OS Lab5 - ExprimentReport

1. 思考题

5.1查阅资料，了解 Linux/Unix 的 /proc 文件系统是什么？有什么作用？ Windows 操作系统又是如何实现这些功能的？proc 文件系统这样的设计有什么好处和可以改进的地方？

/proc 文件系统是一个虚拟文件系统，它将系统运行的状态的详细信息组织成一个虚拟文件系统的形式，通过它可以使用一种新的方法在 Linux内核空间和用户间之间进行通信。在 /proc 文件系统中，我们可以将对虚拟文件的读写作为与内核中实体进行通信的一种手段，但是与普通文件不同的是，这些虚拟文件的内容都是由内核动态创建的。其内的文件也常被称作虚拟文件，并具有一些独特的特点。例如，其中有些文件虽然使用查看命令查看时会返回大量信息，但文件本身的大小却会显示为0字节。此外，这些特殊文件中大多数文件的时间及日期属性通常为当前系统时间和日期，这跟它们随时会被刷新(存储于RAM中)有关。查看这些文件可以让我们了解系统的一些基本信息和此时CPU的状况。

WIN中使用WIN API即系统调用的方式返回系统的相关信息。但将这些系统信息打包提供给用户的方式似乎更像WINDOWS的资源管理器

5.2如果我们通过 kseg0 读写设备，我们对于设备的写入会缓存到 Cache 中。通过 kseg0 访问设备是一种错误的行为，在实际编写代码的时候这么做会引发不可预知的问题。请你思考：这么做这会引起什么问题？对于不同种类的设备（如我们提到的串口设备和 IDE 磁盘）的操作会有差异吗？可以从缓存的性质和缓存刷新的策略来考虑。

会导致外部设备的数据与CPU获取的数据间不同步。对于其他地址空间的访存，都完全由CPU完成，这些地址空间中的值都是由CPU修改的，因而Cache可以保证数据访存的一致性，而外设对应的地址空间上的数据，会被外部设备自己更改，而CPU只有通过直接读取这些外设地址上的信息才能获得正确的数据。

有区别，如终端串口设备是字符设备，没有经过缓存，直接以字符(字节)为单位进行读写。而IDE磁盘设备属于块设备，经过缓存并通过缓存的刷新访存相应的外设。

5.3 一个磁盘块最多存储1024 个指向其他磁盘块的指针，试计算，我们的文件系统支持的单个文件最大为多大?

4096B * 1024 = 4MB

5.4 查找代码中的相关定义，试回答一个磁盘块中最多能存储多少个文件控制块？一个目录下最多能有多少个文件?

一个磁盘块中最多能保存`FILE2BLOCK`=16个文件控制块。一个目录文件最多存储1024个指向其他磁盘块的指针所以最多保存16*1024个文件控制块，因此一个目录下最多有16*1024个文件

5.5 请思考，在满足磁盘块缓存的设计的前提下，实验使用的内核支持的最大磁盘大小是多少?

fs/fs.h中定义DISKMAX = 1 GB

```
/* Maximum disk size we can handle (1GB) */
#define DISKMAX 0x40000000
```

5.6 如果将DISKMAX 改成0xC0000000, 超过用户空间，我们的文件系统还能正常工作吗？为什么?

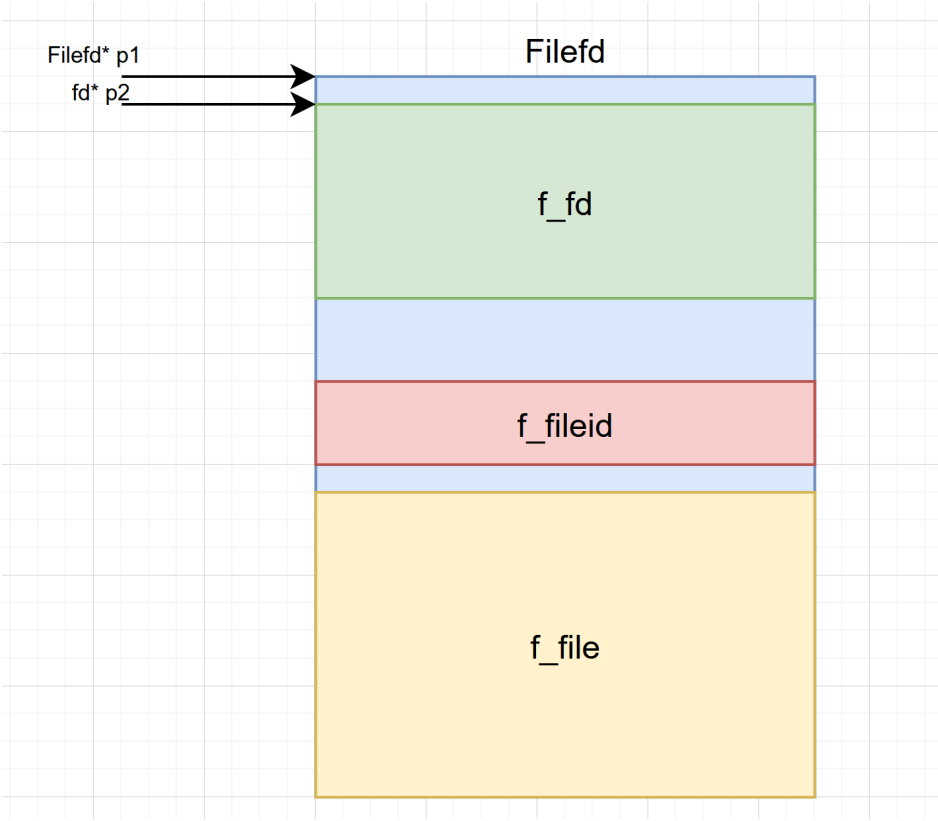
不能，超过用户空间会导致改写内核重要数据(如内存的页式内存管理、进程管理等等)，导致该文件控制进程本身的异常。

