操作系统课程设计

Xv6实验报告

Lab4 traps

|  |  |
| --- | --- |
| Name | 朱从周 |
| ID | 2351893 |
| Number | 42028701 |
| major | 软件工程 |



同济大学

Tongji University

1. **环境搭建**
2. 下载VMware虚拟机，并且下载 Ubuntu 镜像并安装 Ubuntu 系统。
3. 启动Ubuntu，安装本项目所需的所有软件，运行：

$ sudo apt-get update && sudo apt-get upgrade

$ sudo apt-get install git build-essential gdb-multiarch qemu-system-misc gcc-riscv64-linux-gnu binutils-riscv64-linux-gnu

1. 下载xv6内核源码

$ git clone git://github.com/mit-pdos/xv6-riscv.git

1. 更新镜像源

$ sudo nano /etc/apt/sources.list

$ sudo apt-get update

1. 获取源码

$ git clone git://g.csail.mit.edu/xv6-labs-2021

$ cd xv6-labs-2021

$ git checkout traps

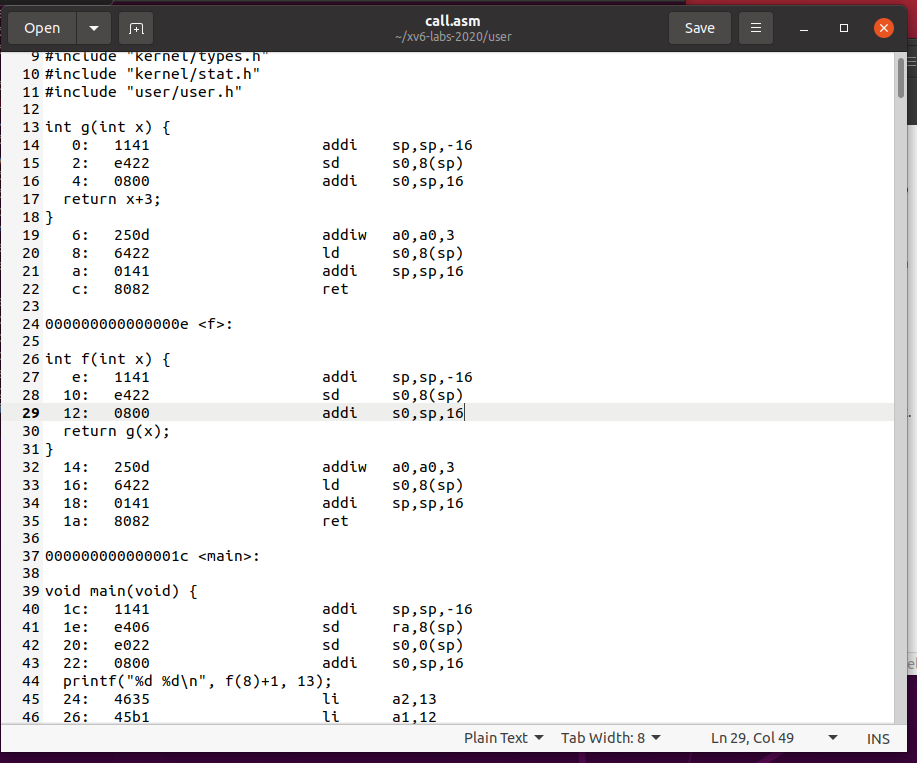
**1. 实验目的**

实现一个回溯（backtrace）功能，用于在操作系统内核发生错误时，输出调用堆栈上的函数调用列表。这有助于调试和定位错误发生的位置。

本次实验将向 xv6 内核添加一个新的功能，即周期性地为进程设置定时提醒。这个功能类似于用户级的中断/异常处理程序，能够让进程在消耗一定的 CPU 时间后执行指定的函数，然后恢复执行。通过实现这个功能，我们可以为计算密集型进程限制 CPU 时间，或者为需要周期性执行某些操作的进程提供支持。

