北京郵電大學

Beijing University of Posts and Telecommunications

操作系统课设

操作系统课程设计

概要设计说明书

组号：8

时间：2022/04/06

目录

1 总体设计

1.1 概述

1.1.1 功能描述

我们实现的模拟操作系统，即 PD-OS（Paper Doll Operation System）从操作系统课设的目标出发，专注于操作系统的上层逻辑结构和操作系统主要功能的实现及相互配合。我们把操作系统所能实现的主要功能分为五块，分别是：UI、进程管理、内存管理、文件管理和设备管理。这其中，UI 采用图形化界面为用户提供使用渠道；进程管理处理内存中的进程，实现进程状态的转换、timer 和中断处理；内存管理实现对内存存储空间的管理；文件管理对文件和目录进行处理；设备管理维护 IO 设备的队列和磁盘的管理。

从用户的角度来讲，PD-OS 使用图形化的界面，为用户提供进程的创建与中断（将以运行数个可执行程序的形式实现）、文件的创建，修改和删除、系统内各个资源的监控。

1.1.2 运行环境

PD-OS 的运行环境为 Windows 10 操作系统。

1.1.3 开发环境

采用 Visual Studio 2019 平台上的 C++语言完成程序，同时，为了实现多人的协同开发，我们采用 CODING 平台：一个提供 Git/SVN 代码托管、项目协同、测试管理等在线工具的一站式软件研发管理协作平台进行代码的共享和项目的进程协作。

1.2 设计思想

1.2.1 软件设计构思

PD-OS 的主要目标是对操作系统的上层逻辑结构进行模拟。我们最初对《30 天自制操作系统》等相关书目进行了阅读，发现市面主流的教程着眼于硬件的适配，对于汇编语言和磁盘等硬件技术性质的功能着墨较多，而对于操作系统实际的逻辑功能等不够关注。而在仔细实验指导书后，我们决定我们的模拟操作系统 PD-OS 主要是为了对操作系统的上层功能进行模拟和演示，完成进程管理和调度、内存管理(存储分配与回收，进程交换)、时钟管理、中断处理、用图形界面展示多道程序并发执行的过程等操作系统的基本功能，从而加深理解操作系统的基本功能、原理和工作机制。

1.2.2 关键技术与算法

进程管理：采用 FCFS 调度算法、SJF 调度算法、多级反馈队列调度算法；
文件管理：索引分配；FCFS, SCAN, C-SCAN, LOOK, C-LOOK 等磁盘寻道算法
设备管理：FCFS 算法；
内存管理：采用页式分配、best-fit 分配算法，页面替换的 LRU 算法和 FIF 算法；
以上算法的具体说明将在下面分模块介绍部分具体解释。

1.2.3 基本数据结构

①进程管理块 PCB：

```
class PCB
{
    int PID;    //进程标识符
    int processState;  //进程状态
    long IR;  //指令寄存器，保存当前正在执行指令的地址
    long pSeg;  //program segment，程序段地址
    long dSeg;  //data segment，数据段地址
};
```

②就绪/阻塞队列：

```
class QUEUE
{
    int priority;  //队列优先级
    int PID[MAX_Process];    //进程队列
};
```

③储存设备对象的基本信息：

```
struct device{
    string deviceName;//设备名称
    double tansmitRate;//传输速率（若有）
    int isBusy;//是否被占用
    int request[REQUESTNUM];//该设备的请求队列
};
```

④储存设备请求的信息：

```
class deviceRequest{
    int pid;//进程 ID
    string deviceName;//请求的设备名称
    string data;//数据内容（若有）
    double ioTime;//需使用该设备的时长（若有）
};
```

