Lab 4: Traps

RISC-V assembly

完成该实验，需要同学们对RISC-V有一定的了解，具体可以参照学习该文档（[6.1810 / Fall 2022 (mit.edu)](https://pdos.csail.mit.edu/6.1810/2022/)）

此后需要同学们阅读以下代码：

user/call.c

通过编译该文件，你能够得到对应的汇编文件(.asm)，阅读函数g,f,main，在实验主目录下并创建文件 answers-traps.txt。

回答以下问题，将你的答案保存在此前创建的answers-traps.txt中。

1. Which registers contain arguments to functions? For example, which register holds 13 in main's call to printf?

2. Where is the call to function f in the assembly code for main? Where is the call to g? (Hint: the compiler may inline functions.)

3. At what address is the function printf located?

4. What value is in the register ra just after the jalr to printf in main?

5.

Run the following code.

```
unsigned int i = 0x00646c72;
printf("H%x Wo%s", 57616, &i);
```

What is the output? Here's an ASCII table that maps bytes to characters.

The output depends on that fact that the RISC-V is little-endian. If the RISC-V were instead big-endian what would you set i to in order to yield the same output? Would you need to change 57616 to a different value?

参考资料:

http://www.webopedia.com/TERM/b/big_endian.html

https://www.rfc-editor.org/ien/ien137.txt

Backtrace

我们平常在GDB调试中，依靠backtrace对bug进行定位。在本次实现中我们需要实现打印函数的调用栈。实现类似GDB中bt的效果。

在 kernel/printf.c 中实现 backtrace() 函数

Implement a backtrace() function in kernel/printf.c. Insert a call to this function in sys_sleep, and then run bttest, which calls sys_sleep. Your output should be a list of return addresses with this form (but the numbers will likely be different):

```
backtrace:
0x0000000080002cda
0x0000000080002bb6
0x0000000080002898
```

After bttest exit qemu. In a terminal window: run addr2line -e kernel/kernel (or riscv64-unknown-elf-addr2line -e kernel/kernel) and cut-and-paste the addresses from your backtrace, like this:

```
$ addr2line -e kernel/kernel
  0x0000000080002de2
  0x0000000080002f4a
  0x0000000080002bfc
  Ctrl-D
```

You should see something like this:

```
kernel/sysproc.c:74
kernel/syscall.c:224
kernel/trap.c:85
```

Backtrack Hints

Add the prototype for your backtrace() to kernel/defs.h so that you can invoke backtrace in sys_sleep.

The GCC compiler stores the frame pointer of the currently executing function in the register s0. Add the following function to kernel/riscv.h:

```
static inline uint64
r_fp()
{
  uint64 x;
  asm volatile("mv %0, s0" : "=r" (x) );
  return x;
}
```

and call this function in backtrace to read the current frame pointer. r_fp() uses in-line assembly(https://gcc.gnu.org/onlinedocs/gcc/Using-Assembly-Language-with-C.html) to read s0.

These lecture notes(https://pdos.csail.mit.edu/6.1810/2022/lec/l-riscv.txt) have a picture of the layout of stack frames. Note that the return address lives at a fixed offset (-8) from the frame pointer of a stackframe, and that the saved frame pointer lives at fixed offset (-16) from the frame pointer.

Your backtrace() will need a way to recognize that it has seen the last stack frame, and should stop. A useful fact is that the memory allocated for each kernel stack consists of a single page-aligned page, so that all the stack frames for a given stack are on the same page. You can use PGROUNDDOWN(fp) (see kernel/riscv.h) to identify the page that a frame pointer refers to.

Once your backtrace is working, call it from panic in kernel/printf.c so that you see the kernel's backtrace when it panics.

Alarm

Alarm实验需要同学们为xv6添加一个功能，使其中的进程在每运行指定次数的ticks之后，触发一个函数，就像时钟一样。

这需要同学们新增一个系统调用sigalarm(interval, handler)。

If an application calls sigalarm(n, fn), then after every n "ticks" of CPU time that the program consumes, the kernel should cause application function fn to be called. When fn returns, the application should resume where it left off. A tick is a fairly arbitrary unit of time in xv6, determined by how often a hardware timer generates interrupts. If an application calls sigalarm(0, 0), the kernel should stop generating periodic alarm calls.

You'll find a file user/alarmtest.c in your xv6 repository. Add it to the Makefile. It won't compile correctly until you've added sigalarm and sigreturn system calls

实现之后，同学们可以通过alarmtest来查看函数调用的效果

若实现正确，应看到右图结果：

```
$ alarmtest
test0 start
........alarm!
test0 passed
test1 start
...alarm!
..alarm!
...alarm!
..alarm!
...alarm!
..alarm!
...alarm!
..alarm!
...alarm!
..alarm!
test1 passed
test2 start
................alarm!
test2 passed
test3 start
test3 passed
$ usertest -q
...
ALL TESTS PASSED
$
```

Alarm Hints

- Your solution will require you to save and restore registers---what registers do you need to save and restore to resume the interrupted code correctly? (Hint: it will be many).
- Have usertrap save enough state in struct proc when the timer goes off that sigreturn can correctly return to the interrupted user code.
- Prevent re-entrant calls to the handler----if a handler hasn't returned yet, the kernel shouldn't call it again. test2 tests this.
- Make sure to restore a0. sigreturn is a system call, and its return value is stored in a0.

Once you pass test0, test1, test2, and test3 run usertests -q to make sure you **didn't break any other parts of the kernel.**

test0\test1\test2\test3 考察的重点各有不同，详见文档附件。

实验提交

在各位同学提交实验之前，务必运行 make grade命令以检验自己是否通过了所有的测试，并且**保证**xv6的其他模块没有被你新增的代码所损坏（详见上页PPT）。

1. 实验代码文件夹（提交前请make clean）
2. 实验报告（中文即可，PDF格式）
   - 实现思路，实现代码，测试结果。
   - 本实验中有问题回答环节，请勿遗漏
   - 实验中遇到的问题，如何思考并解决
   - 实验总结
3. 实验报告提交截止日期：2022年11月21日24:00点

1、2放入一个文件夹，命名为：学号-姓名-oslab4.zip

注意：请各位同学独立完成实验，参考代码需注明

实验报告（**PDF**）提交：

【腾讯文档】操作系统2022课程 Lab4

https://docs.qq.com/form/page/DTkloRmhvREJzQ3Nh

实验压缩包提交：

教室	对应邮箱
302	22210240347@m.fudan.edu.cn
204 205	michael_chen22@126.com
202	21210240021@m.fudan.edu.cn

请各位同学，务必提交到**链接**和**邮箱**。