**2. 实验内容**

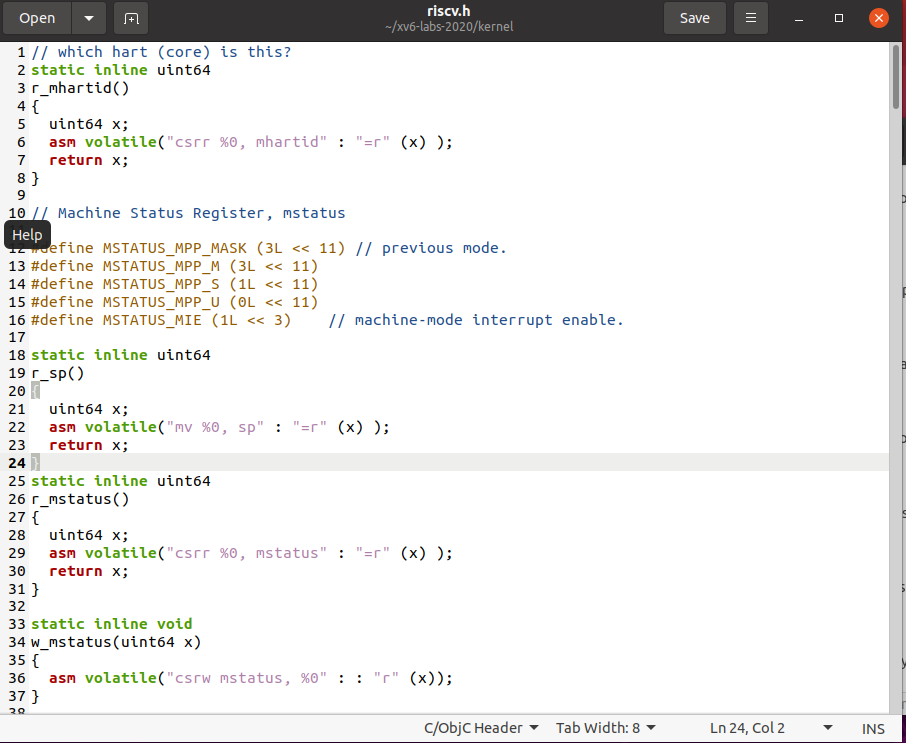
**RISC-V assembly (**[**easy**](https://pdos.csail.mit.edu/6.828/2021/labs/guidance.html)**)**



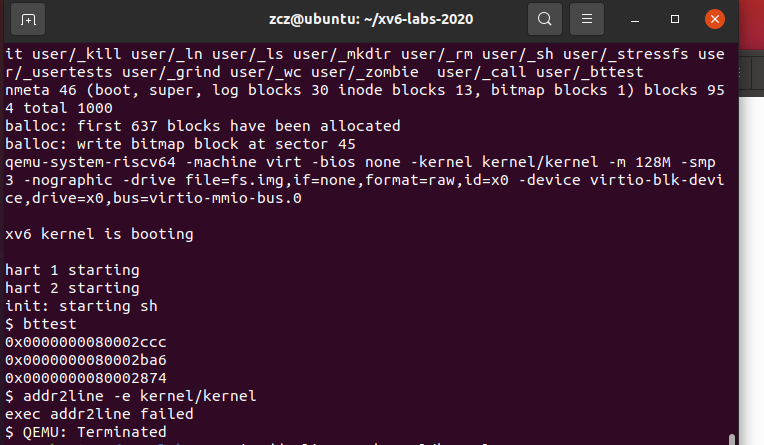
**Backtrace (**[moderate](https://pdos.csail.mit.edu/6.828/2021/labs/guidance.html)**)**

