**浙江大学实验报告**

课程名称： 操作系统 实验类型： 综合

实验项目名称： RV64 缺页异常处理

学生姓名： 专业： 计算机科学与技术 学号：

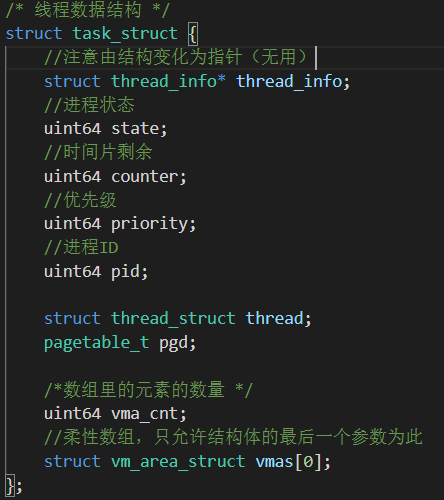
实验日期： 2023 年 12 月 12 日

1. **实验内容**

**记录实验过程并截图，对每一步的命令及结果进行必要的解释**

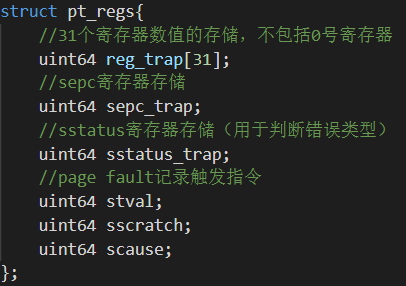
* 1. 规范已有代码

在本实验的设计中，我们需要扩充task\_struct结构体，将vm\_area\_struct（虚拟内存管理基本单元）纳入task\_struct结构体中。

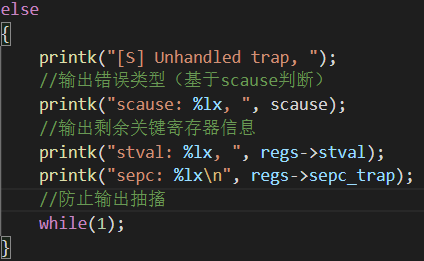


在修改task\_struct结构体之外，我们还需修改switch\_to函数的部分实现细节，保证进程调度时上下文切换操作的正常运行。

修改pt\_regs结构体和trap\_handler函数，使其更好地捕获各类异常并辅助调试。其中我们可以向pt\_regs结构体中添加stval、sscratch、scause三个CSR寄存器以辅助记录调试需要信息。pt\_regs结构体的具体实现如下：



在trap\_handler函数中，我们也适当增加输出了调试需要的各类寄存器信息。部分输出信息如下：



* 1. vma结构的实现

在linux系统中，vm\_area\_struct是虚拟内存管理的基本单元，vm\_area\_struct保存了有关连续虚拟内存区域的信息。其中vma结构主要记录信息如下：