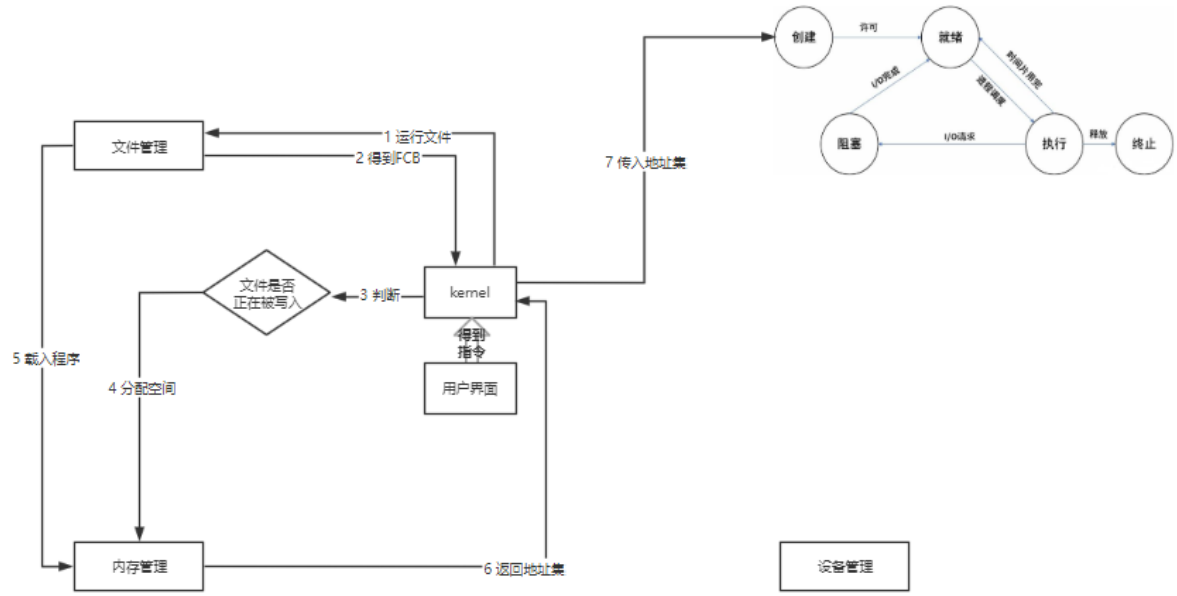
⑤进程存储位置：

```
class processStorage{
    int ID;
    int* dataSegment;
    int* codeSegment;
};
```

