OSLab

Name:Zheng Zangwei StdID:171860658 Date:June, 2019

L1:kalloc

设计架构

在实验前看了诸如 Buddy System, Slab System 等实现方法，但最终还是选择了简单的以链表为结构的方法。

```
struct node {
  struct node *next, *pre;
  uintptr_t start, end;
};
```

每一个 node 指示了一个分配出去的空间，即每次 alloc 会额外占用 sizeof(node) 的空间用于储存链表信息。head, tail 是两个分别位于可用空间头和尾的空分配（即只占用 sizeof(node) 空间）。每次分配时，从头遍历，判断 p->next->begin-p->end 是否大于 size ，若大于则插入节点，分配空间。

该实现较为简单，利用自旋锁锁住插入节点和删除节点部分即可。

为了方便后面的实验，添加了两个宏。DEBUG 用于输出一些调试信息，CORRECTNESS_FIRST 使用只分配不释放的方法实现。

印象深刻的 Bug

(a) 考虑分配空间时为链表节点分配的空间：由于链表节点与分配的空间一起分配空间，在一开始的实现中，p->end 没有加上该部分的偏移，导致 assert(p->next->pre==p) 不满足。

(b) 把函数定义写到头文件里了：头文件应当写的是函数的声明。以及需要使用宏使头文件只被引用一次。

L2:kthreads

设计架构

我的代码中对于 irq 和 tasks 使用数组实现，有数量上限，最多支持 1024 个 tasks 的注册。os 和 kmt 模块实现起来话的时间不多，但花了大量时间 debug，而且查的欲生欲死。

最后还是成功把终端跑起来了，令人开心。

印象深刻的 bug

Bug 太多了，我挑选一些记得的和调了很久的记录。

(a) dead lock：编程中死锁出现的次数很多，但是触发死锁的原因很多，查起来就很难。主要有两个原因，一个是没有正确的关中断，另一个是 ABBA 的出现。解决这个 bug 的一个好的调试方法是先使用一个 cpu 跑着调试。

(b) 锁未初始化就使用：出问题后打印锁的名称，发现是乱码。

(c) ud2：这是一个较为精彩（坎坷）的 bug。首先是出现了 relock，根据调试信息发现是收到了错误中断导致 relock，加上判断后发现是 Invalid opcode，根据汇编发现是指令 ud2，查阅资料发现是程序中有 undefined behaviour，确认到程序中，发现是 assert 中的条件写错了。

(d) 中断中中断：不确定关中断后 os_trap 是否会来中断，为此实验了多次，并且其它 bug 的存在影响了对这个的判断，导致浪费了很多时间。

(e) spinlock 错误：在检测

(f) cpu 空转：当可运行的 threads 数量小于 cpu 总数时，存在 cpu 无法获得进程。为此特别设置了“空”进程，当没有可运行的进场时运行。

(g) debug 错误：这类错误即程序本身没有错误，但是加进去的 debug 信息反而有问题导致 bug。这分三类，第一种是 assert 中的条件本身就不应该成立（或没有考虑到所有情况），第二种是调试信息输出有误，比如给 Log 加锁后再让其输出是否关中断。第三类是改变了程序中的值。这类 bug 在所有 bug 中占比不小。

L3:vfs

使用指南

命令	功能	特点	错误信息
pwd	显示当前路径		
mount	显示所有挂载点		
cd [dir]	改变当前目录	支持任意深度嵌套，无参数返回根目录	nofile,notdir
stat file|dir	显示文件属性	针对不同类型文件输出各类信息 (对于挂载点如/proc 请使用 stat /proc/.)	noarg,nofile
ls [dir]	显示目录下所有文件	无参数显示当前目录文件	nofile,notdir
mkdir dir	创建文件夹		noarg
rmdir dir	移除空文件夹	非空文件夹无法删除	noarg, nofile, notdir, fail
rm file	移除文件	无法移除设备文件	noarg, nofile, isdir
touch file	创建文件		noarg, nofile, exists
cat file	显示文件内容		noarg, nofile, isdir
link file|dir file|dir	创建链接文件	硬链接	noarg, nofile
open file	打开文件，返回 fd		noarg, nofile, isdir
read fd	通过 fd 读取文件内容	从文件偏移量处开始读取	noarg, nofd
write fd string	通过 fd 写入文件内容	从文件偏移量出开始写入	noarg, nofd
close fd	关闭文件		noarg, nofd
lkset fd num	lseek(fd,num ,SEEK_SET)		noarg, nofd
lkcur fd num	lseek(fd,num ,SEEK_CUR)		noarg, nofd
lkend fd num	lseek(fd,num ,SEEK_END)		noarg, nofd

Table 1　命令一览

终端实现了以上指令。限于篇幅，这里不再列出初始文件。/etc/passwd 储存了一些前置内容，ramdisk1 挂载在 /mnt 处。在后面展示了一些运行样例。

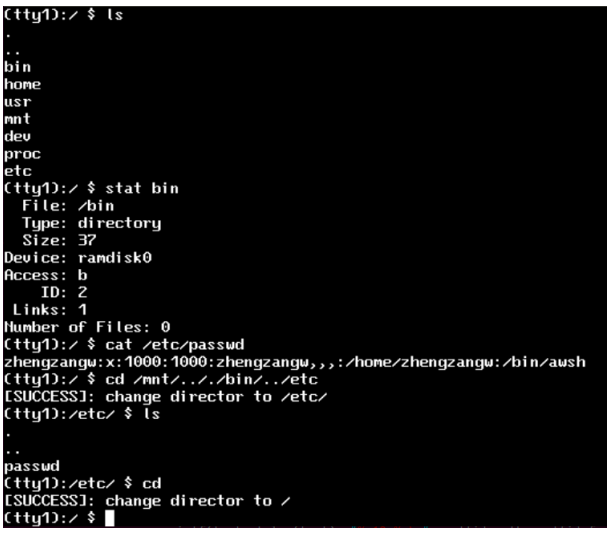
错误命令指命令可能执行失败的原因。noarg 指没有参数，notdir 指命令参数非目录，isdir 指命令参数非通常文件，exists 指文件已存在，nofile 指文件不存在。

