实验7 分组实验

谷建华
2023-12-03 v0.4

此次实验为大作业，大家以两人一组，选取一个实验题目实现。

> 注意测试程序的设计，这是项目的重要组成部分，并且对于最终得分有重要影响。测试程序可以是用户程序，也可以实现在内核中（例如打印统计信息），当然也可以是系统调用。
>
> 不一定要求所有的功能都能100%实现，可以从易到难逐步完成。评分标准不仅是工作量，还有完成的质量。
>
> 基准代码保证一定会存在bug，欢迎提交你们发现的基准代码中的bug以及解决方案。

1. 编写线程相关系统调用
    - 实现线程的创建、退出、等待
        - 实现 `pthread_create` `pthread_exit` `pthread_join` 三个函数的功能
    - 实现一些线程同步机制
        - 实现 `pthread_spin_*` `pthread_mutex_*` 等线程相关同步互斥原语。
    - 按照`glibc`给出的语义实现，如果有不一致的地方需要说明原因
    - 自行编写测试程序，参考以下要点：
        - 证明多个线程并发执行，并且共享同一个地址空间，最好是能够让多个多线程的进程同时执行。
        - 证明接口功能的实现符合语义要求
        - 利用前述的接口实现线程间同步，例如实现同步原语
2. 实现CFS调度策略和增量式sleep：
    - 实现CFS调度策略：
        - 参考Linux中实现的完全公平调度算法
        - 编写nice系统调用控制不同进程的分配时间片权重
    - 增量式sleep上课讲过，通过维护sleep队列中的等待时间增量实现O(1)弹出，插入为O(n)的sleep实现
    - 自行编写测试程序，参考以下要点：
        - 创建多个进程，设置每个进程的nice值，观察调度的变化。
        - 增加每个进程的统计信息，包括进程的nice值、运行时间等。
        - 让多个进程使用增量式sleep等待不同的时间，观察它们被唤醒的时间顺序。
        - 在大规模并发的情况下运行程序，检查CFS调度和增量式sleep的性能表现。
3. 实现`execve`系统调用：
    - 实现`exec`系列函数，主要是`execve`，为系统增加程序参数和环境变量支持
        - 语义参考Linux中的execve
    - 基于`execve`，实现更加完善的shell程序
        - 要求可以向程序传递参数，可以设置环境变量，可以同时启动多个程序。
        - 格式参考常见的Linux shell
    - 自行设计测试程序，参考以下要点：
        - 可以同时运行多个shell，shell可以向程序传递参数，可以设置环境变量，可以同时启动多个程序。

        - 环境变量应该从父进程继承到子进程，子进程中修改环境变量不能影响父进程

4. 实现缺页中断处理：
    - 实现写时复制（Copy on Write，CoW）机制
        - 通过修改fork系统调用，使系统调用并不是完全复制父进程的所有内存信息，而是与父进程内存共享，但是所有物理页修改成只读权限不允许修改。
        - 如果进程要执行写操作，就会触发Page Fault，对处理函数修改使得写时复制能够实现。
    - 实现延迟加载（Lazy Load）机制
        - 在启动用户程序的时候，只加载部分的ELF程序就直接开始执行。和 CoW 类似，当执行到一个不存在的页会触发Page Fault，然后在处理程序中再继续进行加载。
    - 自行编写测试程序，参考以下要点：
        - 证明你的 CoW 实现是正确的，例如多次执行 fork 和 exec 之后多个进程并发执行；例如子进程退出后父进程依然能够正常运行；等。
        - 证明 CoW 实现起到了作用，建议在内核中增加统计信息，例如缺页的次数和类型；例如编写一个申请了很多内存的程序再 fork（这时候如果没有 CoW 就会内存不足）。
        - 证明你的 Lazy Load 实现是正确且有效的，例如编写一个有很多静态数据的程序，对比加载执行的时间；或者是增加统计信息

5. 实现qemu虚拟机网络通讯：
    - 实现数据链路层驱动。例如，外界向qemu虚拟机网卡发送MAC帧，虚拟机能够接受到对应的数据并做出应答。
    - 实现系统调用或者是类似的接口，可以动态配置网卡。
    - 本任务自由度较高，测试程序按照实现情况编写。