vm\_start: 当前段虚拟内存区域的开始地址；

vm\_end: 当前段虚拟内存区域的结束地址；

vm\_flags: 当前vm\_area的权限标志；

vm\_pgoff: 虚拟内存映射区域在文件内的偏移量；

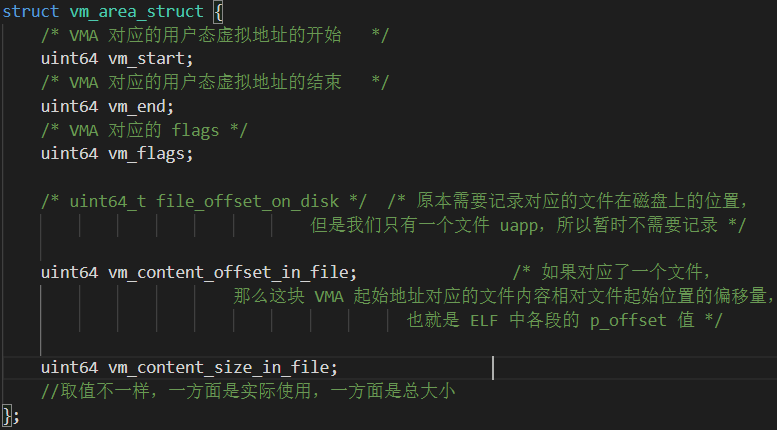
vm\_file: 记录映射文件所属设备号，指向关联文件结构的指针文件名；

vm\_ops: 当前vm\_area中的一组工作函数，其中是一系列函数指针，可根据需要定制；

vm\_next/vm\_prev: 同一 task 的所有虚拟内存区域由链表结构链接，这是分别指向前后两个vm\_area\_struct结构体的指针；

除了跟文件建立联系以外，vma还可能是一块匿名 (anonymous) 的区域。

我们需要在proc.h文件中添加相应的vma结构体，vma结构体的具体记录信息如下：



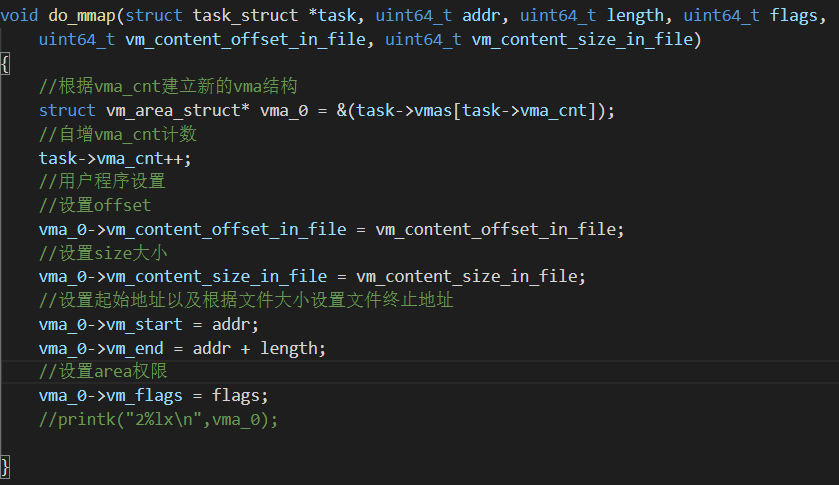
其中task\_struct中使用柔性数组记录不定个数的vm\_area\_struct结构体；且每一个 vm\_area\_struct都对应于task\_struct地址空间的唯一连续区间。为了实现Page fault的处理函数，我们需要先实现do\_mmap与find\_vma两个辅助函数。

* 1. dommap函数

do\_mmap主要创建一个新的vma结构体。其具体函数签名如下：

do\_mmap（task, addr, length, flags, vm\_content\_offset\_in\_file, vm\_content\_size\_in\_file);该函数接收六个参数，task为需要建立vma结构体对映的进程，addr记录该进程对映用户态程序的开始（虚拟）地址，length表示用户态程序长度，flags记录当前vma\_area的管理权限，vm\_content\_offset\_in\_file为用户态程序相对于elf文件头的偏移量，vm\_content\_size\_in\_file为用户态程序的大小。

函数的具体实现如下图：

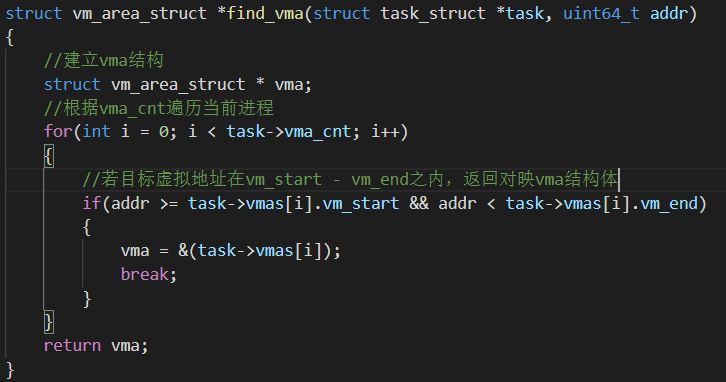


* 1. find\_vma函数

find\_vma函数查找包含某个addr的vma结构体，该函数主要在Page Fault处理时起作用。

find\_vma函数具体函数签名如下：

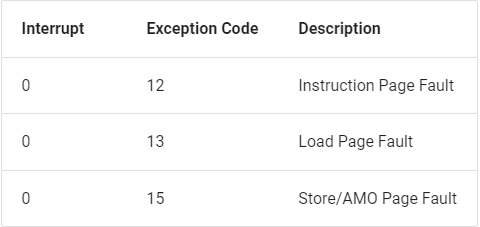
find\_vma(task, addr)其中addr为需要判断的虚拟地址，task为指定搜索vma结构的进程。find\_vma函数的具体实现如下：



* 1. Page fault的实现

在实现了do\_mmap与find\_vma两个函数的基础上，我们可以开展page table的处理。

当系统运行发生异常时，可即时地通过解析scause寄存器的值，识别如下三种不同的Page Fault。其中第一种page fault发生于instruction的读取上，第二第三种的page fault发生于stack数据的读取与修改上。



