OSlabs Report

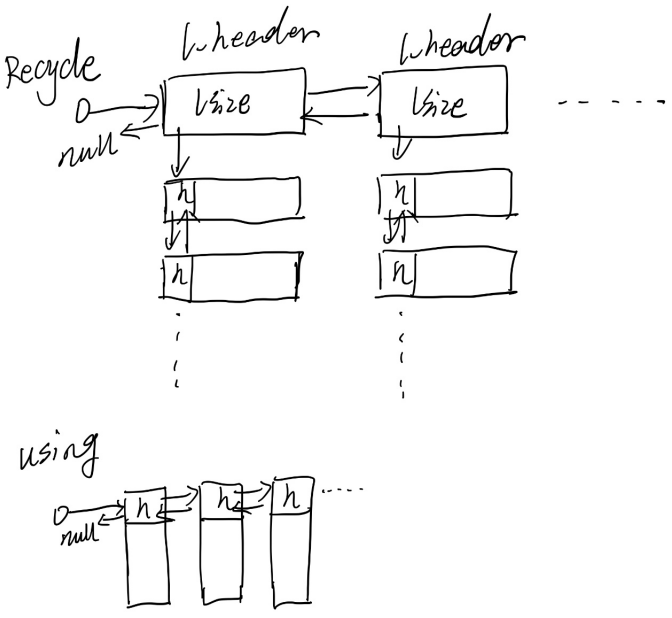
姓名：张灵毓　　**学号：**171240524

2019 年 6 月 5 日

1　L1:kalloc/kfree

1.1　框架设计

框架设计没有区分小内存分配回收和大内存分配回收。
分配空间都分配 2^kB
采用 slab 的思想, 不过对空闲的空间不进行合并, 用两个 list:using 和 recycle 分别表示当前在使用的内存之和之前使用过并被回收的内存, 如图所示,recycle 所指的链表的每一个 l_header 中记录其所指的内存链表中每一个内存块的大小 lsize, 而 using 指针则直接指向若干个正在使用的内存块



分配: 先根据所需要的内存大小选取适当的 k 满足 $size > 2^{k-1}$ 且 $size \leq 2^k$ 记 $2^k = alloc\_size$, 然后遍历 recycle 链表, 找到满足条件的 lsize = alloc_size, 且 $l\_header->begin_h \neq$ NULL 的结点, 即表示此处存在未分配的内存块, 将第一个符合条件的内存块分配即可并将这一个内存块从 recycle 中删除, 插入 using 链表, 如果没有符合条件的 l_header, 那么就通过 create 同 _heap 中分配一块新的空间来使用 (空间大小为 $4 *$

$alloc\_size$, 每次分配 4 块, 提高分配效率)
内存回收: 只需要将内存块从 using 中删除, 插入到 $lsize == alloc\_size$ 的对应 l_header 指向的链表中, 留下次使用即可.

1.2 something about bug

在使用自己实现的 test 函数时, 创建一个 void * 的数组用来得到 kalloc 的返回值, 如果将这个数组定义在 test 函数外面, 即全局变量, 则无论多大, 测试能够通过 (测试了很多次...) 如果放在函数里面, 则超过一定大小就会出现神奇的问题, 不超过这个大小也可以通过测试 (测试了很多次...), 一开始也不知道为什么通过检测否决了爆栈的可能性, 从而转向考虑是否是结构体对齐问题, 检查代码中的结构体, 在 heap 中都按照 16 字节对齐, 所以没有问题, 在最近的测试中, 发现, 如果在 pmm->init 里面打印 pm_start 的值, 为正常的 0x200000, 而在 kalloc 里面打印变成了 0, 说明还是一开始想到的问题, 即爆栈了, 将静态数据都给覆盖掉了, 从而导致问题.

2 L2:Kthreads

2.1 Bug,bug and bugs!

L2 实验由于比较困难, 遇到的 bug 也有点多. 主要几个印象深刻的点在于:

- 在 L1 中由于没有中断的问题, 实现的自旋锁就特别的简陋, 而在 L2 中加上中断后, 原来的锁对开关中断的控制就不再能保证正确性, 所以得加以更新, 可以想 xv6 中的自旋锁一样, 创建 cpu 结构体, 在结构体中记录 intena 等信息来帮助控制开关中断.
- 在 os_trap 函数中, 虽然上了锁, 但还是可能被 I/O(键盘) 设备的中断所影响, 我的实现是刚进入 os_trap 函数就 spinlock(traplock), 然而可能出现的问题是刚上好锁后, 马上收到了 I/O 设备的中断请求, 再一次进入了 os_trap 这个函数, 于是此时 cpu 会尝试重复获取 traplock 这把锁, 从而出现错误, 而解决的办法就是在获取锁之前先检查一下当前 cpu 是否已经获得了这把锁, 如果获得了就不再去获得锁.
- 关于信号量的问题, 讲义中提到在 sem_wait 中要先把释放锁之后再去 yield, 然而在释放锁到陷入内核之间可能会被中断打断, 这个中断可能正好 signal 同一个信号量, 将这个线程唤醒, 而在 yield 的时候并不知道, 从而可能会出现错误. 解决的办法是可以在 os_trap 中加一个判断如果发现其状态为已经被唤醒的状态, 则不进行线程的切换, 使得这个线程继续运行.
- 关于是否跨核调度的问题, 到目前为止几位同学怀疑框架代码中在切换上下文的时候出现了小问题导致实现的跨 cpu 调度出现诡异的情况, 所以为了避免这种情况, 可以在线程创建的时候就平均分配给 cpu, 然后 cpu 在调度时只能调度属于自己的线程.

