Mitjos9-12 实验

1511489 李东闻

一、实验目的

Exercise 9. Determine where the kernel initializes its stack, and exactly where in memory its stack is located. How does the kernel reserve space for its stack? And at which "end" of this reserved area is the stack pointer initialized to point to?

Exercise 10. To become familiar with the C calling conventions on the x86, find the address of the `test_backtrace` function in `obj/kern/kernel.asm`, set a breakpoint there, and examine what happens each time it gets called after the kernel starts. How many 32-bit words does each recursive nesting level of `test_backtrace` push on the stack, and what are those words?

Note that, for this exercise to work properly, you should be using the patched version of QEMU available on the tools page or on Athena. Otherwise, you'll have to manually translate all breakpoint and memory addresses to linear addresses.

Exercise 11. Implement the backtrace function as specified above. Use the same format as in the example, since otherwise the grading script will be confused. When you think you have it working right, run `make grade` to see if its output conforms to what our grading script expects, and fix it if it doesn't. *After* you have handed in your Lab 1 code, you are welcome to change the output format of the backtrace function any way you like.

If you use `read_ebp()`, note that GCC may generate "optimized" code that calls `read_ebp()` *before* `mon_backtrace()`'s function prologue, which results in an incomplete stack trace (the stack frame of the most recent function call is missing). While we have tried to disable optimizations that cause this reordering, you may want to examine the assembly of `mon_backtrace()` and make sure the call to `read_ebp()` is happening after the function prologue.

Exercise 12. Modify your stack backtrace function to display, for each `eip`, the function name, source file name, and line number corresponding to that `eip`.

In `debuginfo_eip`, where do `__STAB_*` come from? This question has a long answer; to help you to discover the answer, here are some things you might want to do:

- look in the file `kern/kernel.ld` for `__STAB_*`
- run `i386-jos-elf-objdump -h obj/kern/kernel`
- run `i386-jos-elf-objdump -G obj/kern/kernel`
- run `i386-jos-elf-gcc -pipe -nostdinc -O2 -fno-builtin -I. -MD -Wall -Wno-format -DJOS_KERNEL -gstabs -c -S kern/init.c`, and look at init.s.
- see if the bootloader loads the symbol table in memory as part of loading the kernel binary

Complete the implementation of `debuginfo_eip` by inserting the call to `stab_binsearch` to find the line number for an address.

二、实验内容

Exercise9:

在 entry.S 中可以看到

```
relocated:

        # Clear the frame pointer register (EBP)
        # so that once we get into debugging C code,
        # stack backtraces will be terminated properly.
        movl    $0x0,%ebp                       # nuke frame point

        # Set the stack pointer
        movl    $(bootstacktop),%esp

        # now to C code
        call    i386_init
```

内核初始化将寄存器%ebp 初始为 0，%esp 初始化为 bootstacktop
检查 bootstacktop 位置数据

```
.data
###################################################################
# boot stack
###################################################################
        .p2align        PGSHIFT         # force page alignment
        .globl          bootstack
bootstack:
        .space          KSTKSIZE
        .globl          bootstacktop
bootstacktop:
```

```
#define KSTKSIZE        (8*PGSIZE)
```

栈的空间定义在.data 段，栈共分为两段 KSTKSIZE 和 bootstacktop。KSTKSIZE 大小定义在 inc/memlayyout.h 中，为 8*PGSIZE；bootstacktop 为栈底，由于栈向低地址位置生长，所以最高位置就是栈底，将会赋值给%esp。

Exercise10:

观察 obj/kern/kernel.asm

```
 74 void
 75 test_backtrace(int x)
 76 {
 77 f0100040:       55                      push   %ebp
 78 f0100041:       89 e5                   mov    %esp,%ebp
 79 f0100043:       53                      push   %ebx
 80 f0100044:       83 ec 0c                sub    $0xc,%esp
 81 f0100047:       8b 5d 08                mov    0x8(%ebp),%ebx
 82         cprintf("entering test_backtrace %d\n", x);
 83 f010004a:       53                      push   %ebx
 84 f010004b:       68 40 19 10 f0          push   $0xf0101940
 85 f0100050:       e8 7c 09 00 00          call   f01009d1 <cprintf>
 86         if (x > 0)
 87 f0100055:       83 c4 10                add    $0x10,%esp
 88 f0100058:       85 db                   test   %ebx,%ebx
 89 f010005a:       7e 11                   jle    f010006d <test_backtrace+
0x2d>
 90                 test_backtrace(x-1);
 91 f010005c:       83 ec 0c                sub    $0xc,%esp
 92 f010005f:       8d 43 ff                lea    -0x1(%ebx),%eax
 93 f0100062:       50                      push   %eax
 94 f0100063:       e8 d8 ff ff ff          call   f0100040 <test_backtrace>
 95 f0100068:       83 c4 10                add    $0x10,%esp
```

```
 94 f0100063:       e8 d8 ff ff ff          call   f0100040 <test_backtrace>
 95 f0100068:       83 c4 10                add    $0x10,%esp
 96 f010006b:       eb 11                   jmp    f010007e <test_backtrace+
0x3e>
 97         else
 98                 mon_backtrace(0, 0, 0);
 99 f010006d:       83 ec 04                sub    $0x4,%esp
100 f0100070:       6a 00                   push   $0x0
101 f0100072:       6a 00                   push   $0x0
102 f0100074:       6a 00                   push   $0x0
103 f0100076:       e8 26 07 00 00          call   f01007a1 <mon_backtrace>
104 f010007b:       83 c4 10                add    $0x10,%esp
105         cprintf("leaving test_backtrace %d\n", x);
106 f010007e:       83 ec 08                sub    $0x8,%esp
107 f0100081:       53                      push   %ebx
108 f0100082:       68 5c 19 10 f0          push   $0xf010195c
109 f0100087:       e8 45 09 00 00          call   f01009d1 <cprintf>
110 }
111 f010008c:       83 c4 10                add    $0x10,%esp
112 f010008f:       8b 5d fc                mov    -0x4(%ebp),%ebx
113 f0100092:       c9                      leave
114 f0100093:       c3                      ret
115
```