在处理page fault时，我们需要如下信息：

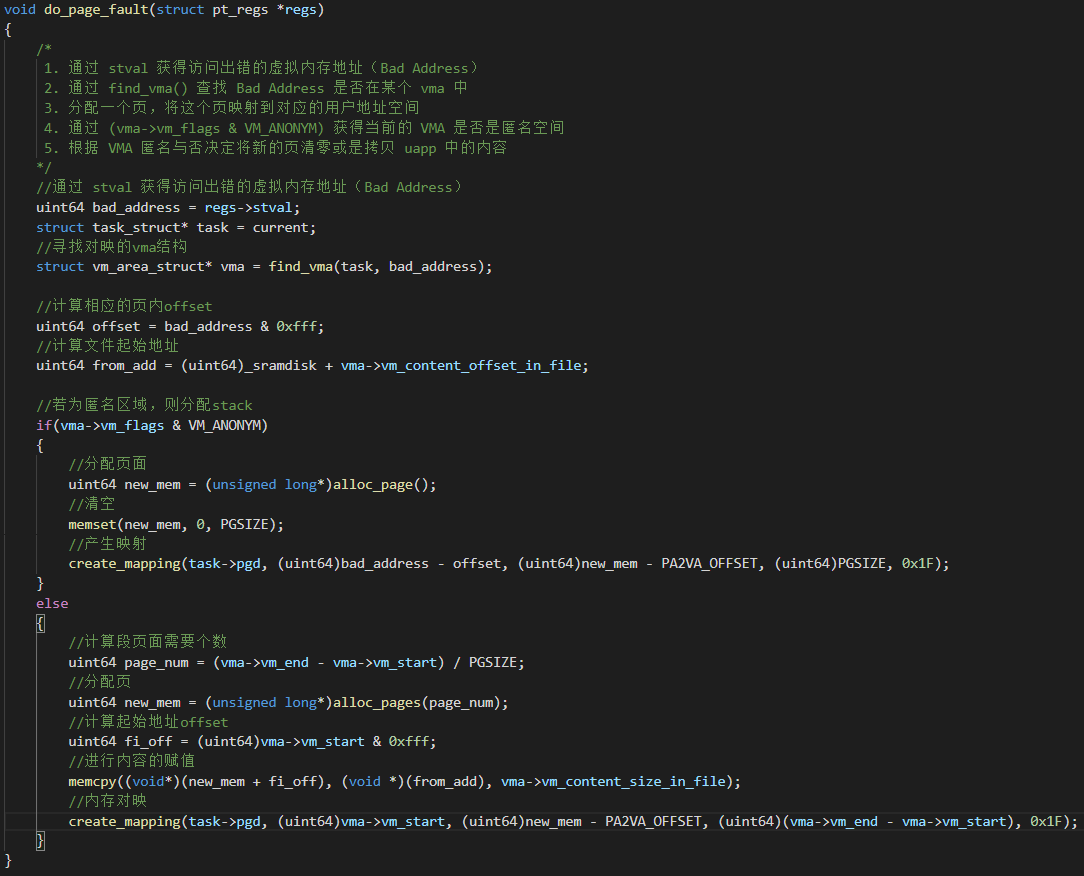
1.触发Page Fault时访问的虚拟内存地址VA。当触发 page faul时，stval寄存器会通过硬件自动设置为该出错的VA地址

2.发生Page Fault时的指令执行位置，会保存在sepc寄存器中。

对于处理缺页异常，我们需要在其发生时，检查VMA结构，如果当前访问的虚拟地址在 VMA中没有记录，即不合法的地址，则输出运行出错；如果当前访问的虚拟地址在 VMA 中存在记录，则进行相应的映射即可：如果访问的页是存在数据的，则需要从文件系统（本实验为elf格式文件）中读取内容，随后进行映射；若为匿名映射，则寻找一个可用的帧进行映射即可。

且在本实验中，在初始化task时我们既不分配内存，又不更改页表项来建立映射。在回退到用户态进行程序执行时就会因没有映射而Page Fault，从而进入Page Fault Handler函数，至此，我们再分配空间并进行映射。而为了减少task初始化时的开销，我们对一个Segment或用户态的栈只分别建立一个VMA。

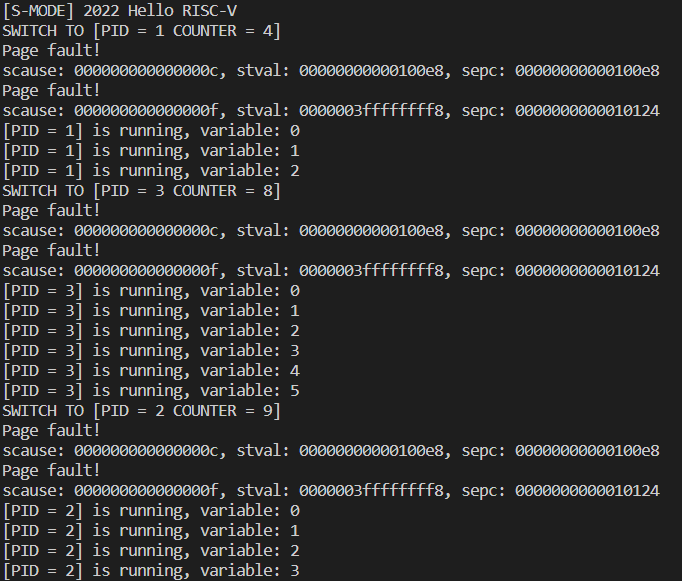
do\_page\_fault(pt\_regs \*regs)函数的具体实现如下。