2.2 鸣谢

在此特别感谢欧先飞和赵士轩两位助教给予的帮助!

3 L3:Virtual Filesystem

3.1 设计实现思路

实现的 vfs 采用的是类似于 linux 基础文件系统 ext2. 将磁盘分为 inode 区和 block 区,inode 里面存 blkfs 文件的 inode,block 区里面存 blkfs 里面的文件内容包括普通文件和目录文件实现中并没有采用对 inode 的操作, 而是把操作都集中在 vfs 的 API 中.L3 自我感觉没有 L2 这么难, 但是自己的实现方式码量比较大.

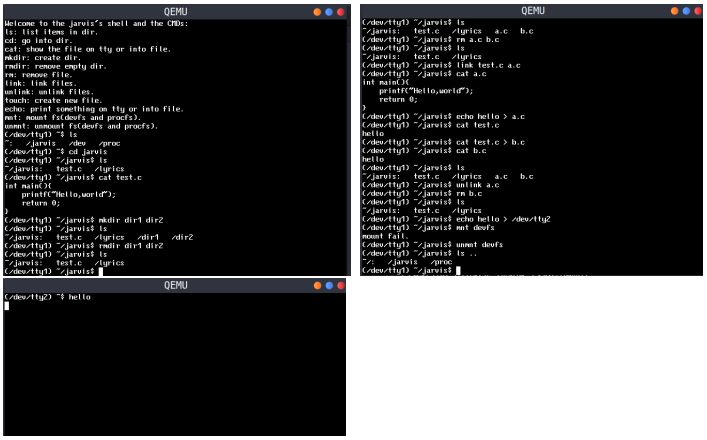
3.2 SHELL 进程操作示意

在自己实现的 shell 中, 实现了如下 cmd: help,ls,cd,cat,mkdir,rmdir,rm,link,unlink,touch, echo,mnt,unmnt, 详细解释如下:
help 命令会输出可以使用的 cmd 和简单解释.
ls 指令同 linux, 支持多路径,ls dir1 dir2 ... 会输出多个路径中的内容. 示例:ls ,ls .. ,ls jarvis,ls jarvis/lyrics
cd 指令同 linux, 特别的实现了同 linux 中只输入 cd 返回到根目录. 示例:cd ,cd .., cd jarvis, cd jarvis/lyrics
mkdir 指令同 linux, 同样支持多参数,mkdir a b ... 将创建 a b ... 文件夹. 示例:mkdir dir1, mkdir jarvis/dir2, mkdir dir1 jarvis/dir2
rmdir 指令同 linux, 同样支持多参数,rmdir a b ... 将删除空文件夹 a b ..., 非空目录不能删除. 示例:rmdir dir1, rmdir jarvis/dir2, rmdir dir1 jarvis/dir2
rm 指令用于删除文件, 同样支持多参数,rm a b ... 将删除文件 a b ...(未实现如-rf 参数). 示例:rmdir file1, rmdir jarvis/file2, rmdir file1 jarvis/file2
link 指令用于创建某个文件的链接,link oldfile newfile, 将创建一个新 newfile 其指向 oldfile. 示例:link jarvis/test.c a.c
unlink 指令用于解链接,unlink file1, 如果 file1 之前通过 link file file1 链接过, 将在路径上删除 file1, 如果没有经过 link 则效果同 rm. 示例:unlink a.c, unlink jarvis/test.c
touch 指令用于创建某个文件, 同样支持多个文件参数. 示例 touch a.c, touch jarvis/a.c, touch a.c jarvis/a.c
echo 指令意思同 linux, 将 echo 的内容写到 tty 或者 blkfs 中某个文件. 示例: echo hello, echo hello > /dev/tty(id), echo hello > a.c, 特别地 echo hello > b.c world, b.c 中将会存储 hello world.
mnt 指令用于挂载 devfs 或者 procfs 中某一个, 示例:mnt devfs, mnt procfs.
unmnt 指令意思显然, 用法同 mnt.

特别说明:
ls 显示中名字第一个为/的表示目录, 否则表示文件.

mkdir,rmdir,link,unlink 只支持在 blkfs 中操作, 即不支持在/devfs,/procfs 操作.
实现了绝对路径和相对路径, 在当前路径下输入的路径如果以/开头, 则说明是绝对路径, 无/开头则表示当前路径下的相对路径.
有些错误信息会显示在 tty 中, 有些则会打印在实际终端中, 比如 permission denied. no such a file.

测试图示例:



echo hello > /dev/tty2 示例