操作系统课程设计

Xv6实验报告

Lab5 copy-on-write fork for xv6

|  |  |
| --- | --- |
| 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 pgtbl

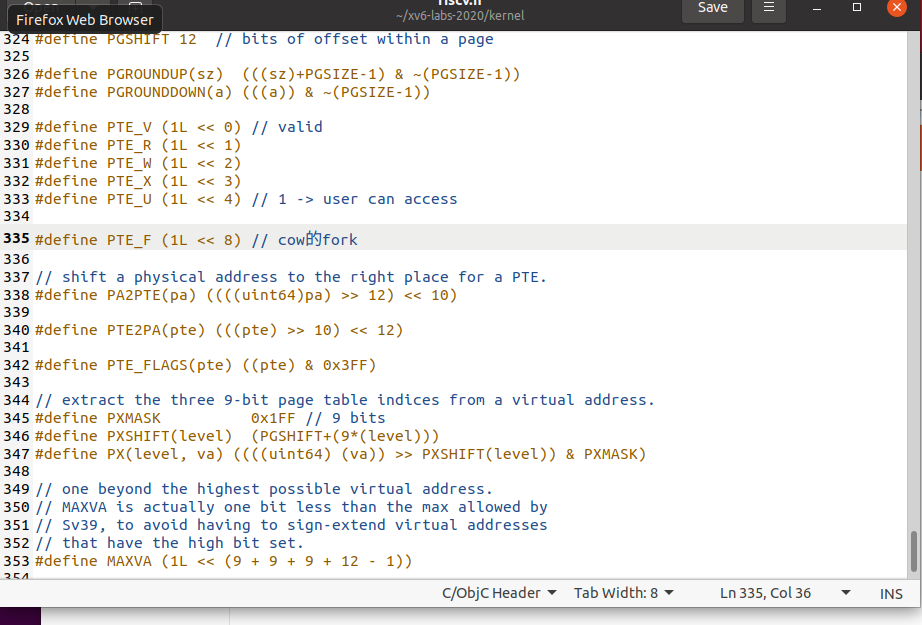
**1. 实验目的**

实验的主要目的是在 xv6 操作系统中实现写时复制（Copy-on-Write，COW）的 fork 功能。传统的 fork() 系统调用会复制父进程的整个用户空间内存到子进程，而 COW fork() 则通过延迟分配和复制物理内存页面，只在需要时才进行复制，从而提高性能和节省资源。通过这个实验，你将了解如何使用写时复制技术优化进程的 fork 操作。**2. 实验内容**

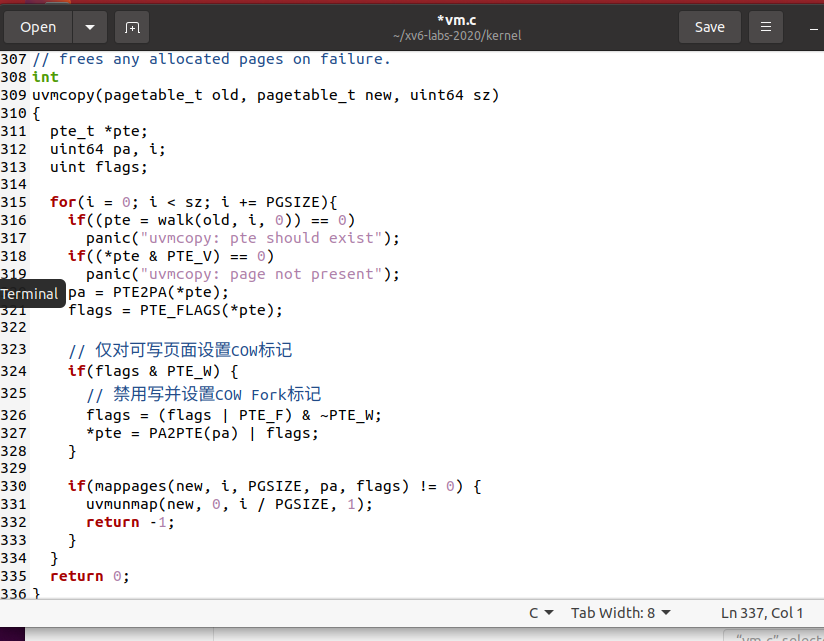
**Implement copy-on write**

第一步 修改uvmcopy

修改uvmcopy()将父进程的物理页映射到子进程，而不是分配新页。在子进程和父进程的PTE中清除PTE\_W标志。，但是得加一个标准位 （在\*\*\*kernel/riscv.h\*\*\*的末尾），使用RISC-V PTE中的RSW（reserved for software，即为软件保留的）位来实现此目的。