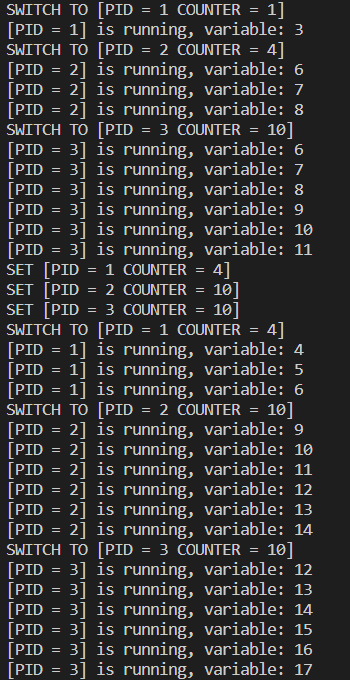
首先我们通过stval寄存器获得访问出错的虚拟地址，接着使用find\_vma函数查找该虚拟地址对映的vma结构位置；根据具体的vma结构，进行用户栈或者用户态程序的页面分配与拷贝操作。

至此，page fault的处理设计便已完成。

* 1. 测试结果

第一次调度运行结果：

由第一次调度结果我们可以看出。首先，我们设定的三个进程在进行正常的时间片轮转，并在第一次调度时均发生了page fault的情况。另外，我们也可以看到，每个进程均有属于自己的变量，因而每个进程的变量输出均从0开始递增。最后，对于用户栈的读取与写入以及代码段的读取，进程均发生了一次page fault。

第二次调度运行结果：

从测试结果我们可以看到，三个进程还在正常的进行时间片轮转调度；且每个进程对映的变量数值也不尽相同；最重要的是，因为是进程的第二次调度运行，各个进程不再产生page fault的情况。

1. **讨论题**
2. uint64\_t vm\_content\_size\_in\_file;对应的文件内容的长度。为什么还需要这个域?

首先，vm\_content\_size\_in\_file的大小与vm\_smart - vm\_end的大小并不一定相同；vm\_content\_size\_in\_file 往往表示文件内容的长度，而vm\_end - vm\_start往往表示在用户空间中vma的虚拟地址范围长度（往往更大）。

在具体的实现中，这两者也对映着p\_memsz与p\_filesz之间的差别；在可执行文件中，某些段可能包含了一些在运行时生成或计算的数据。即这些数据本身在文件中是不存在的，但在程序运行的过程中却需要为其分配内存空间来进行存储。因而vm\_smart - vm\_end 与vm\_content\_size\_in\_file尚存一定的区别，从而后者有存在的必要。

2.struct vm\_area\_struct vmas[0];为什么可以开大小为 0 的数组? 这个定义可以和前面的 vma\_cnt 换个位置吗?

Vma\_cnt与vm\_area\_struct\_vmas是柔性数组的一类实现。其中vma\_cnt表示在 vmas 结构体中的 struct vm\_area\_struct 元素的数量。对于柔性数组而言，其只允许在结构的末尾定义一个变长的数组，使其大小在运行时动态确定。在vma结构体中，vma\_cnt 就用于记录数组中实际元素的数量。

因而我们在定义vma结构体时，可以先开大小为0的数组，在运行时再根据需要动态分配数组大小；vm\_area\_struct\_vmas与vma\_cnt的定义也不能交换位置。

1. **心得**

相较于先前的实验，page fault处理函数的实现更为简洁明白，但其中的特别关键点却更为丰富。

在实验的实现过程中，依照框架，我天真的认为需要单独处理scause寄存器为12、13、15三种情况下的page fault，因而在初步实现page fault handler函数时，程序无法进行有效的输出（条件处理选择错误）与处理。

另外，对于p\_flags与vm\_flag没有一一对应的情况，我们并不需要复杂的去check p\_flags的每一bit，再设置vm\_flag中对应的权限控制位；借助于宏定义的mask，我们可以轻松的得到同样的flag设置结果。

总结而言，感觉本次实验的实现只是更换了page allocate与create\_mapping函数进行相应地址bind的位置，将原本在task初始化时调用转移到产生page fault时才进行。