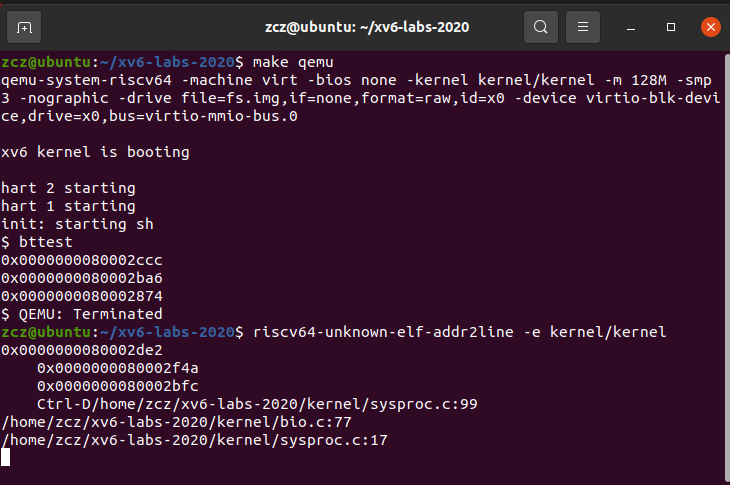


在 kerne/riscv.h 中添加 r\_sp函数。



测试结果：

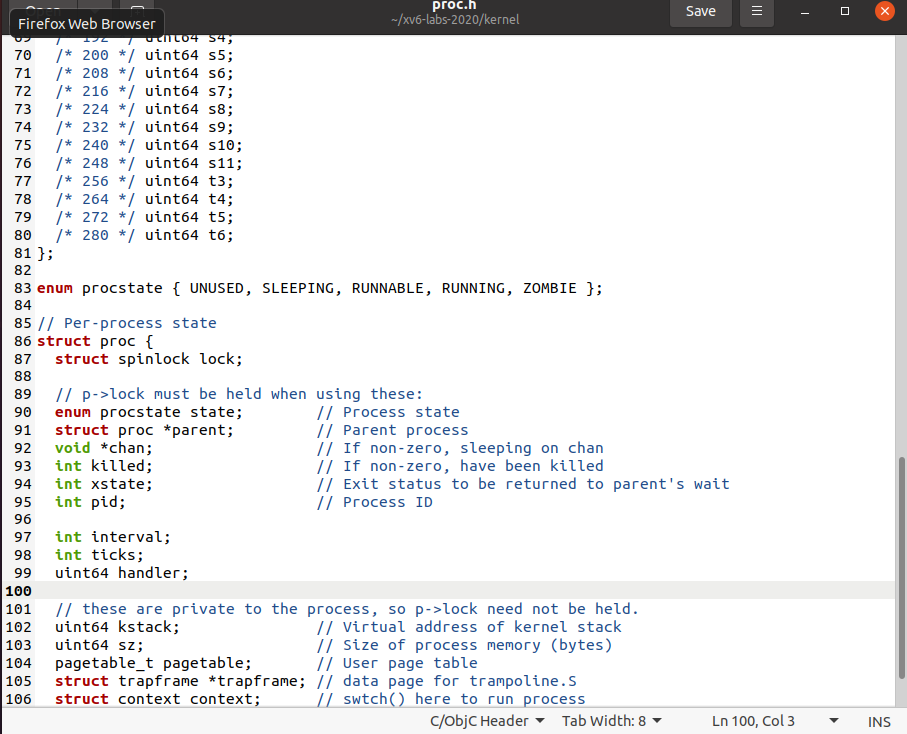




**Alarm (**[**hard**](https://pdos.csail.mit.edu/6.828/2021/labs/guidance.html)**)**

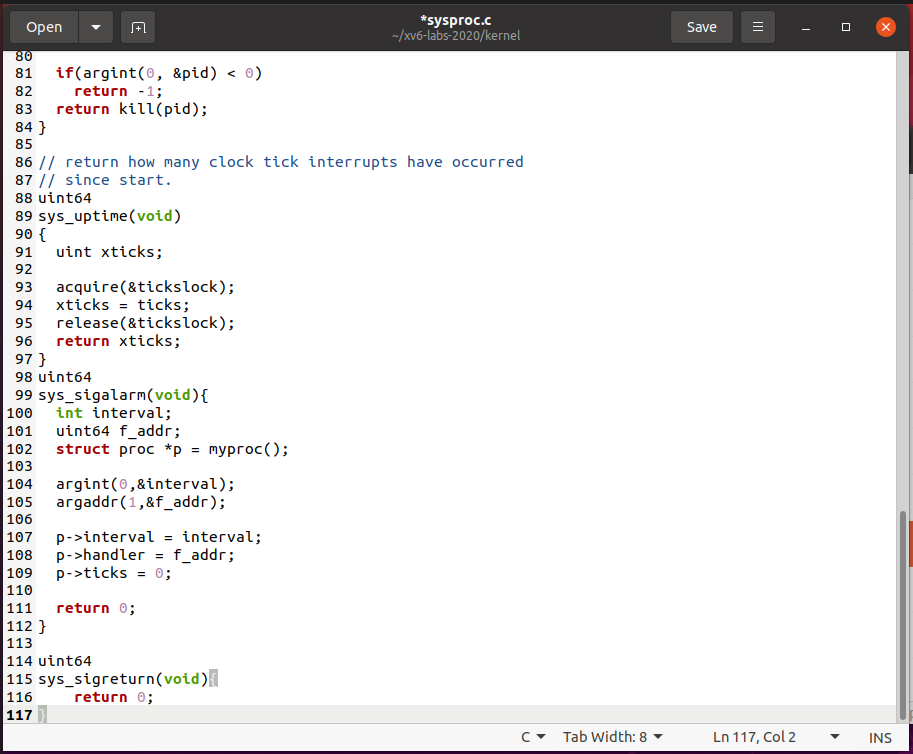
