OSLAB-6实验报告

`OSLAB讲义`

> 南京大学匡亚明学院 刘志刚
> 学号：141242022
> 邮箱：njuallen@foxmail.com

实验进度

总体进度：完成了shell lab，实现了所有功能。

- [x] ls(支持所有选项)
- [x] cat
- [x] touch(支持`-c`选项)
- [x] echo
- [x] 重定向(支持`<`，`>`以及`>>`，支持通用的重定向)
- [x] cd(支持多级目录，以及相对路径访问)
- [x] 执行程序
- [x] readline(支持上下方向键操作得到历史输入命令)
- [x] ctrl-c

文件系统的时间

要完整地支持touch的话，则文件系统中必须要能记录时间。对于每个文件，我们记录三个时间，分别是创建、访问和修改时间。创建操作会初始化创建时间，read操作会更新访问时间，write操作会更新修改时间。

另外一个问题就是，对于每一个操作，我们要能得知其时间。对于时间的获取，通过查阅资

料，我采取的是通过读取cmos中的相关寄存器来获得时间。但需要注意的是，我们获得的时间是标准时间，会比我们的主机时间慢8个小时，因为我们处在东八区。因此，在创建文件系统时，我们给所有文件初始赋的时间必须是标准时间，这样子操作才一致。我认为是由于qemu的设置的问题，才导致其虚拟出的cmos时间是标准时间。

---

malloc

由于我们这次涉及到大量的关于路径的字符串处理，因此如果没有动态内存分配，处理起来会很麻烦。我实现了一个最简单的malloc、free。我们的堆从一个固定的位置开始生长，每来一个请求，如果当前堆上剩余空间不够，我们就再分配一些页供使用，这样子就完成了堆的生长。至于free，我们不进行任何操作。

这样子内存利用率极低，且是有内存泄漏的，但由于我们的用户程序使用的动态内存都很少，因此这样子是没有问题的。

相对路径

工作目录的处理

要支持相对路径访问，我们首先就要支持当前工作目录这个概念。我们的当前工作目录是在lib中实现的，最后存在的形态是程序运行时内存空间中的一个char型数组。每当用户发起一个文件系统相关的操作，在库中，我们根据用户给的路径是不是以`/`开头来判断其是绝对路径还是相对路径。如果是相对路径，我们就在lib中将其翻译(包括拼接，特殊路径的处理)成绝对路径，再进行相应的操作。另外，所有用户的默认起始工作目录都是`/`。

工作目录的继承

由于我们肯定会切换目录，并到那个目录下进行操作。切换目录后，shell创建的子进程必须继承shell的工作目录。否则诸如`cd dir;ls .`这样的操作表现就不太正常。

由于我们的工作目录本质上lib中一个static类型的char数组，同时我们的lib又是静态链接的，因此工作目录的继承是一个很大的问题。

1. 对于fork
   由于fork是共享内存空间的，因此内存空间中的工作目录数组自然而然地被共享了。
2. 对于spawn(fork + execve)
   JOS中没有采取传统的fork + execve的组合，而是通过spawn将这两项工作一步到位。在这个过程中，除了文件描述符页被共享之外，其他任何内存都没有被共享。因此spawn之后，工作目录自然也没被共享。为了能解决这个问题，我们为spawn加上参数pwd，将其作为一个环境变量压到新的进程的栈上。新的进程在一开始时就检查环境变量是否是NULL，如果不是，就把它当作当前工作目录。这样子，我们解决了这个问题。

`.`与`..`

由于我们的文件系统本质上不是基于inode的，只支持单链接。因此要在文件系统层面，保存住当前目录与上一级工作目录是很困难的。因此，我将对这两个特殊路径名的处理放在lib中完成，在路径翻译时一并完成，通通化成绝对路径，再交由文件系统处理。