设计架构

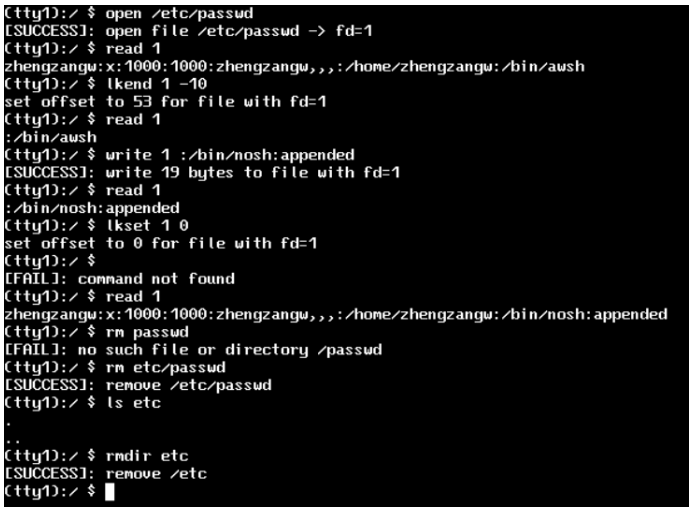
- Ext2: 根目录文件系统基本复现了 ext2 的文件系统，由 Inode Map, Data Map, Inode Table 和 Data Table 四部分组成，具体信息可 cat /proc/meminfo。通过实现读写支持非连续 Block 的读写，可以支持任意长的目录名、目录可容纳任意多文件。
- vfs: vfs 抽象了具体的 fs（ext2fs, procfs, devfs），与讲义基本保持一致。

遇到的问题

- 在编写 Ext2 文件系统时，写入错位、未写入、未清零等错误时常发生。为了方便调试，编写了一个 hexdump 函数，可以观察到磁盘中的状态，因此可以方便的检验 Ext2 文件系统的正确性
- 在将 Ext2 抽象到 vfs 架构的过程中，一开始做了很多重复工作。最后通过将 inode->fs_inode 赋值为 ext2inode 并重构代码完成了自认为不错的抽象化和解耦
- 设备读写不能加锁
- 总而言之，Lab3 在实现上难度不大，但代码量较大（1630 行），且需要设计好抽象层间的接口。



(a) basic operation



(b) read and write

```
(tty1):/ $ touch a
[SUCCESS]: create file /a
(tty1):/ $ link a /usr/mya
[SUCCESS]: create link /usr/mya -> /a
(tty1):/ $ open a
[SUCCESS]: open file /a -> fd=2
(tty1):/ $ write 2 ihopeicangetagoodgradethankyou
[SUCCESS]: write 30 bytes to file with fd=2
(tty1):/ $ cat /usr/mya
ihopeicangetagoodgradethankyou
(tty1):/ $ close 2
[SUCCESS]: close file with fd=2
(tty1):/ $ rm a
[SUCCESS]: remove /a
(tty1):/ $ cat /usr/mya
ihopeicangetagoodgradethankyou
(tty1):/ $
```

(a) read, write and link

```
(tty1):/ $ open /dev/tty1
[SUCCESS]: open file /dev/tty1 -> fd=1
(tty1):/ $ write 1 thistextwillbeshown
thistextwillbeshown[SUCCESS]: write 19 bytes to file with fd=1
(tty1):/ $ read 1
Hello World!
Hello World!
ite 19 bytes to file with fd=1
(tty1):/ $ close 1
[SUCCESS]: close file with fd=1
(tty1):/ $ link tty1 /
[FAIL]: cannot create link to file /tty1
(tty1):/ $ rm tty1
[FAIL]: no such file or directory /tty1
(tty1):/ $
[FAIL]: command not found
(tty1):/ $ rm /dev/tty1
[FAIL]: fail to remove /dev/tty1
(tty1):/ $
```

(b) device

```
(tty1):/ $ mount
PATH        DEVICE
/           ramdisk0
/mnt/       ramdisk1
/dev/
/proc/
(tty1):/ $ ls /dev/.
.
..
tty1
tty2
tty3
tty4
ramdisk0
ramdisk1
(tty1):/ $ stat /proc/897
  File: /proc/897
  Type: normal file
  Size: 0
Device:
Access: 1
    ID: 3
 Links: 1
(tty1):/ $ cat /proc/897
    name: shell-task-1
      id: 897
runnable: 1
sleeping: 0
(tty1):/ $
```

(a) proc

```
(tty1):/ $ cat /proc/cpuinfo
cpus amount: 1
(tty1):/ $ cat /proc/meminfo
 heap start: 0x200000
   heap end: 0x8000000
  heap size: 132120576

Root Filesystem Info
         FS:ext2
 Block Size: 0x400
 IBlock Num: 10
Inode Start: 2
 Inode Size: 0x80
 Data Start: 12
(tty1):/ $ ls /proc/.
.
..
cpuinfo
meminfo
107
331
454
507
638
897
(tty1):/ $
```

(b) info