1. 在proc.h进程结构体中添加结构体字段

* interval：触发alarm的时钟中断数量
* ticks：需要统计的时钟中断的数量
* handler：记录处理alarm的函数的地址

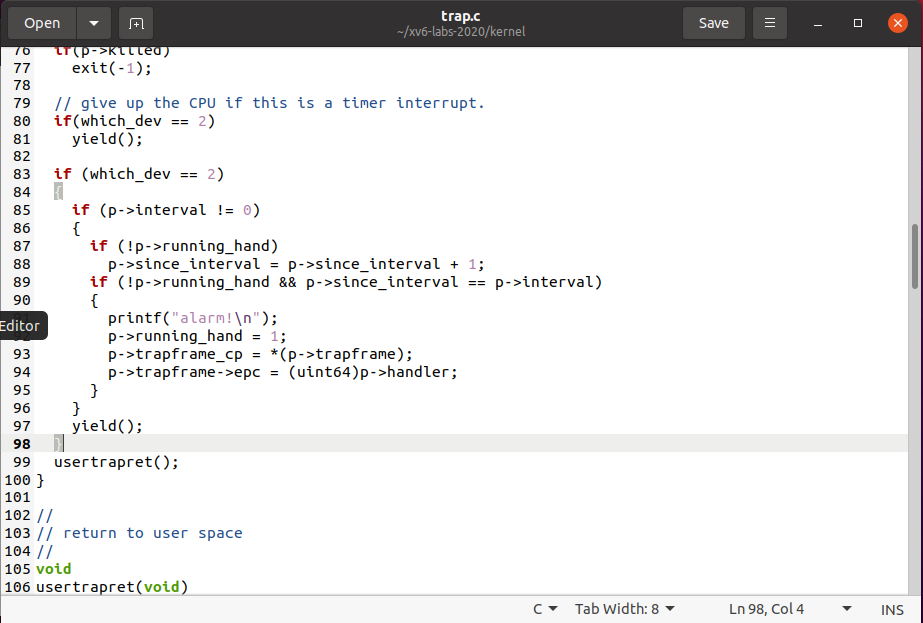


在sysproc中添加系统调用函数的实现

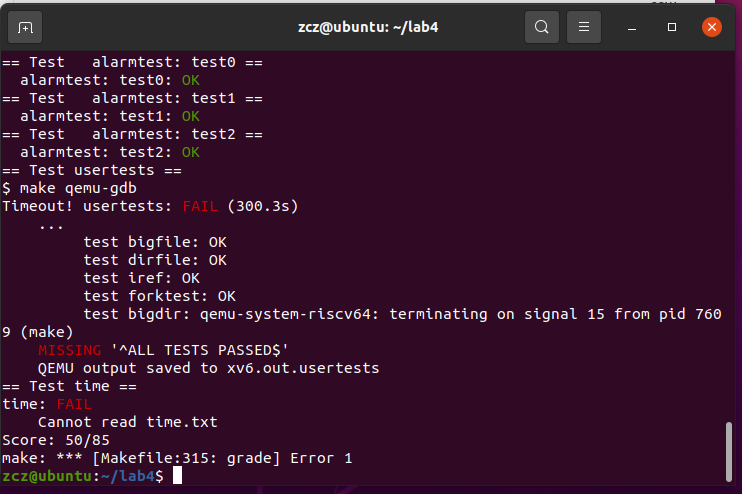