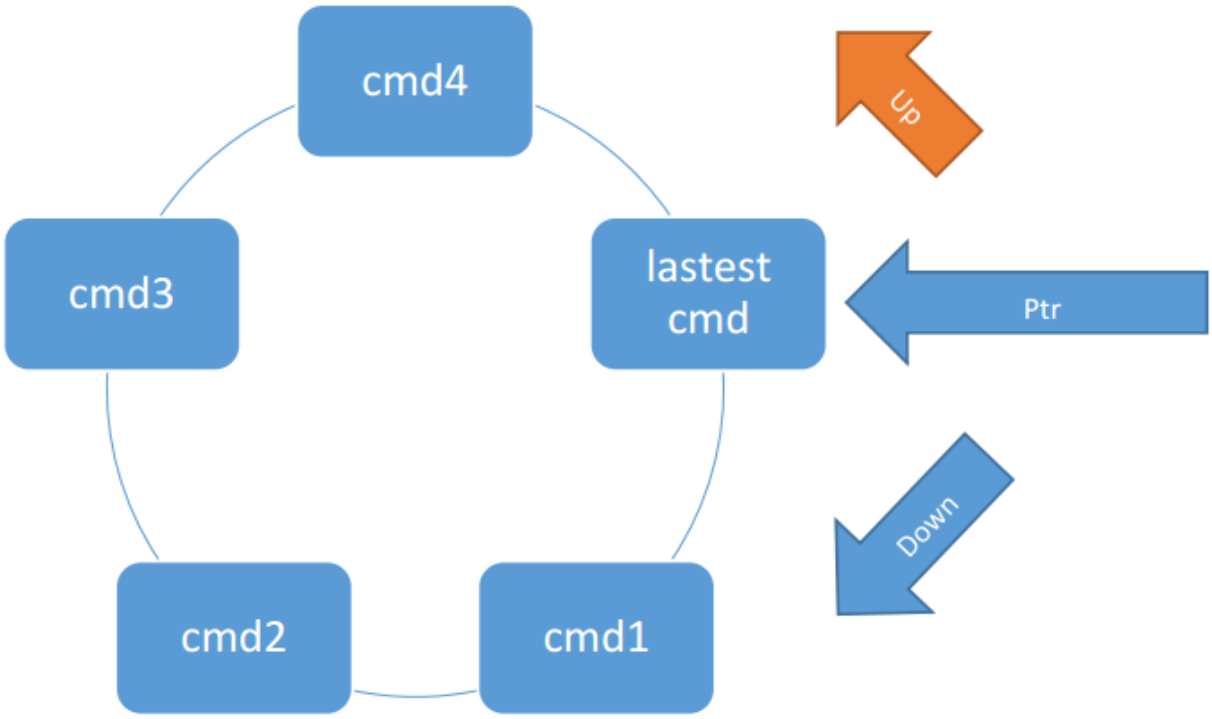
相对路径的处理全部是在lib中完成的，这一切对用户都是透明的。另外我提供了pwd和chdir这两个库函数，用户程序通过调用这两个函数可以知道自己当前的工作目录，并可以进行修改。在shell中，shell提供了pwd和cd这两个内建指令，可以用于输出当前工作目录及切换工作目录。

效果展示

```
$ pwd
/1/2/3/
$ cd ../../
$ pwd
/1/
$ cd ./
$ pwd
/1/
$ ls
:
    2
$
```

readline

在有动态内存分配之后，readline的实现变得简单了许多。首先，我的readline背后的数据结构是这样的：



但在实现过程中涉及到一个坑是JOS的键盘输入有两个来源，直接对着终端敲出来的是串口输入，而从弹出的qemu窗口中敲出的是键盘输入。尤其坑的是在串口输入和键盘输入下，上下

左右这类特殊键的键值不一样！因此为了方便处理，我们只处理了串口输入，你就不要从弹出的qemu窗口中输入了。最好的是运行时加上nox选项，不使用x界面。

在按下上下键之后，如何更新屏幕输出这是个问题。助教大人的建议是直接另起一行，但我感觉这样太不美观了。于是我采取的策略是手动记录光标位置，每输出一个字符，光标位置就进行相应更新。当我们要更新屏幕输出时，就通过退格加输出空白符的方式，把这一行清空。经测试，实际效果很好。但就按上下键而言，显示效果和一般的shell没有区别。

