lab1 文档

Exercise 8. We have omitted a small fragment of code - the code necessary to print octal numbers using patterns of the form "%o". Find and fill in this code fragment. Remember the octal number should begin with '0'.

先putch一个字符'0'，再将base设置为8，使用getuint获得数字。其他部分不需要改动，因为printnum可以处理16进制以下所有进制的数字的正常显示。

Exercise 9. You need also to add support for the "+" flag, which forces to precede the result with a plus or minus sign (+ or -) even for positive numbers.

当检查到'%'后跟着'+'则把padc设为'+'。在打印有符号十进制数字时，判断十进制数大于等于0且padc等于'+'时，先putch出字符'+'再打印数字。

Exercise 10. Enhance the cprintf function to allow it print with the %n specifier, you can consult the %n specifier specification of the C99 printf function for your reference by typing "man 3 printf" on the console. In this lab, we will use the **char *** type argument instead of the C99 **int *** argument, that is, "the number of characters written so far is stored into the **signed char** type integer indicated by the char * pointer argument. No argument is converted." You must deal with some special cases properly, because we are in kernel, such as when the argument is a NULL pointer, or when the char integer pointed by the argument has been overflowed. Find and fill in this code fragment.

vprintfmt中检查到字符'n'时，将putdat的值移动到%n所对应指针类型的参数所指向的地址。检查putdat大于127则警告溢出。检查到指针类型参数为NULL，则报错并break。

Exercise 11. Modify the function `printnum()` in `lib/printfmt.c` to support `"%-"` when printing numbers. With the directives starting with "%-", the printed number should be left adjusted. (i.e., paddings are on the right side.) For example, the following function call:

```
cprintf("test:[%-5d]", 3)
```

, should give a result as

```
"test:[3    ]"
```

(4 spaces after '3'). Before modifying `printnum()`, make sure you know what happened in function `vprintffmt()`.

printnum时检查padc，若padc等于'-'时，则递归打印高位数字时padc传入'@'。当最高位数字检查到padc='@'则不打印左边的padding。当打印完当前数位时，检查到padc等于'-'，则说明当前打印的是最低位数字。然后计算width和数位长度的差，并打印对应长度的空格作为padding。

Exercise 14. 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.

通过read_eip和read_ebp获得当前位置的eip寄存器值和ebp寄存器值。其中eip寄存器值的获取，是调用了一个不可被inline的函数`read_eip()`，并在里面使用inline汇编`movl 4(%%ebp),%0`获得的。在ebp高8字节的位置开始读取5个4字节的参数，打印出来作为当前栈帧的backtrace。

迭代地，取ebp所指向的内存地址上的4字节作为caller栈帧的ebp，取ebp高4字节的位置开始的4字节作为return address，即caller栈帧的eip。在每个迭代里都按上述方法获得参数信息并打印出来。

Exercise 15. Modify your stack backtrace function to display, for each `eip`, the function name, source file name, and line number corresponding to that `eip`.

在debuginfo_eip里面，加入对行号的查找。调用stab_binsearch，取lline为index处的n_desc作为行号，即`stabs[lline].n_desc`.在上题打印的信息的基础上，打印出debuginfo当中的文件名、行号、函数名、相对函数开始的字节数。其中相对函数开始的字节数是通过eip和函数地址作差得出的。

Exercise 16. Recall the buffer overflow attack in ICS Lab. Modify your start_overflow function to use a technique similar to the buffer overflow to invoke the do_overflow function. **You must use the above cprintf function with the %n specifier you augmented in "Exercise 10" to do this job, or else you won't get the points of this exercise**, and the do_overflow function should return normally.

通过对kernel.asm的汇编观察到，`overflow_me()`被inline到mon_backtrace的函数当中，并且观察到mon_backtrace中在调用`start_overflow()`之前有4字节之后不再被使用的空间。因此通过将`start_overflow()`的return address改为`do_overflow()`的第一条指令地址，并将`start_overflow()`的return address高4字节处的4字节空间改为本来应该正确返回到`mon_backtrace()`的return address。这样第一次在`start_overflow()`中ret，控制流会跳转到`do_overflow()`中。从`do_overflow()`中返回时，控制流会跳转到`mon_backtrace()`的正确地址。

重写return address的方法：先调用`read_pretaddr()`获得return address的位置，再通过%n逐步将需要的字节写入到对应的内存位置。

Exercise 17. There is a "time" command in Linux. The command counts the program's running time.

```
$time ls
a.file b.file ...

real  0m0.002s
user  0m0.001s
sys 0m0.001s
```

In this exercise, you need to implement a rather easy "time" command. The output of the "time" is the running time (in clocks cycles) of the command. The usage of this command is like this: "time [command]".

添加time命令。mon_time中检查第二个参数是否是正确的命令，如果正确则调用该命令`commands[i].func(argc - 1, ++argv, tf);`.调用对应命令时，需要去掉第一个time参数。调用对应命令前，调用`read_tsc()`获得开始时间，对应命令返回后再次调用`read_tsc()`获得结束时间，两者作差即为运行时间。