* sys\_sigaram：初始化proc有关alarm字段
* sys\_sigreturn：这时只返回0

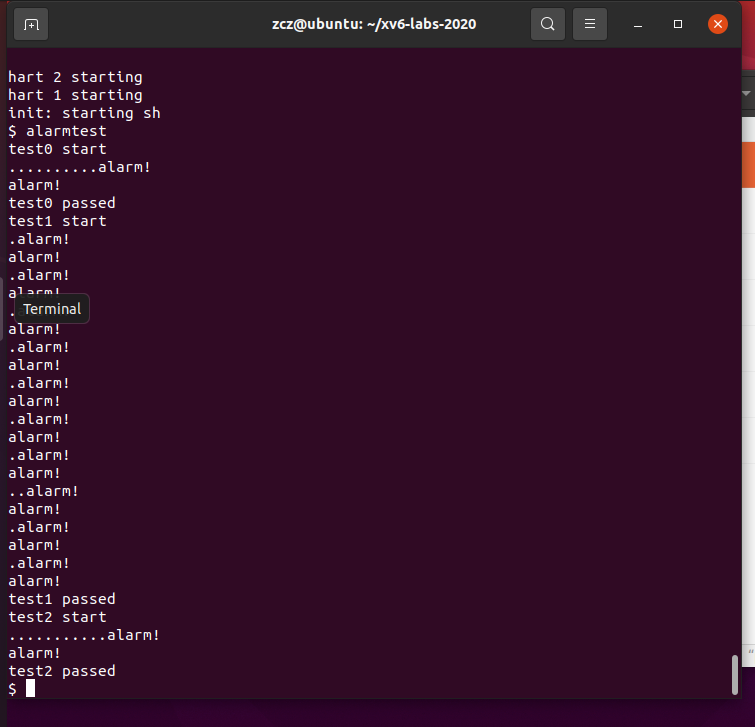


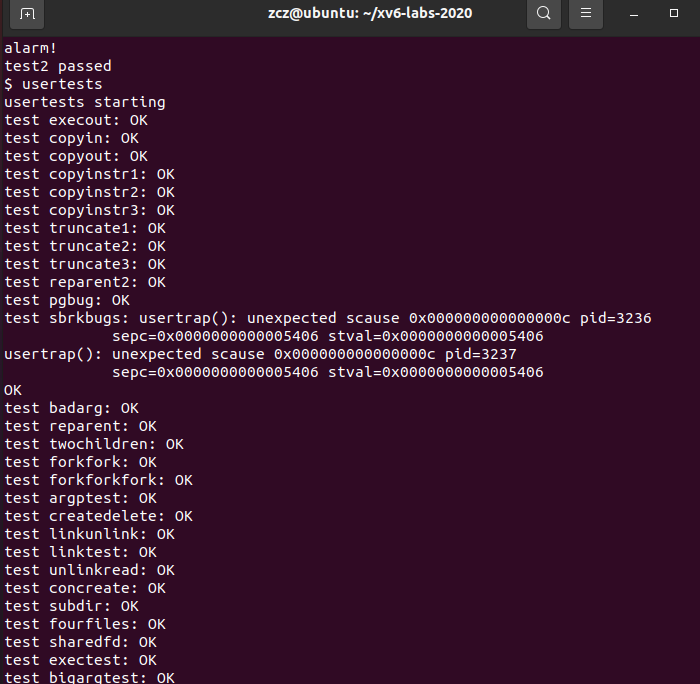
3. 在usertrap函数中的时间中断处处理alarm，判断设备中断为时钟中断则增加ticks字段，当时钟中断次数到达**指定数量**时，就将内存中epc的寄存器值改变为**alarm处理函数的地址**