5.7 阅读user/file.c，思考文件描述符和打开的文件分别映射到了内存的哪一段空间

在打开文件时，文件描述符的位置被函数`fd_alloc`安排。在该函数中，我们发现文件描述符被放置在内存中的`FDTABLE`域。即`0x60000000`以下的一段空间。

打开的文件与文件描述符被共同装在一个`Filefd`结构体中，位于文件描述符所独占的那一页中。

5.8 阅读 user/file.c，你会发现很多函数中都会将一个 struct Fd* 型的 指针转换为 struct Filefd* 型的指针，请解释为什么这样的转换可行。



Filefd的结构体中第一个结构体成员就是Fd结构体，所以一个处于Filefd结构体中的fd结构体的指针所指向的地址与其所处的Filefd结构体指针所指向的地址是相同的。而C语言又是弱类型语言，支持结构体指针的强制转化，因此可以直接进行这样的转化。

5.9 请解释Fd, Filefd, Open 结构体及其各个域的作用。比如各个结构体会在哪些过程中被使用，是否对应磁盘上的物理实体还是单纯的内存数据等。说明形式自定，要求简洁明了，可大致勾勒出文件系统数据结构与物理实体的对应关系与设计框架。

Fd 是用户空间保存的文件结构。其中`fd_dev_id`指示了该文件所处的设备，`offset`为文件偏移指针，`fd_omode`指示了文件的访问权限。全部都为单纯的内存数据。

Filefd是文件系统用到的包含一个FCB和一个Fd的结构体，其中的FCB可以对应到磁盘上的文件块物理实体(在内存的块缓存中但在与磁盘同步时将从磁盘装入或写回)，其他的为内存方便维护文件的内存数据。

Open用于抽象化记录打开文件这一行为，其中`o_file`指向具体的`file`。`o_fileid`为`file`的id。`o_mode`为打开权限，指只读，只写等。`o_ff`：读或写的当前位置，指偏移量。

5.10 阅读serv.c/serve函数的代码，我们注意到函数中包含了一个死循环for (;;) {...}，为什么这段代码不会导致整个内核进入panic 状态?

该进程在调用`sys_ipc_recv`时，会被设置为`ENV_NOT_RUNNABLE`并调用`yield`主动让步，只有当其接收到了信息，才会被发送消息的一方重新设置位`ENV_RUNNABLE`转化为运行态。这类似于`wait-notify`机制从而避免了忙等待，因而不会导致整个内核进入panic状态。

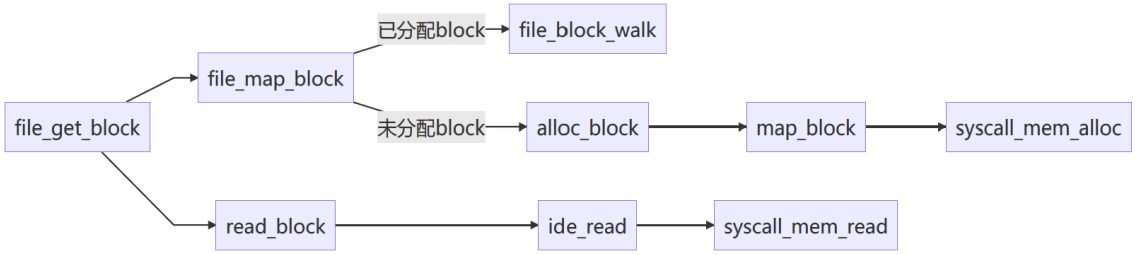
5.11 观察user/fd.h 中结构体Dev 及其调用方式。综合此次实验的全部代码，思考这样的定义和使用有什么好处

```
struct Dev {
 int dev_id;
 char *dev_name;
 int (*dev_read)(struct Fd *, void *, u_int, u_int);
 int (*dev_write)(struct Fd *, const void *, u_int, u_int);
 int (*dev_close)(struct Fd *);
 int (*dev_stat)(struct Fd *, struct Stat *);
 int (*dev_seek)(struct Fd *, u_int);
};
```