对于左右键，我没有做特殊处理，不支持命令行上通过左右键移动，并进行编辑。对于左右键，我只是输出默认的乱码`[D`和`[C`。

---

ctrl-c

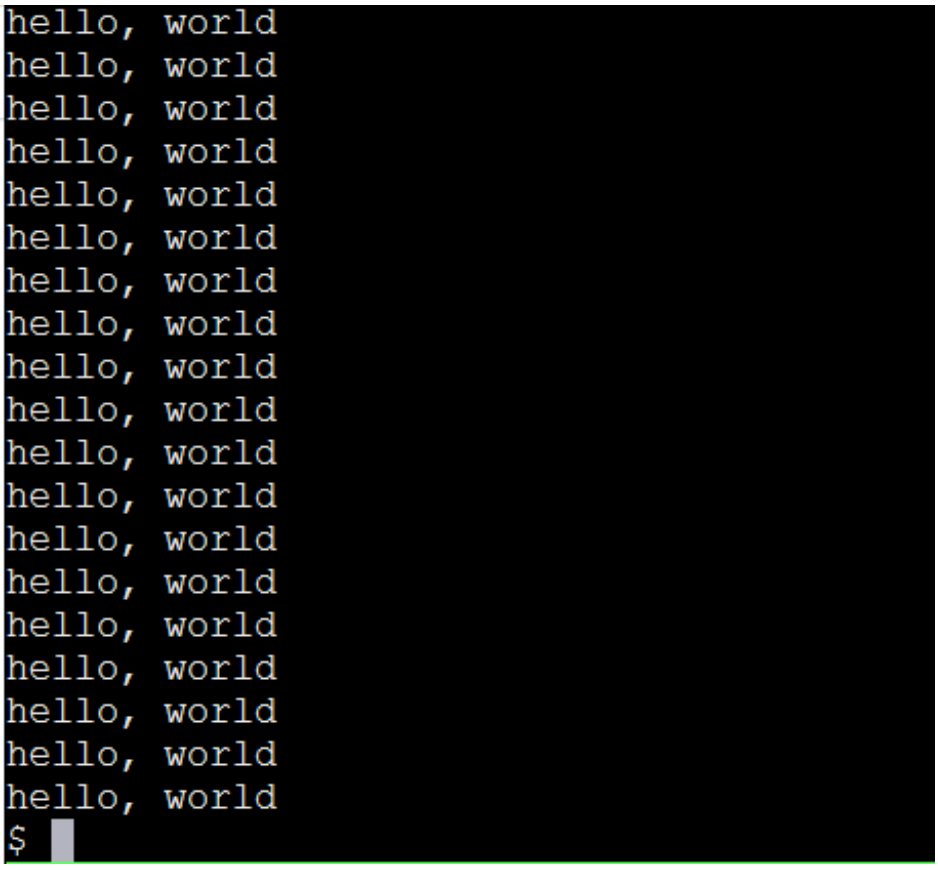
怎样通过ctrl-c来中断程序是另外一个问题。我思考了一下几种解决方案。

1. shell进程不停地读，如果发现有ctrl-c出现，就把相应的进程杀死。
   这是不靠谱的，因为如果shell调用的用户程序就在不停地读，则该进程有可能无法被杀死。
2. 实现简单的信号机制
   即操作系统自身截获到ctrl-c信号后，就将控制权转移到lib的代码中(可以采用类似于JOS中pgfault_upcall)的方式。再在lib中，将控制权转移到对应的handler上。每一个信号都有一个默认的handler，对于ctrl-c这类的信号，默认的应该就是终止运行。当然，每个进程也可以通过signal函数来自定义对应信号的handler。
   这是一个比较靠谱的方案，但还有问题：
   1. 信号发送给谁，按理说信号应该发送给同一个session下的进程。信号从哪个session来，信号就发送给那个session下面的进程。但现在我们的内核根本就没有session的概念，因此我们就选择发给所有进程。
   2. 如果有进程正在执行，是把他立即撤下来，还是先标记，后来再撤下。如果选择立即撤下，在单核情况下还可以。但在多核情况下，如果目标进程在另一个核上运行，我们就要通过发送核间中断才能将其撤下。而即使发送了核间中断，由于我们的kernel现在是不可重入的，所有内核即使在这个核上触发了核间中断，另外的核也没法处理。

因此，我的最终方案是：内核截获到ctrl-c之后，将所有进程标记为接收到sigint信号。等到下一次调度时，如果发现某一个进程有sigint信号待处理，就进行相应的处理。

信号的处理我也进行了简化，没有采取到转到用户态的库的方式，也不支持自定义信号处理函数。进程通过sys_ignore_sigint和sys_accept_sigint来通知内核自己是否需要这个信号。如果接收信号，则调度时若发现进程控制块被标记了sigint信号，则杀死该进程。否则，忽略该信号。所有进程默认都接收这个信号，因此所有进程在接收到ctrl-c之后都会被杀死。为了保证shell以及其他一些重要进程存活，我修改了init、shell以及fs的代码，让他们都调用一遍sys_ignore_sigint。

要对ctrl-c的效果进行演示，我的文件系统中准备了一个程序foreverhello，它会不停地输出"hello, world"，按下ctrl-c之后，我们可以看到它终止了。



