

Jan 26, 10 13:47

I03-handout.txt

Page 1/3

```

1 Handout for CS 372H
2 Lecture 3
3 26 January 2010
4
5 1. Fun with function pointers
6
7     /* countas.c */
8
9     #include <stdio.h>
10    /*
11     * define new type fptr: "pointer to a function whose input is a char* and
12     * whose return is an int"
13     */
14    typedef int (*fptr)(char*);
15
16    int return_0(char* p)
17    {
18        return 0;
19    }
20
21    int funfun(char* p)
22    {
23        /*
24         * on the survey, it was:
25         * int (*func_ptrs[])(char*) = {return_0, funfun};
26         */
27
28        fptr func_ptrs[] = {return_0, funfun};
29
30        return (*p == 'a') + (*func_ptrs[*p != 0])(p+1);
31    }
32
33    int
34    main(int argc, char** argv)
35    {
36        printf("%d\n", funfun(argv[1]));
37        return 0;
38    }
39
40    NOTES
41
42    --to compile this:
43
44        % gcc -g -Wall -o countas countas.c
45
46    --to get an assembly dump:
47
48        % gcc -S countas.c
49
50    --no conditional branches! (so no looping, if statements, etc.)
51
52    --(*p != 0) gets compiled into:
53        testb  %al, %al
54        setne  %al
55        movzbl %al, %eax
56
57    --then %eax is used to help index into the table of function
58    pointers
59
60    --common design pattern: use a value to index into a table of
61    function pointers. (the table is a list of addresses.)
62
63    --even happens at hardware/software interface. for example, when an
64    interrupt happens, the x86 chooses which interrupt handler to
65    execute by using the value of the interrupt as an index into a table
66    of interrupt handlers.
67

```

Jan 26, 10 13:47

I03-handout.txt

Page 2/3

```

68 2. Example for gcc calling conventions
69
70 Here is the C code:
71     int main(void) { return f(8)+1; }
72     int f(int x) { return g(x); }
73     int g(int x) { return x+3; }
74
75 The assembly code:
76
77     _main:
78         prologue
79
80         pushl %ebp
81         movl %esp, %ebp
82
83         body
84         pushl $8
85         call _f
86         addl $1, %eax
87
88         epilogue
89         movl %ebp, %esp
90         popl %ebp
91         ret
92
93     _f:
94         prologue
95
96         pushl %ebp
97         movl %esp, %ebp
98
99         body
100        pushl 8(%esp)
101        call _g
102
103        epilogue
104        movl %ebp, %esp
105        popl %ebp
106        ret
107
108    <small version of _g>:
109        movl 4(%esp), %eax
110        addl $3, %eax
111        ret
112
113    <longer version of _g>:
114        prologue
115        pushl %ebp
116        movl %esp, %ebp
117
118        save %ebx
119        pushl %ebx
120
121        body
122        movl 8(%ebp), %ebx
123        addl $3, %ebx
124        movl %ebx, %eax
125
126        restore %ebx
127        popl %ebx
128
129        epilogue
130        movl %ebp, %esp
131        popl %ebp
132        ret
133

```

```

134 3. pseudocode for Bochs CPU simulation
135
136     for (;;) {
137         read_instruction();
138         switch (decode_instruction_opcode()) {
139             case OPCODE_ADD:
140                 int src = decode_src_reg();
141                 int dst = decode_dst_reg();
142                 regs[dst] = regs[dst] + regs[src];
143                 break;
144             case OPCODE_SUB:
145                 int src = decode_src_reg();
146                 int dst = decode_dst_reg();
147                 regs[dst] = regs[dst] - regs[src];
148                 break;
149             ...
150         }
151         eip += instruction_length;
152     }
153
154 4. simulate PC's physical memory map
155
156     #define KB      1024
157     #define MB      1024*1024
158
159     #define LOW_MEMORY  640*KB
160     #define EXT_MEMORY  10*MB
161
162     uint8_t low_mem[LOW_MEMORY];
163     uint8_t ext_mem[EXT_MEMORY];
164     uint8_t bios_rom[64*KB];
165
166     uint8_t read_byte(uint32_t phys_addr) {
167         if (phys_addr < LOW_MEMORY)
168             return low_mem[phys_addr];
169         else if (phys_addr >= 960*KB && phys_addr < 1*MB)
170             return bios_rom[phys_addr - 960*KB];
171         else if (phys_addr >= 1*MB && phys_addr < 1*MB+EXT_MEMORY) {
172             return ext_mem[phys_addr-1*MB];
173         }
174         else ...
175     }
176
177     void write_byte(uint32_t phys_addr, uint8_t val) {
178         if (phys_addr < LOW_MEMORY)
179             low_mem[phys_addr] = val;
180         else if (phys_addr >= 960*KB && phys_addr < 1*MB)
181             /* ignore attempted write to ROM! */
182         else if (phys_addr >= 1*MB && phys_addr < 1*MB+EXT_MEMORY) {
183             ext_mem[phys_addr-1*MB] = val;
184         }
185         else ...
186     }
187
188     [Credit to Frans Kaashoek, Robert Morris, and Nickolai Zeldovich for 2-4
189     above]

```