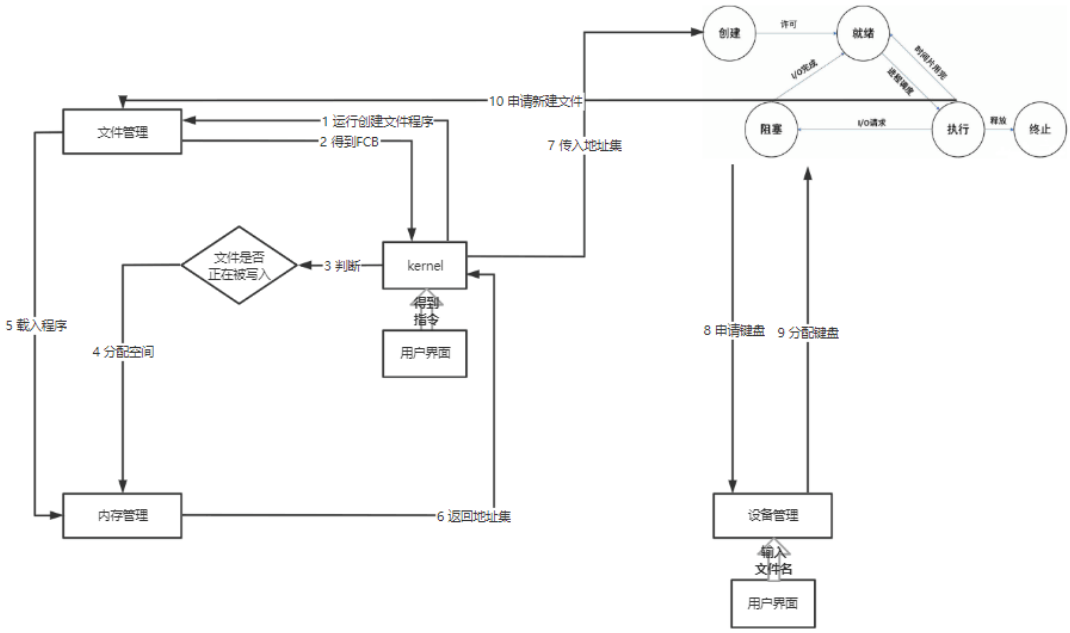
注：文件和内存部分中数据结构请参考各模块设计说明

1.3 基本处理流程

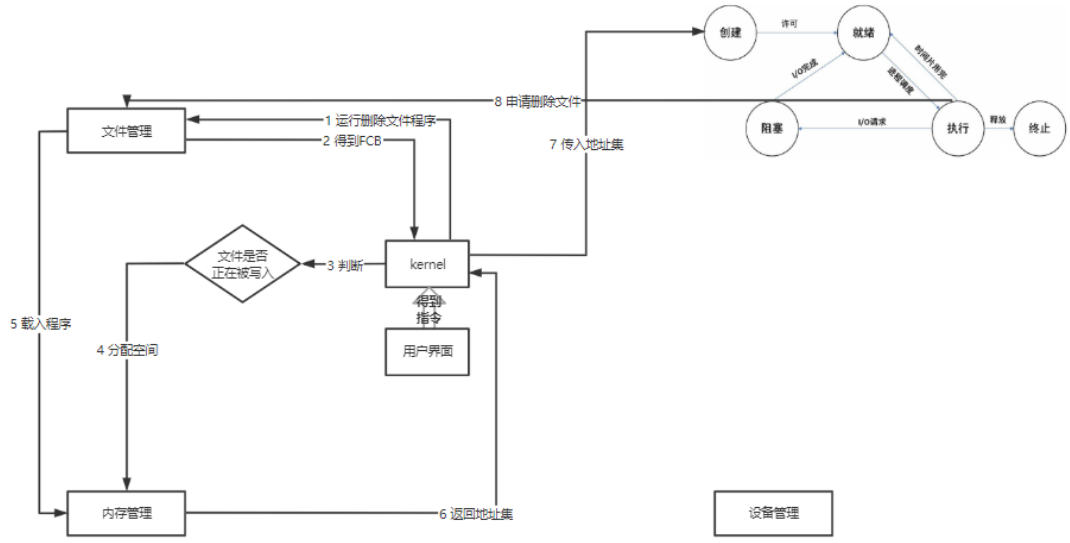
创造进程时：



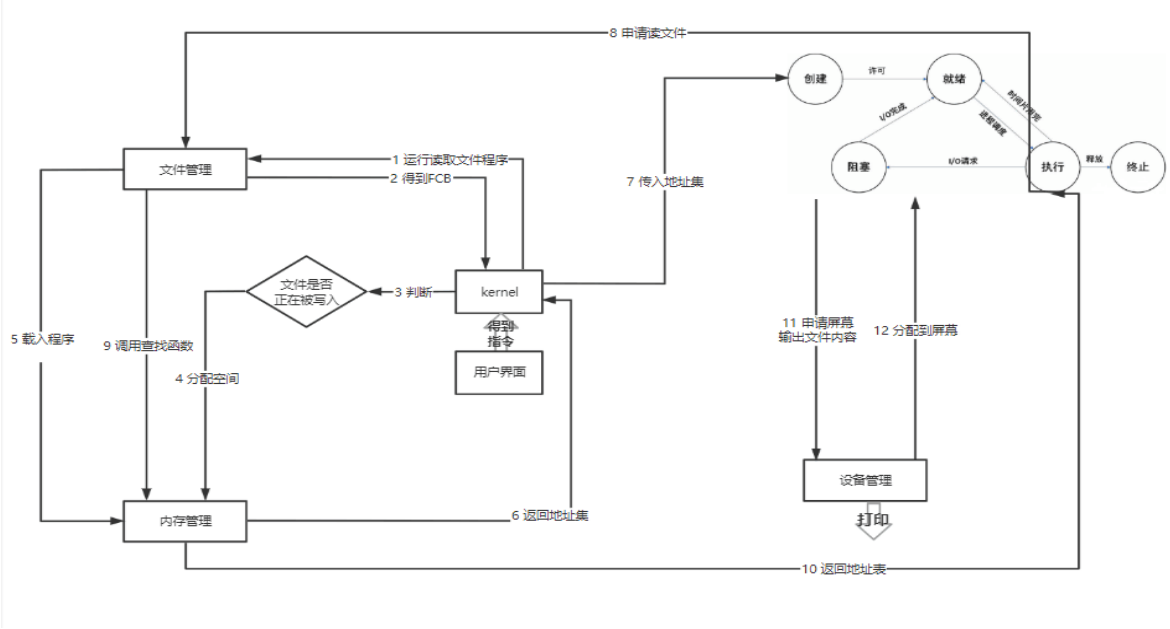
新建文件：


创建
许可
就绪
I/O完成
时间片用完
选择调度
阻塞
I/O请求
执行
释放
终止
10 申请新建文件
文件管理
1 运行创建文件程序
2 得到FCB
7 传入地址集
文件是否
正在被写入
3 判断
kernel
得到
指令
用户界面
5 载入程序
4 分配空间
8 申请键盘
9 分配键盘
内存管理
6 返回地址集
设备管理
输入
文件名
用户界面

删除文件：


创建
许可
就绪
I/O完成
时间片用完
选择调度
阻塞
I/O请求
执行
释放
终止
8 申请删除文件
文件管理
1 运行删除文件程序
2 得到FCB
7 传入地址集
文件是否
正在被写入
3 判断
kernel
得到
指令
用户界面
5 载入程序
4 分配空间
内存管理
6 返回地址集
设备管理