可以看到共有四类栈空间被使用：

1. 开始时将%ebp 保存空间
2. 在栈中保存%ebx
3. 保留 0xc 空间用于临时变量储存
4. 执行 call 命令时，将%eip 压入栈中

综上，每次调用共有 4+4+12+4=24byte 的空间入栈

Exercise11:

在 init.c 中找到 test_backtrace

```
void
test_backtrace(int x)
{
        cprintf("entering test_backtrace %d\n", x);
        if (x > 0)
                test_backtrace(x-1);
        else
                mon_backtrace(0, 0, 0);
        cprintf("leaving test_backtrace %d\n", x);
}
```

函数递归调用，在最后一层调用了 mon_backtrace

```
int
mon_backtrace(int argc, char **argv, struct Trapframe *
{
        uint32_t bp,ip,arg1,arg2,arg3,i;
        bp=read_ebp();
        //读取ebp值
        ip=*((uint32_t*)bp+1);
        //从ebp指向的堆栈位置读取函数调用返回地址
        arg1=*((uint32_t*)bp+2);
        arg2=*((uint32_t*)bp+3);
        arg3=*((uint32_t*)bp+4);//从ebp指向的堆栈位置读
        do{
        cprintf("ebp %x eip %x args %x %x %x \n",bp,ip,
实验要求打印信息
        bp=*(uint32_t*)bp;//读取外层函数的ebp
                if(bp!=0){
                ip=*((uint32_t*)bp+1);
                arg1=*((uint32_t*)bp+2);
                arg2=*((uint32_t*)bp+3);
                arg3=*((uint32_t*)bp+4);
                }
        }while(bp!=0);//循环到最外层的程序位置
        return 0;
```

Exercise12:

得到行号并修改 mon_backtrace

```
stab_binsearch(stabs, &lline, &rline, N_SLINE, addr);
if (lline > rline) {
    return -1;
} else {
    info->eip_line = stabs[rline].n_desc;
}
```

```
mon_backtrace(int argc, char **argv, struct Trapframe *tf)
{
        // Your code here.
        uint32_t *ebp,*eip;
        uint32_t arg0,arg1,arg2,arg3,arg4;
        ebp=(uint32_t*)read_ebp();
        eip=(uint32_t*)ebp[1];
        arg0=ebp[2];
        arg1=ebp[3];
        arg2=ebp[4];
        arg3=ebp[5];
        arg4=ebp[6];

        cprintf("Stack_backtrace:\n");
        while(ebp!=0){
                cprintf("  ebp %08x eip %08x  args %08x %08x %08x %08x %08x\
n",ebp,eip,arg0,arg1,arg2,arg3,arg4);

                struct Eipdebuginfo info;
                debuginfo_eip(ebp[1], &info);
                cprintf("\n     %s:%d: %.*s+%d\n", info.eip_file, info.eip_l
ine, info.eip_fn_namelen, info.eip_fn_name, ebp[1] - info.eip_fn_addr);
                ebp = (uint32_t*)ebp[0];
                eip=(uint32_t*)ebp[1];
                arg0=ebp[2];
                arg1=ebp[3];
                arg2=ebp[4];
                arg3=ebp[5];
                arg4=ebp[6];
        }
        return 0;
```

三、实验结果截图

```
+ mk obj/kern/kernel.img
make[1]: Leaving directory '/home/lidongwen/Documents/6.828/lab'
running JOS: (2.2s)
  printf: OK
  backtrace count: OK
  backtrace arguments: OK
  backtrace symbols: OK
  backtrace lines: OK
Score: 50/50
lidongwen@ubuntu:~/Documents/6.828/lab$
```