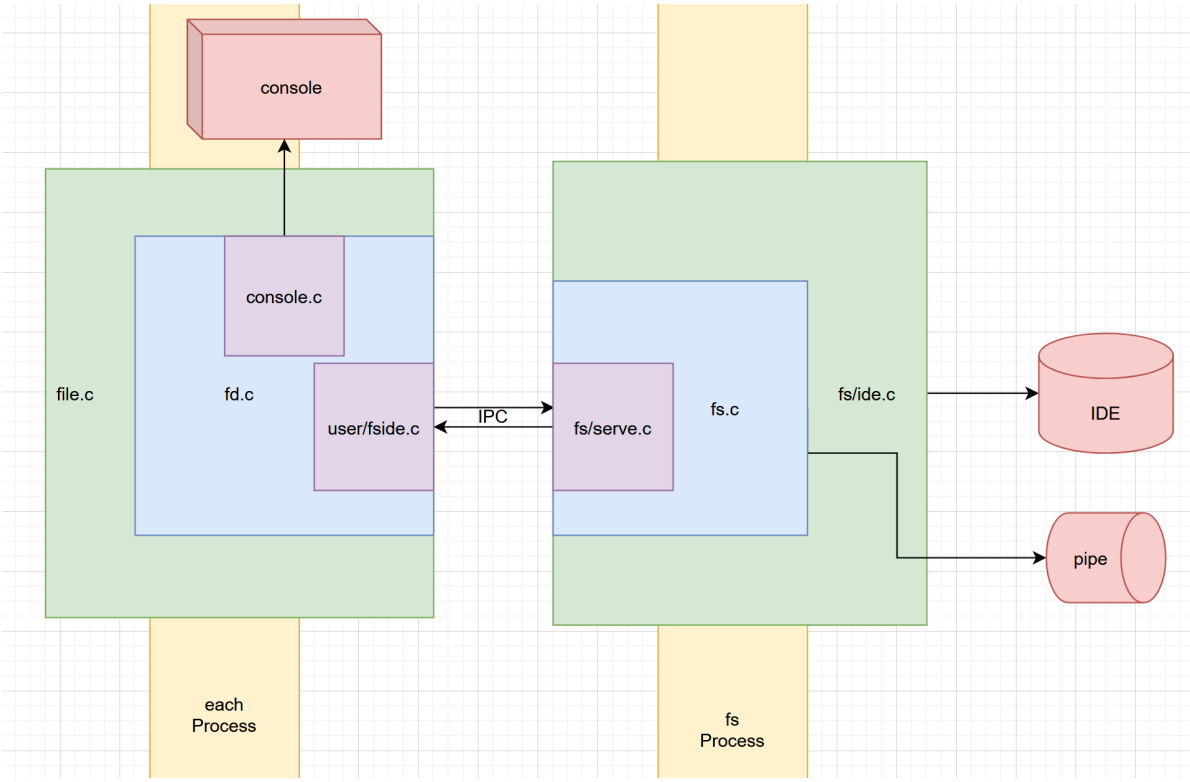
这样的定义使得对于不同的类型设备(本实验中为磁盘文件设备、终端、管道文件)可以执行不同的文件增删改查的相关命令但对于调用者而言是统一的(OO思想)，这样可以提高代码的扩展性，如果需要更改函数，只需添加函数后再相应的定义中注册函数指针即可。

