     } else if (ch < 0x80 && (modifiers & MOD_CONTROL)) {
60         ch = 0;
61     }
62
63     return ch;
64 }
65

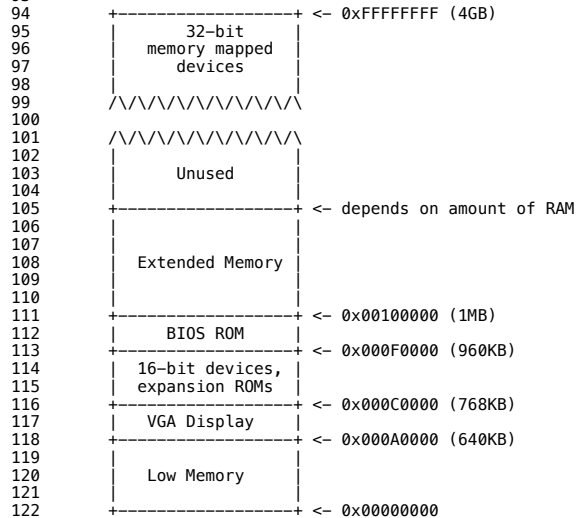
```

```

66
67 (b) Setting the cursor position
68
69 The code below is also excerpted from WeensyOS. It uses I/O
70 instructions to set a blinking cursor. To set the cursor to
71 the upper left of the screen, run: console_show_cursor(0)
72
73 // console_show_cursor(cpos)
74 // Move the console cursor to position 'cpos',
75 // which should be between 0 and 80 * 25.
76
77 void console_show_cursor(int cpos) {
78     if (cpos < 0 || cpos > CONSOLE_ROWS * CONSOLE_COLUMNS)
79         cpos = 0;
80
81     outb(0x3D4, 14); // Command 14 = upper byte of position
82     outb(0x3D5, cpos / 256); // upper byte (256 = 2^8)
83     outb(0x3D4, 15); // Command 15 = lower byte of position
84     outb(0x3D5, cpos % 256); // lower byte
85
86 }
87
88 // if interested, see details: https://wiki.osdev.org/Text\_Mode\_Cursor

```

89  
90 2. Memory-mapped I/O  
91  
92 (a) Here is a 32-bit PC's physical memory map:  
93



123 [Credit to Frans Kaashoek, Robert Morris, and  
124 Nickolai Zeldovich for this picture]  
125  
126

```

127
128 (b) Loads and stores to the device memory "go to hardware".
129
130 Here is an excerpt of the console printing code from WeensyOS.
131
132
133 /* Compare the address below to the map in panel 2(a). */
134 PROVIDE(console = 0xB8000);
135
136 This is an excerpt about printing; notice how it uses the address
137 "console":
138
139 /*
140 * prints a character to the console at the specified
141 * cursor position in the specified color.
142 * Question: what is going on in the check
143 * if (c == '\n')
144 * ?
145 * Hint: '\n' is "newline" (the user pressed enter).
146 */
147 static void console_putc(printer* p, unsigned char c, int color) {
148     console_printer* cp = (console_printer*) p;
149     if (cp->cursor >= console + CONSOLE_ROWS * CONSOLE_COLUMNS) {
150         cp->cursor = console;
151     }
152     if (c == '\n') {
153         int pos = (cp->cursor - console) % 80;
154         for (; pos != 80; pos++) {
155             *cp->cursor++ = ' ' | color;
156         }
157     } else {
158         *cp->cursor++ = c | color;
159     }
160 }

```