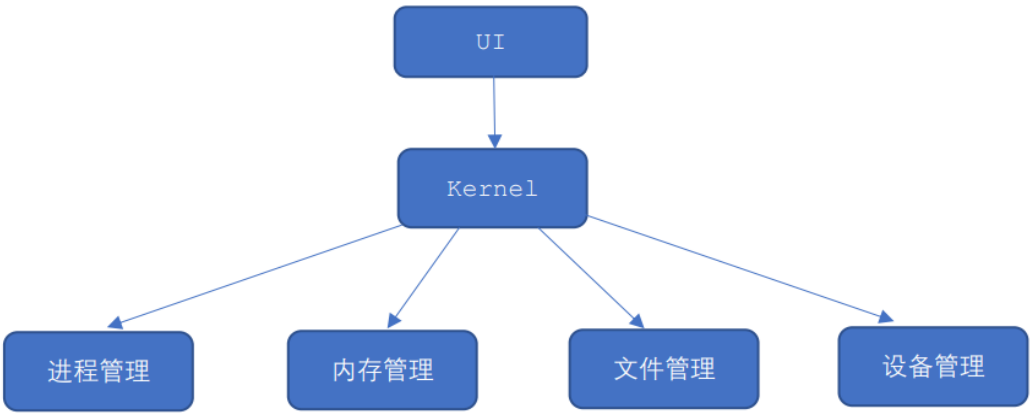
读取文件：



2　软件的体系结构和模块设计

2.1　软件的体系结构

2.1.1　软件体系结构框图



2.1.2 软件主要模块及其依赖关系说明

从上图可以看到，PD-OS 主要分为了六个模块，分别是最外层的图形化 UI 设计，链接 UI 和 OS 后端的 kernel，进程管理，内存管理，文件管理和设备管理部分。各个模块之间互相依赖，紧密联系。进程、内存、文件、设备四个模块依赖 kernel 与用户 UI 建立联系；进程管理依赖文件存储源程序，依赖内存管理创建进程，依赖设备管理完成进程状态的转换；内存管理依赖文件和进程分配内存，依赖设备管理中的磁盘管理把文件读入内存；文件管理依赖磁盘管理存储文件，依赖进程管理创建和修改删除文件，依赖内存管理把文件调入内存；设备管理依赖进程管理分配 IO 设备，依赖文件管理整理磁盘内容。

依赖关系	进程管理	内存管理	文件管理	设备管理
UI	提供用户 UI	提供用户 UI	提供用户 UI	提供用户 UI
Kernel	链接	链接	链接	链接
进程管理		分配内存	创建修改文件	分配 IO 设备
内存管理	创建进程		把文件调入内存	
文件管理	存储进程源文件	分配内存程序段		整理磁盘内容
设备管理	完成进程状态转换	把文件读入内存	存储文件	

2.2 软件数据结构设计

详见 1.2.3 基本数据结构。

2.3 软件接口设计

一、操作界面（命令接口）

采用图形化界面，用户使用这个界面组织工作流程和控制程序的运行。

二、系统功能服务界面（程序接口）

即 kernel 部分，用户程序在其运行过程中，使用系统调用功能来请求操作系统的服务。

2.3.1 外部接口

键盘：输入数据保存文件。
屏幕显示：显示程序图形界面关系。
Printer：可调用的显示设备之一。
磁盘：存储文件的设备。

2.3.2 内部接口

一、内存部分：

1.best-fit:

- 分配内存
  参数：PCB，size
  功能：根据提供的参数分配内存，如果分配失败返回-1，如果分配成功返回起始地址。
- 回收内存：
  参数：PCB，size
  功能：根据提供的参数回收相应内存，无返回参数

2. 分页存储

- 创建进程分配内存
  参数：PCB，size
  功能：根据参数分配内存，如果有空闲页直接分配，如果没有则进行页面替换，返回页起始地址，一页系统内存，一页数据内存（供写操作用）
- 查找虚拟内存中的页是否在物理内存中：
  参数：虚拟内存块号指针（可能含有多个）
  功能：根据虚拟内存块号查找物理内存中是否有对应页，如果有则给进程返回页地址，如果没有则重新分配页并调用读入函数，将读入后的页地址返回给进程
- 访问页表：
  参数：虚拟内存块号
  功能：根据虚拟内存块号找到物理页号并返回
- 删除特定页内容：
  参数：页地址
  功能：删除页地址原内容
- 释放内存：
  参数：PCB
  功能：释放 PCB 对应的内存