6. 重写FAT32文件系统：
    - 目前的基准代码中，FAT32文件系统的实现存在问题，需要重新实现
        - 实现参考前面实验中的FAT文件系统手册
    - 测试程序设计参考
        - 运行多个进程，同时读取和写入多个文件，文件操作例如连续读写、随机读写、同时读写同一个目录下的多个文件，多个进程同时读写一个文件，同时新建和删除目录等。你不仅需要保证系统不崩溃，还需要考虑到数据的完整性不被破坏。
        - 使用linux下的`mkfs.vfat`工具制作空的文件系统，挂载并写入一些目录和文件，虚拟机能够正确地读取的这个文件系统，同时从虚拟机中写入的文件也能够从Linux下读取
    - 如果你实现了FAT32文件系统，请清理干净基准代码
        - 删除基准代码中的Orange文件系统相关的内容，并将用户程序直接打包进FAT32分区中。
        - 优化基准代码的编译过程，去除Makefile中硬编码的文件系统地址，使用更加通用的方式制作文件系统

7. 实现ext2文件系统：
    - 参考EXT2文件系统的标准手册，实现EXT2文件系统
        - EXT2文件系统更加复杂，可以先考虑简单实现出`read`,`write`,`open`,`close`,`create`这些vfs接口。
    - 测试程序设计参考

        - 运行多个进程，同时读取和写入多个文件，文件操作例如连续读写、随机读写、同时读写同一个目录下的多个文件，多个进程同时读写一个文件，同时新建和删除目录等。你不仅需要保证系统不崩溃，还需要考虑到数据的完整性不被破坏。
        - 使用linux下的`mkfs.ext2`工具制作空的文件系统，挂载并写入一些目录和文件，虚拟机能够正确地读取的这个文件系统，同时从虚拟机中写入的文件也能够从Linux下读取
8. 块设备（硬盘）缓冲区管理：
    - 目前给的实验代码中它对缓冲区的管理可以说是几乎没有，遇到一个请求就开始读/写，这样有一个不好的地方就是需要进行大量的磁盘I/O。相比于从物理设备中读入数据，在内存中修改明显更快。
    - 所以就需要在磁盘中构建一个缓冲区用于临时存放磁盘中的数据拷贝，所有的读写操作都在内存中完成，当缓冲区满（或者是出现需要强制刷新的情况）才将部分数据写回磁盘。
    - 要求每个缓冲区以文件系统的文件块（一般是4K）为单位，而不是以扇区为单位实施管理。
    - 测试程序参考
        - 证明的你缓存是正确的，例如进行大量的随机读写，对比进程写入的内容、进程读取的内容、磁盘上的实际内容是否一致
        - 同时运行多个测试程序，测试并发控制是否正确
        - 针对性设计测试用例，例如连续读写大量数据，或者是执行跨度较大的随机读写。
        - 增加缓冲块的统计信息，对比测试有无缓冲时的访问性能，证明实现的有效性
9. 实现一个基于内存的文件系统：
    - 在内核中开辟一段内存用于存放文件系统数据
        - 基本原理可以参考Linux的`ramfs`，在系统层次上和磁盘类似，属于块设备。不同的是它不能持久化存储，当系统关闭时文件系统也被销毁
        - 文件系统格式任意，可以实现得比较简单，重点在于实现驱动支持，将来可以支持更多的文件系统格式
        - 通过vfs的API能够对该文件系统进行访问
    - 本任务自由度较高，测试程序按照实现情况编写，同时参考其余两个文件系统的测试要求
10. 图形界面GUI实现：
    - 实现类似于现代操作系统的图形界面，包括如下基本元素
        - 程序窗口结构及其交互，如重叠、移动、最大最小化、新建和关闭等
        - 鼠标指针及其交互
    - 更加复杂的任务：
        - 图形组件库，例如按钮、菜单等
        - 用户程序对窗口的控制
    - 测试程序按照实现情况编写，本任务的自由度较高
    - 这里涉及到framebuffer的知识，如果学生能够采用AMD技术或AGP或PCIe来实现高效的图形处理，将会获得更好的成绩。
    - 实现图形界面有部分开源程序miniGUI、openGUI等，其他具有图形界面的开源操作系统项目也极具参考价值。
11. 实现signal信号机制：
    - 实现signal需要的系统调用如下：
        - `kill` 向指定进程发送信号
        - `sigaction` 用户可以根据信号的类型提供相应的用户态处理函数

