****



**MIT xv6实验报告**

**——lab4 : traps**

**学生姓名 胡轶然**

**学 号 3019244355**

1. **实验目的**
2. 理解系统调用是如何使用traps实现的。
3. 实现一个简单的中断处理。
4. 了解栈的结构。
5. **前期准备**
6. 切换git分支。
7. 阅读指导书第4章。
8. 阅读kernel文件夹下的trampoline.S和kernel/trap.c。
9. **实验内容及实现步骤**

**任务1 RISC-V assembly**

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

答：a) 由实验2可知，参数存储在**a0~a7**寄存器中。

b) 13是printf的第三个参数，故位于**a2**寄存器中。

1. 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.)

答：阅读汇编代码可知，函数f的代码被内联嵌入到main函数中(**从函数main的0x24开始**)，而函数g的代码则被内联嵌入到函数f中（**从函数f的0x14开始**），在**内联f、g函数之前，都会将原函数的栈帧指针存入寄存器s0中**（函数main：0x22，函数f：0x12）。

1. At what address is the function printf located?

答：查看main函数的汇编代码，根据0x30处的指令可知，printf位于(pc(0x30)+1528)=**0x628**

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

答：执行auipc指令(**0x30**)后，ra存储内容与程序计数器pc相同，为0x30。0x34的**jair指令会将**

**（当前地址0x34+ra）写入ra寄存器中**，所以此时ra存储的内容为0x34+0x04=**0x38**

1. What is the output?

答：按题意编写程序，运行结果为：He110 World （110是数字）

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?

答：无论采用大端序还是小端序，输出的16进制数不会改变，仍然是0xe110，不会改变；而字符串rld需要使用大端序表示和存储，即0x726c6400

1. [Here's a description of little- and big-endian](http://www.webopedia.com/TERM/b/big_endian.html) and [a more whimsical description](http://www.networksorcery.com/enp/ien/ien137.txt).

In the following code, what is going to be printed after 'y='? (note: the answer is not a specific value.) Why does this happen?

答：输出的是不确定的长整数，具体取值与上一个函数的第二个参数有关。理由如下：

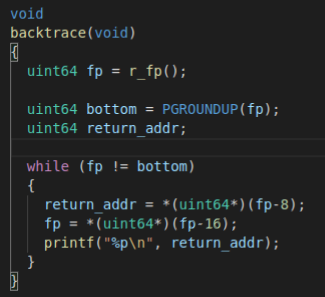
因为printf的格式化输出调用了两个参数(2个%d)，对应a0, a1寄存器，但只传入了1个参数（a0=3），printf输出的第2个参数是**a1中存储的值，也就是上一个“参数不少于2个的函数”的第2个参数**。

**任务2：Backtrace ([moderate](https://pdos.csail.mit.edu/6.828/2020/labs/guidance.html))**

1. **问题描述**

要求实现一个backtrace函数，被调用时，循环地打印进程栈帧页中各栈帧的返回值内容。

1. **思路与实现步骤**
2. 添加backtrace 函数的定义。
3. 在kernel/riscv.h文件中添加r\_fp函数，用于获取栈帧地址。指导书已给出函数内容。
4. 实现backtrace函数，从当前栈帧顶部向栈低循环输出栈帧内容。具体内容如下：
   1. 调用r\_fp函数，获取当前栈帧顶部地址(top of current frame)。
   2. 调用PGROUNDUP，获取栈帧底部地址。
   3. 从栈帧顶部向底部循环输出内容，**栈帧是从高地址向低地址增长的，uint64的大小为8字节，故返回值位于”fp-8~fp”，上一个栈帧指针位于”fp-16~fp-8”。**循环输出时，**首先输出”cur\_fp-8”处的返回值，然后定位到存储在”cur\_fp-16”处的前一个栈帧的指针。**
   4. 当cur\_fp等于stake page的顶部时，说明已打印到最后一个栈帧，此时退出循环。



1. 在sys\_sleep中调用上述函数，图略。
2. **问题和解决方法**

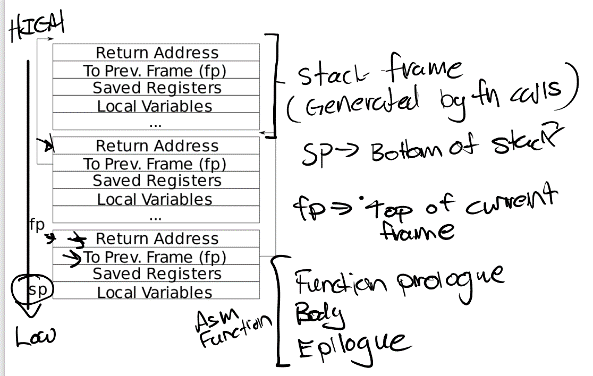
**问题一**：最初不清楚如何确定循环输出的起始地址，终止条件和循环步长（迭代方法）。

**解决方法**：

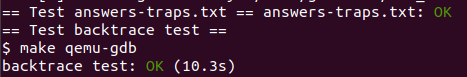
阅读指导书和下图可知，fp地址指向了当前栈帧顶部，上一栈帧的指针位于fp-16，由此可确定输出内容和循环迭代语句（步长）。本任务要求从当前栈帧打印到页顶部的栈帧，**当栈帧地址cur\_fp等于PGROUNDUP(cur\_fp)时，说明已到达页顶部，因为一个进程的栈帧大小是一页，故此时已遍历到最前面的栈帧，可以停止打印。**