二、文件部分

- 创建文件
  参数：文件名称
  功能：在当前目录创建新的文件，成功返回 0，名称重复返回-1，名称非法返回-2
  创建文件夹的 C 语言指令：system("mkdir .\\XXX");（调用系统指令）
  设置一个特定的文件夹，为不可修改
  调用：创建文件时被进程调用

- 删除文件
  参数：文件名称
  功能：删除已有的文件，释放磁盘存储空间，成功返回 0，无权限返回-1，正在被写入返回-2

对于目录类型的文件，需要深度优先遍历目录下的所有文件，并进行删除
调用：删除文件时被进程调用；删除多级目录时被自身递归调用

- 读取文件
  参数：文件名称、块号（？）、虚存的真实地址
  功能：将目标文件载入到给定的内存地址中，成功返回 0，文件正在被占用返回-1，还有下一块返回-2
  文件状态修改：
  若写入标识符不为 0，则返回-1，不改变标识符的值；在文件开始时，打开文件表对应的读文件标识符+1，若返回为 0，则标识符-1；若返回为-2，则标识符保持+1 不变
  调用：读取文件时被进程调用；新建进程时被内存调用

- 写入文件
  （根据 QT 后续研究情况而定）
  情况 A：
  若：可以对输出进行随机读写
  参数：文件名称、内存地址+偏移量（表示从内存地址开始读到偏移量结束）
  功能：用新的内容替换原有文件，注意根据新文件的长度分配或释放磁盘块
  情况 B：
  若：无法对输出值进行随机读写
  参数：文件名称、内存地址+偏移量、写入模式
  功能：根据写入模式，用心的内容替换原有文件；或者在文件末尾写入，注意存储空间分配
  返回值：成功返回 0，正在被占用返回-1
  文件状态修改：
  函数开始时检查文件状态，若读或写标识符不为 0，则返回-1；若标识符为 0，修改写文件标识符为 1，直到写入完毕，修改标识符为 0
  注：因为各种原因，选用的文件锁的问题是，首先，写入必须一次完成， 进行分段写入的时候，文件可能被别的进程修改
  其次，要求每一次读文件，整个文件的所有块都必须全部载入内存（仍然是因为没有 close，没有办法强行终止，只能在读到文件结尾才认为这个文件不再被读），否则不能被写入，并且如果一次读操作只读了三块中的一块就结束的话，会在文件系统中造成一些隐患。
  调用：写入文件时被进程调用

- 查找文件（寻址）
  参数：文件名称
  功能：输入文件名称时，在当前目录查找，输入路径时，按绝对路径从根目录开始查找，返回文件控制块 FCB（包括文件大小、类型、权限、当前在内存的位置等信息）
  调用：被文件系统的上述接口调用；创建新进程时，被 kernel 调用，返回文件大小以判断内存空间是否足够；被内存调用确定是否已经在内存