2. 实验难点图示

1. file_get_block函数调用关系



2. 用户进程-fs进程-三种文件的关系

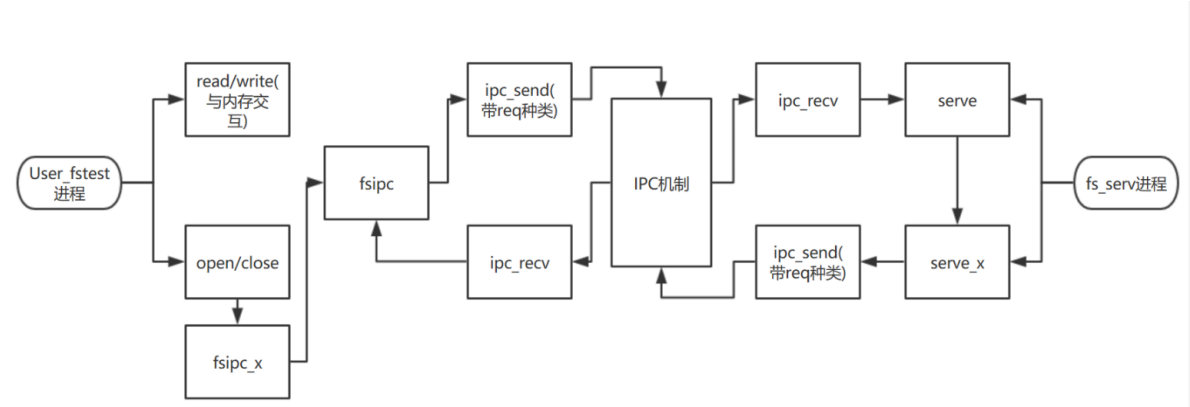


3. 与IDE交互时的更加细致的关系

文件系统的用户请求IPC主要有以下几种：

```
#define FSREQ_OPEN  1
#define FSREQ_MAP   2
#define FSREQ_SET_SIZE  3
#define FSREQ_CLOSE 4
#define FSREQ_DIRTY 5
#define FSREQ_REMOVE    6
#define FSREQ_SYNC  7
```

请求的发送由 `file/fsipc.c` 函数 `fsipc` 实现，其接受的参数 `type` 即为请求的类型，对于每个类型在下图中以 `x` 表示



3. 实验总结

lab5建立文件系统的过程实际上是lab2、4的一次综合应用，相关内容分别为

- 建立文件缓存块与虚拟页式内存空间的映射关系 - lab2
- 通过进程间通讯使得一般进程从文件管理进程那里获得文件访存的服务 - lab4

当然，IO驱动和fd等抽象概念的实现是lab5独有的，由于文件系统本身代码量巨大，理解各部分之间的关系还是要花费一定的时间和精力的。但在理清实际的脉络后，写出的程序正确性较好(也许是lab4将之前lab2、3祖传的bug都de完了~)。课上测试中：

- lab5-1-exam：通过类似ide驱动的方式获取gxemul自带的时钟设备的时间，并实现与console交互字符串。
- lab5-1-extra：实现控制台中断，模仿其他中断合适的建立中断响应的机制(主要统计中断出现的次数等信息)。
- lab5-2-exam：实现用户通过给定路径创建文件(普通\目录)的接口，并通过IPC的方式在fs进程的server中予以实现。
- lab5-2-extra：进一步实现主动创建目录的用户接口，并扩展原有的文件系统使其支持一个目录中同名的目录和文件。

lab5-1较为顺利的通过了。但在lab5-2中由于自己没有认真阅读全部的代码，一开始以为要自己独立实现字符串分析并walk文件系统创建文件，没有发现这部分内容已经实现好了，在助教给出提示后才得以发现，导致做到extra时时间不足，没有通过。

4. 指导书反馈

- 混淆了扇区、文件控制块(FCB)、磁盘块的概念。
- 和之前的lab的指导相比，这次的指导感觉相当不连贯，而且对于每一个execise也没有给出提示，也许是鼓励我们自己看代码?
- 虽然代码较多，还是希望给一些比较复杂的代码添加下注释，较为重要的函数也给出一个overview。
- 之前的代码中都有提示要填写的函数对应`execise x.y`，这次却没有，感觉很不方便。
- 在初始化bitmap时，将大部分bitmap置1的方法如下

```
for(i = 0; i < nbitblock; ++i) {
    memset(disk[2+i].data, 0xff, NBLOCK/8);
}
```

而实际上一个data的位图应该不止`NBLOCK/8`字节，因此这部分代码应当有误。

5.残留难点

我们实验中实际上是通过fsformat使用各种linux的系统调用创建了一个虚拟的磁盘写入到gxemul中进行仿真，在实际的操作系统中这一将文件系统写入磁盘，和从磁盘初始化文件系统的过程又应当如何实现?

实际上的文件不止读写的文本文件，至少应当有可执行文件，那么对于这样的文件应当如何装入内存镜像lab3那样运行？在更高级的系统中还会存在链接文件，和文件系统的权限等概念，又应当如何在MOS中实现扩展和维护?

结构体Fd是Ffd的一部分，那为什么还要留下Fd?