**问题二**：不清楚fp指针和返回值的位置关系。

**解决方法**：由下图可知，返回值位于fp指针的下方，其地址应为**fp-8~fp，**故返回值地址应为fp-8。



1. **结果**



**任务3Alarm**

1. **问题描述**

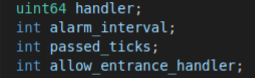
本任务要求实现系统调用sigalarm和sigreturn，每当某个应用调用该系统调用后，每隔一定时间单位触发一次backtrace。

1. **思路与实现步骤**

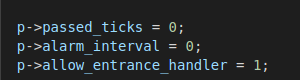
要完成任务，首先要在进程中保存中断步长和距上一次中断的时间。在触发中断时，要存储寄存器中的值，以及结束中断后的回调函数。在结束中断时，要恢复保存的寄存器的值，并继续执行回调函数。

具体思路和实现方法如下：

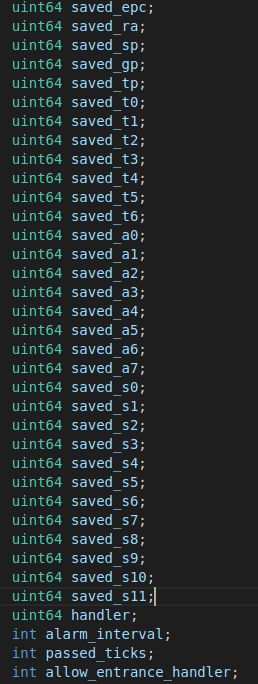
1. 按题意，添加系统调用sigalarm和sigreturn，具体操作同Lab2，这里不再赘述。
2. 在proc结构体中，添加下列属性，其含义依次为：
   1. 处理中断的函数指针。
   2. 中断步长。
   3. 距上一次中断过去的时间。
   4. 标记位，用于防止重复中断。



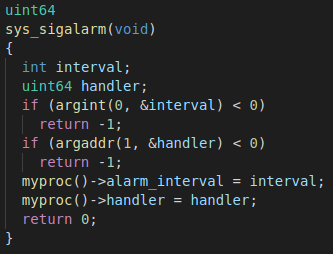
1. 修改allocproc函数，初始化与中断有关属性的值。



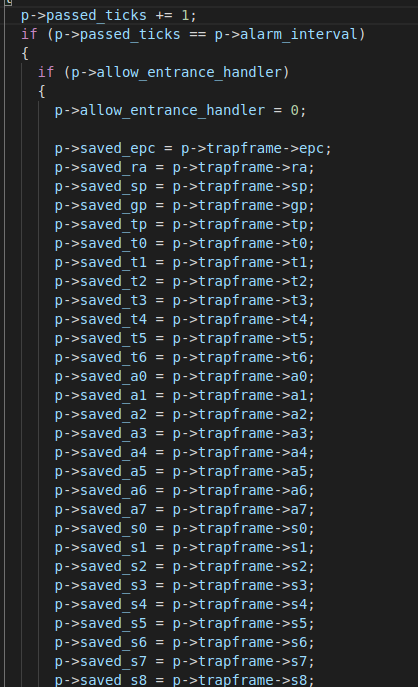
1. 在proc结构体中，添加如下属性，用于在发生中断时保存寄存器中的值，以便在中断结束后恢复现场。



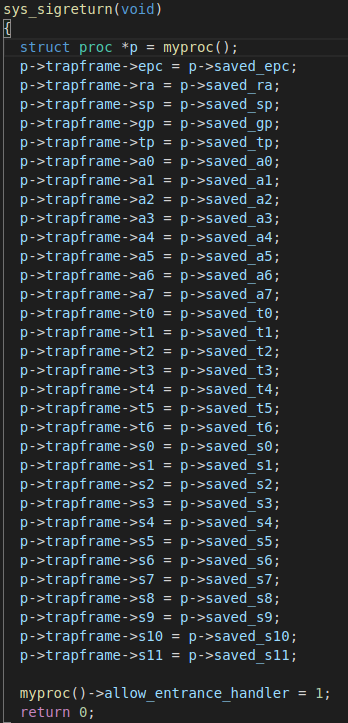
1. 添加sigalarm函数，读取interval和handler，并存入proc结构体对应属性中，具体内容如下
   1. 读取并存储中断间隔。
   2. 读取并存储中断处理函数的指针。



1. 修改usertrap函数，具体内容如下：
   1. 增大计时器的值。
   2. 在发生timeout，且没有重复中断时保存寄存器中的值，以待中断结束时恢复。
   3. 若发生重复中断（上次中断未结束就进入下次中断），则回调计时器的值。
   4. 将处理中断的函数指针存入epc寄存器中，开始执行中断处理函数。



1. 实现系统调用sigreturn，具体内容如下：
   1. 在中断结束后将保存的值恢复到寄存器中。
   2. 恢复完成后，打开”allow\_entrance\_handler”标记位，允许再次中断。

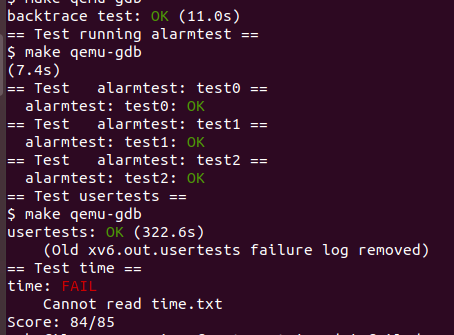


1. **问题与解决方法**

本任务暂无问题，按照指导书要求，实现寄存器的保存和恢复即可。

1. **结果**

通过了make grade测试。



注：因个人虚拟机内存较小，运行较慢，故延长了测试程序最长运行时间。此程序在其他虚拟机上运行时，可在规定时间内完成测试。