- 修改目录
  UI 每一次改变目录，kernel 告知文件系统，修改当前位置指针

- 打印目录

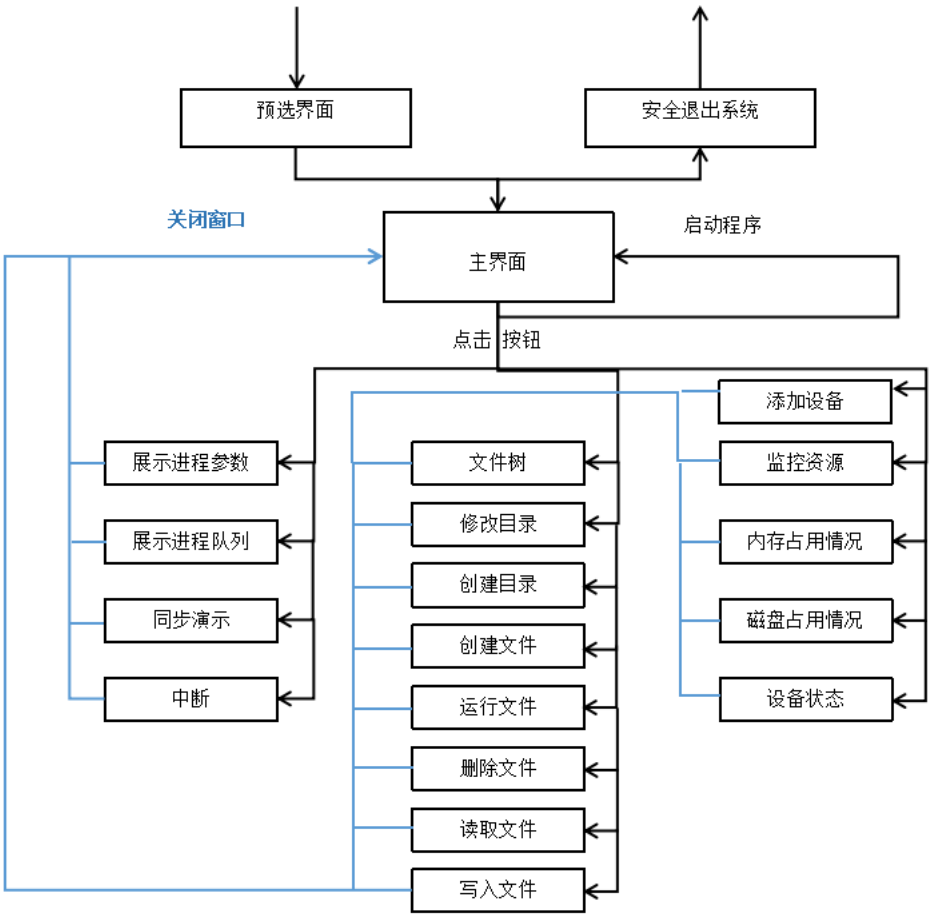
功能：打印当前路径到 UI 界面
调用：被 UI 调用

- 展示当前目录下所有条目
  功能：遍历并打印当前目录下所有文件名称

- 查找当前目录下符合某种条件的所有文件
  返回值：符合某种条件的文件列表
  调用：在 UI 界面选择操作时被调用

3 用户界面设计

3.1 界面的关系图



3.2 界面说明

3.2.1 界面 1：预选界面

算法具有多个选项可供用户选择，但选择过后在本次退出模拟操作系统之前不可更改

3.2.2 界面二：主界面

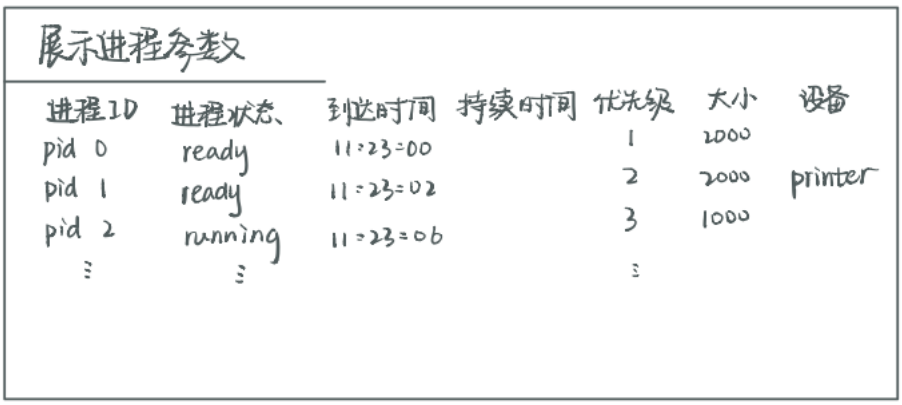