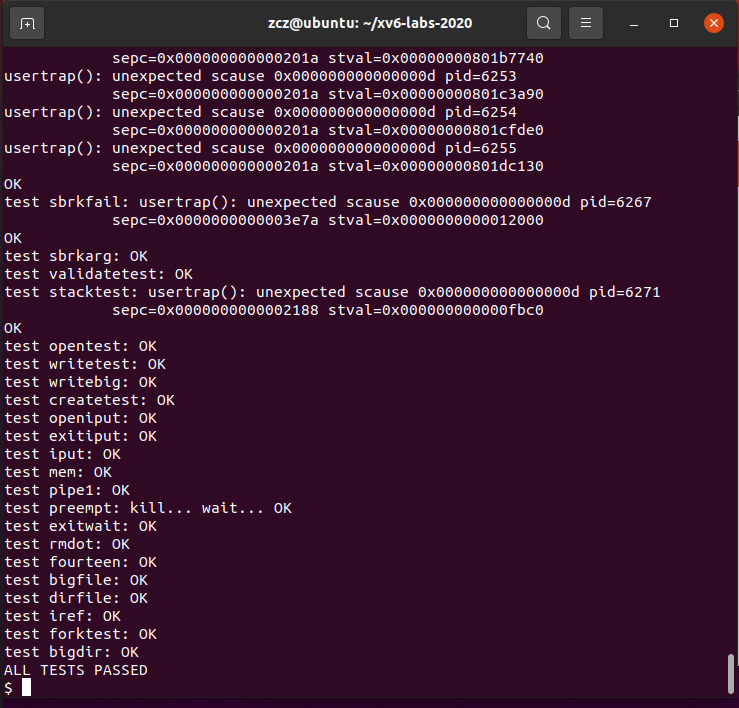
****

**测试结果：**

****

****

****



**3、实验中遇到的问题和解决方法**

1. **帧指针的使用：** 在实验中，我需要理解帧指针在调用堆栈中的作用，以及如何使用它来访问上一级函数的返回地址。

编译器会在每个堆栈帧中放置一个帧指针，用于保存调用者帧指针的地址。我需要反向跟踪并使用这些帧指针在堆栈中向上走动，并在每个堆栈帧中打印保存的返回地址。

此外，为了正确输出地址，我需要理解返回地址和堆栈帧指针之间的位置关系，通过查看笔记，我了解到返回地址与堆栈帧的帧指针有固定偏移（-8），而保存的帧指针与帧指针有固定偏移（-16）。

1. **循环终止条件：** 在 backtrace 函数中，我需要遍历整个调用堆栈，但是起初我不清楚这个循环需要一个什么样的终止条件。后来通过学习我发现，使用 PGROUNDDOWN 和 PGROUNDUP 宏可以帮助我计算栈页的顶部和底部地址，从而确定循环终止的条件。

**3、实验心得**

通过实现定时警报功能，我学会了如何在操作系统内核中处理定时事件。这不仅包括设置和触发定时器，还涉及到在中断处理程序中保存和恢复进程状态。

首先，我理解了警报处理函数的执行和返回机制。通过设置警报间隔和处理函数，在时钟中断时触发警报处理函数，然后通过 sys\_sigreturn 函数恢复原始状态，确保程序能够继续执行。

其次，时钟中断处理是实现定时功能的关键。通过在 usertrap 函数中处理时钟中断，我们可以精确控制警报触发的时间间隔。这让我更加理解了中断处理程序的设计和实现。

此外，本实验还让我体验到了操作系统的实时性要求。在处理定时事件时，需要确保系统的响应速度和准确性，这对提高操作系统的性能和稳定性具有重要意义。