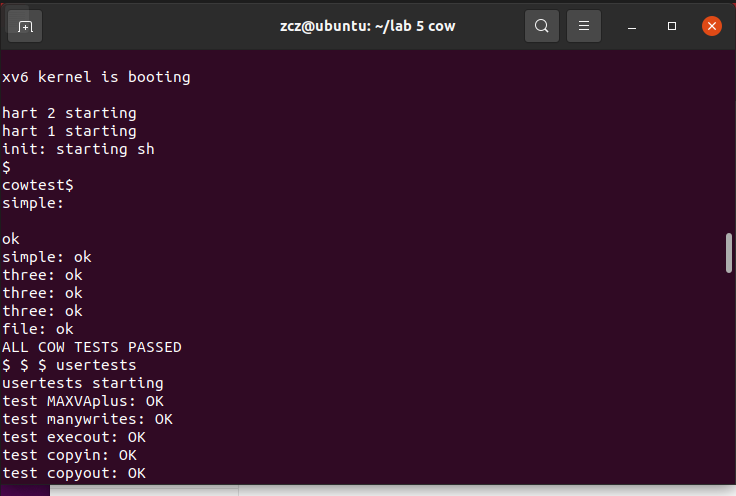


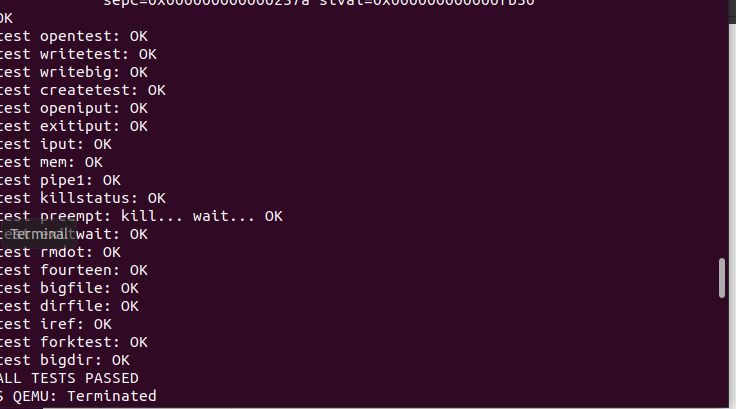
修改uvmcopy，不为子进程分配内存，而是使父子进程共享内存，但禁用PTE\_W，同时标记PTE\_F， **（只mappages，不kalloc）** （这里没加引用计数，后面补上）

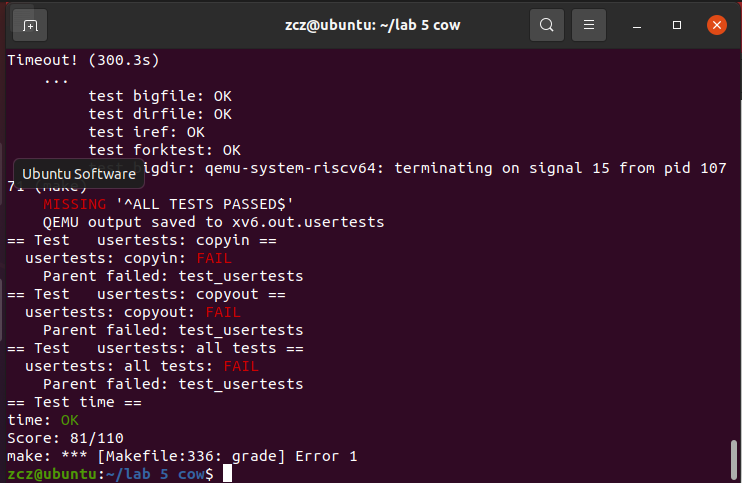




测试结果：







**3.实验心得**

在实验过程中，我遇到了关于处理页面故障的问题。刚开始时，我对于如何确定页面故障以及如何获取相应的异常代码和地址信息感到困惑。然而，通过查阅 RISC-V 架构规范和相关文档，我逐渐理解了异常处理的流程，并找到了解决方法。

首先，我发现 RISC-V 架构规范提供了一个清晰的表格，列出了各种异常代码和异常类型的对应关系。这个表格不仅告诉我们异常代码为 15 时对应于存储访问异常，也提供了同步异常的优先级等信息。通过阅读这些信息，我能够正确地识别和处理存储访问异常。

另一个问题是如何获取引发异常的虚拟地址。在 RISC-V 架构中，存在一个监控程序陷阱值寄存器（stval），用于存储导致发生异常的虚拟地址。通过阅读相关文档，我了解到可以使用 r\_stval() 函数来获取这个寄存器的值。在页面故障发生时，我使用了这个函数来获取引发异常的虚拟地址，从而能够在需要时为其分配新的物理内存页面并进行复制操作。

通过充分理解异常代码、异常类型以及相关寄存器的作用，我成功地在 usertrap() 函数中添加了对页面故障的处理逻辑。我对异常处理的理解变得更加深入，也意识到了在处理底层硬件异常时，详细了解架构规范的重要性。

此外，我还学习了PTE 标志管理，通过修改 PTE 标志以实现页面的写时复制。实验中，我学习了如何正确地设置和清除 PTE 中的标志，以及在页面写入时触发页面故障。

总而言之，在实现写时复制时，需要管理页面的分配和释放，以及处理页面复制。我通过学习如何正确地为页面分配和释放内存，使得其在需要时进行复制，以避免资源泄漏和内存溢出。这是提高计算机性能的重要的操作和思想，通过这样的学习我获益匪浅。