最开始进程还没创建，所以灰色按钮不可点：

双击.exe 文件后，会启动程序创建进程，所有按钮均可使用

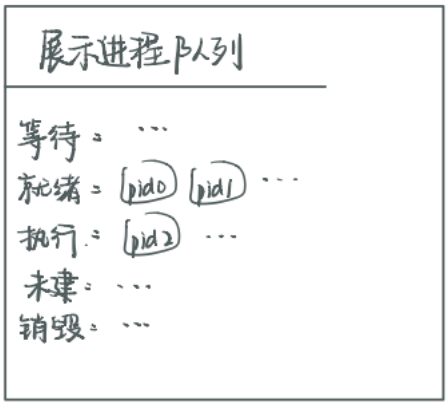
3.3 界面解释

3.3.1 进程部分

“展示进程参数”--进程参数包括分配的进程 ID，进程状态，进程的到达时间和持续时间，进程优先级，进程大小，进程想要请求的外设



“展示进程队列”--进程队列按照排队的进程顺序输出

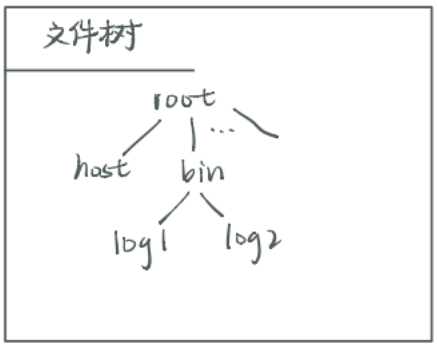


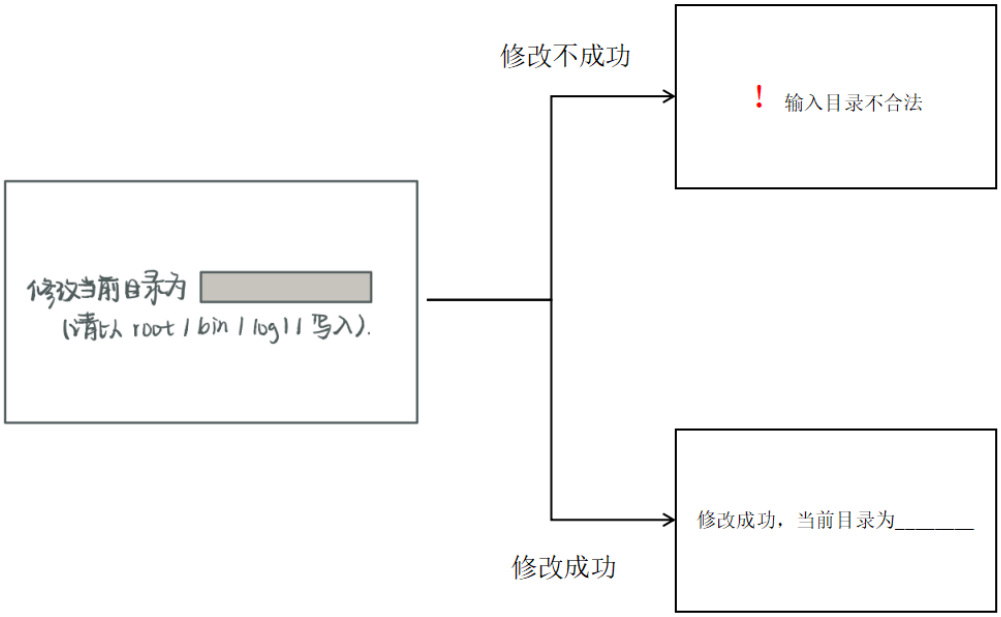
“同步演示”按钮点击后在新的弹窗里直接打印输出结果
“中断”按钮点击后代表用户手动中断