对实现的指令的说明

ls

ls支持所有选项，但它的表现与我们常用的ls有些不同。具体体现在，输入`ls .`和`ls ..`时，不会有任何输出，因为`.`和`..`是隐藏文件。你必须输入`ls -a .`和`ls -a ..`才能有输出。

ls的`-h`选项，将文件大小转换成B、KB、MB等单位输出，我是采取的向下取整的方式。例如3.2MB的文件，我只会输出3MB。因为JOS中没有实现对浮点数的格式化输出。

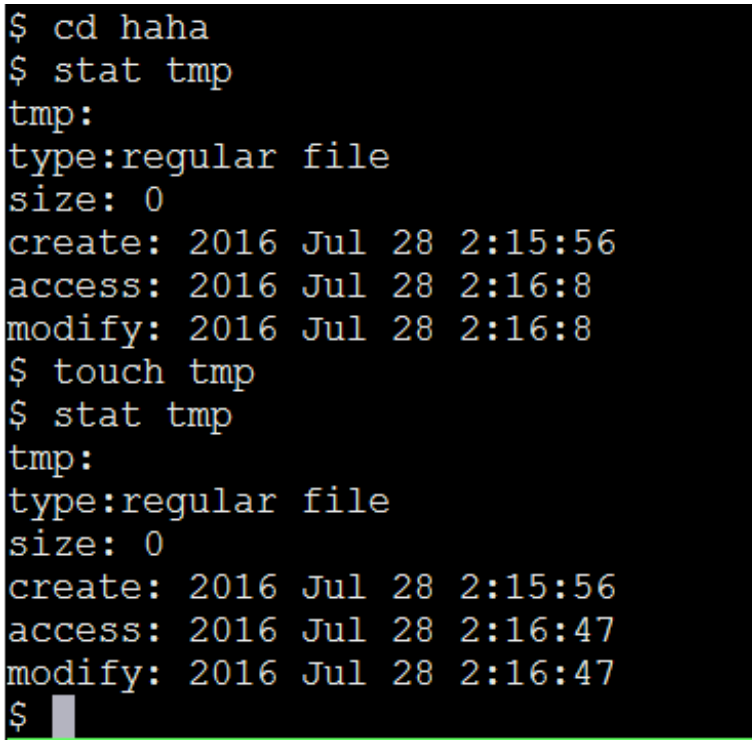
touch

我的touch是通过找到文件的父目录，并修改存放在其中的文件元数据来实现的。

而找到父目录这个操作我的实现也尤其简单，直接从后向前找到第一个`/`，并将那之前的内容作为父目录，之后的内容作为文件名。因此，我不能处理以`/`结尾的路径名。如果你想要touch一个目录，请输入`touch path/dir_name`而不是`touch path/dir_name/`。相应地，由于这个实现上的原因，我们无法touch根目录。

为了便于展示touch的效果，我另外还提供了一个指令stat，用于查看所有和这个文件相关的信息。包括大小，类型，时间戳等。

使用touch和stat的效果如图所示：



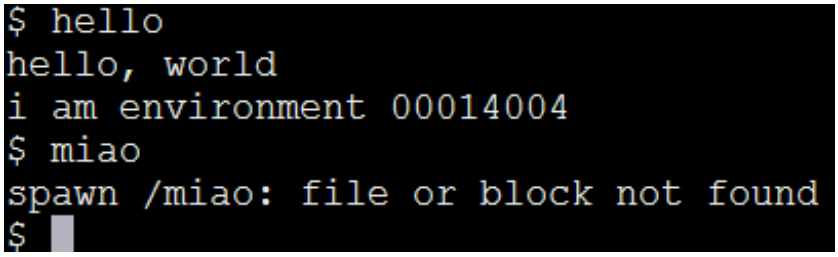
注意：以上的时间均是标准时间，不是东八区的时间。毕竟我是不会在凌晨两点写代码的。

cd

一般来说cd命令都是shell的内建指令，因此我将其实现成了shell中的一个函数。它就干一件事，那就是在shell中调用chdir函数。

执行程序

JOS的shell已经支持这项功能了。程序默认是从根目录下开始搜索。
效果如图所示：



mkdir

实现了简单的mkdir，用于在当前目录下创建一个新的目录，不支持任何选项。

运行说明：
在命令行输入`make run-icode-nox`即可。注意，这不会弹出qemu窗口，因此文件系统中的gobang游戏将无法运行。
退出请按ctrl + a + x。