        - `sigreturn` 用户在执行完用户态处理函数后通过该系统调用告知内核处理结束
        - `sigprocmask` 用于检查或修改当前进程的信号屏蔽集合
    - 实现语义参考Linux
    - 测试参考如下要点:
        - 用户程序能够正确注册处理程序，响应signal，或是屏蔽signal。
        - 多个用户程序能够并发地完成上面的测试。
        - 可以考虑利用信号机制实现进程同步等有趣的功能。或者是基于signal机制, 实现一个shell, 能够向子程序发送信号。
12. 实现管道系统：
    - 实现`pipe`和`dup`两类系统调用：
        - `pipe`创建管道文件描述符，`close`结合`dup`系统调用实现文件描述符的移动
    - 基于管道系统修改shell程序，使得通过管道实现多个进程间数据的传输，以及重定向功能
    - 测试参考
        - 父子进程之间的管道数据传输，写满和读空的情况
        - 特殊情况处理，例如进程退出而没有关闭管道，多组进程同时使用管道系统
        - 其他的功能，例如基于管道实现进程间同步等
13. 实现内存分配算法：
    - 实现buddy算法：以4k页为单位进行内存分配
    - 实现slab分配内存算法：在buddy算法的基础上实现更细粒度的内存分配
    - 在实现的过程中，需要对基准代码的内存分配系统进行梳理和问题修复
    - 测试参考
        - 分配和回收均能正常实现且不出现泄露，不同大小的申请、边界情况的申请
        - 多个进程同时请求的情况，大量的随机分配、读写、回收测试
        - 统计信息，例如性能、内存碎片情况等
14. TTY系统重写：
    - 当前的基准代码中的TTY实现存在诸多问题，需要重新编写
        - 可以参考前面的实验代码对其改造，能够适配CSI转义序列和kprintf；同时具有滚动、翻页等功能。
        - TTY在Linux中是一种字符设备驱动，具有较高的可扩展性，尝试将串口接入TTY
        - TTY在任何时刻都应该可用，因此它不应该关联任何的后台进程
    - 可以考虑基于TTY编写简单的伪终端程序，实现诸如作业控制的高级功能。
    - 本任务自由度较高，按照实现情况自行设计测试程序。
15. 探索启动64位x86系统的可能性：
    - 探索一下如何qemu启用64位系统，而不是32位i386。
    - 不需要把整个miniOS都改造成支持64位的程序。初步建议是，miniOS的boot和loader部分都保持不变。进入kernel后对64位的相关寄存器，例如GDT，LDT，IDT，页表等重新进行初始化
16. 移植多核版本miniOS-MP
    - miniOS-MP是基于miniOS-v1.2.7构建的多核操作系统，目前已经稳定运行在多核qemu上。
    - 本实验希望将miniOS-MP移植到miniOS-v1.3.2中，使miniOS-v1.3.2可以运行在多核qemu上。

- 任务1：移植miniOS-MP的多核启动程序到miniOS-v1.3.2中，使miniOS-v1.3.2实现多核硬盘启动。
- 任务2：在任务1的基础上，使多核的miniOS-v1.3.2可以响应多核架构下的硬盘中断。
- miniOS-MP是刘明轩的本科毕业设计，欢迎感兴趣的同学联系刘明轩询问细节。

17. 虚拟文件系统（VFS）的实现
    - 虚拟文件系统提供了对文件的统一操作模型，向上承接系统调用的接口，向下提供了inode_opeartions、file_operations、superblock_operations三组接口给实例文件系统（FAT32、OrangeFS等）。
    - 虚拟文件系统有vfs_inode、 vfs_dentry、file、superblock 四个通用对象用于实现文件通用操作。
    - 任务： 在minios中实现一个虚拟文件系统用于提供对文件的统一操作和对多个实例文件系统的管理。虚拟文件系统已经完成了设计，请根据设计文档将虚拟文件系统实现。
    - 测试：通过模拟虚拟文件系统和实例文件系统之间接口的实现，测试虚拟文件系统能否实现多文件系统的管理以及对文件操作的通用表示。