3.3.2 文件部分

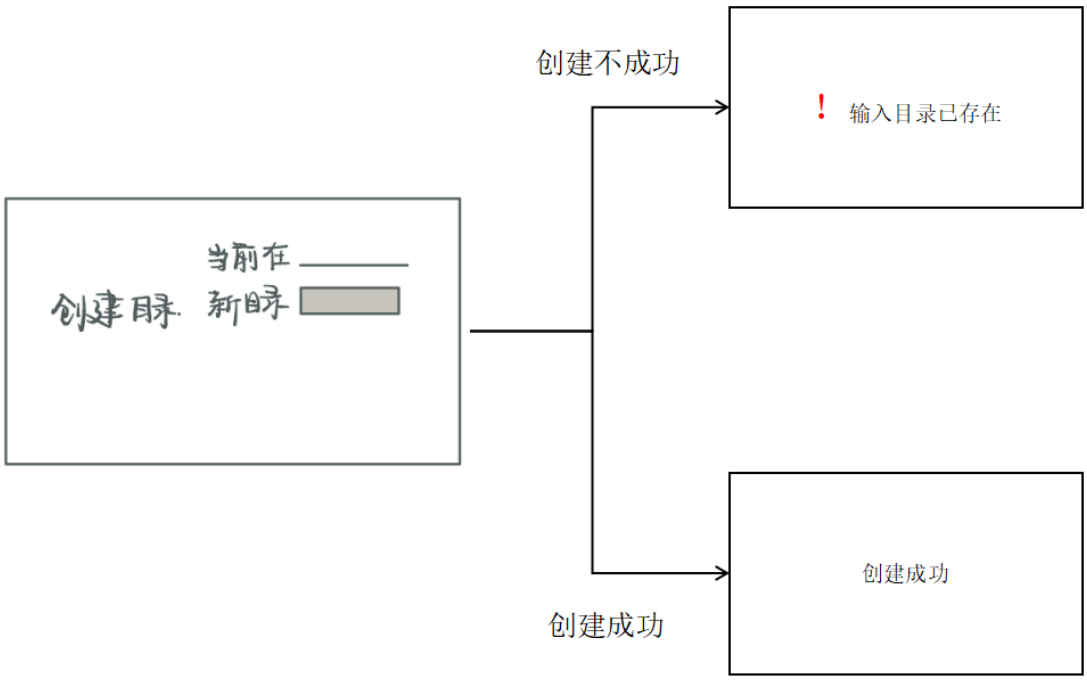
“所在文件目录”--显示当前所在目录路径

“文件树”--展示所有文件，与当前目录无关





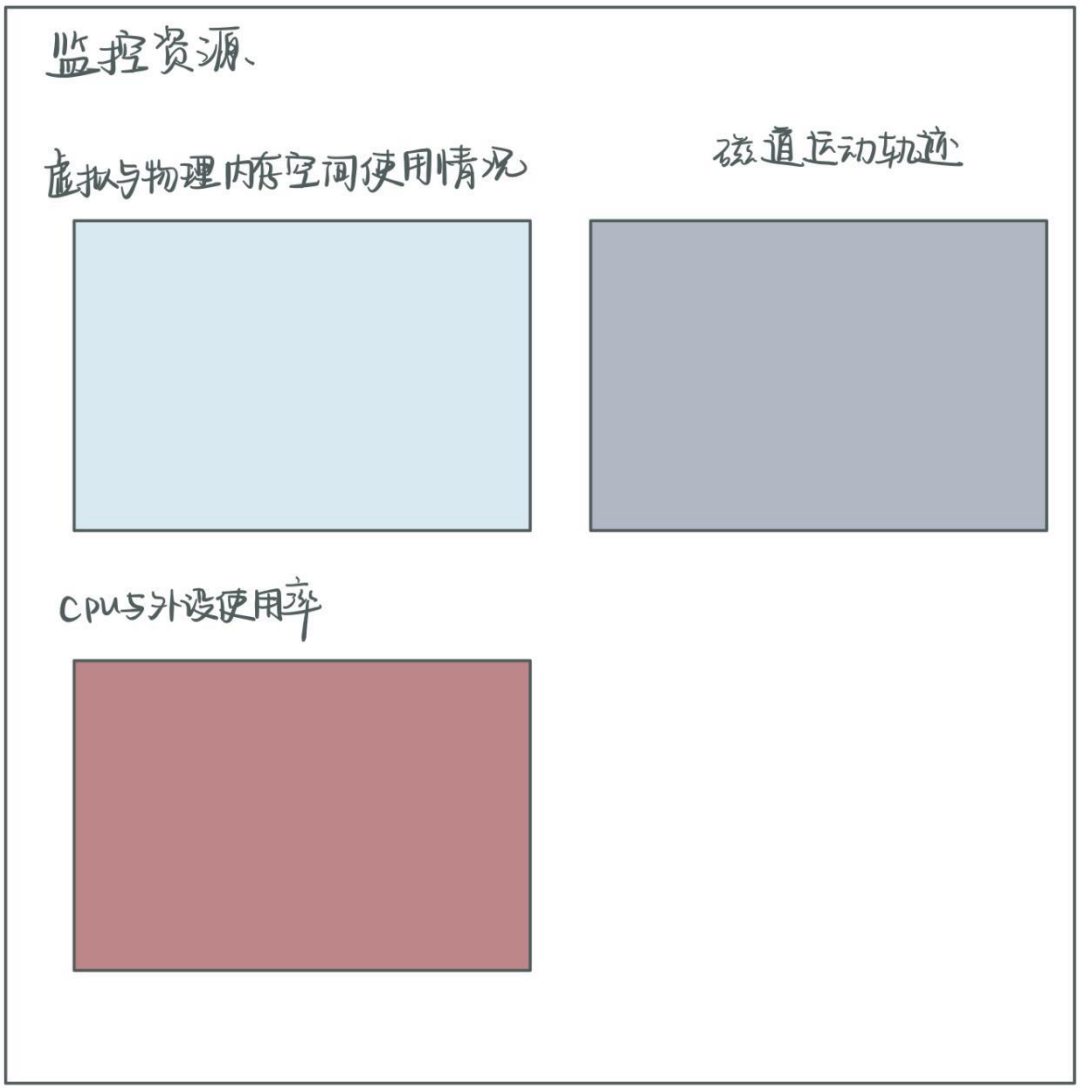
“**创建目录**”--只能在当前目录下创建新的



“运行文件”展示所有可执行 exe 程序，进而创建进程
其他对于文件的操作同样有非法等判断

3.3.3 监控部分

“监控资源”--以图形化形式输出监控数据



“内存占用情况”--输出各进程所占内存块大小及其位置，以及系统空闲内存块大小及其位置

“磁盘占用情况”--输出占用对象，各部分已用大小/各部分总大小

“设备状态”--输出打印机，键盘等外设的占用情况

“添加设备”--添加外设

4 相关处理流程

本章逐个地给出各个层次中的每个程序单元的设计考虑。

4.1 进程管理设计说明

4.1.1 进程管理算法及流程说明

一、 进程调度算法

对于进程的调度常分为长期调度程序和短期调度程序。前者从磁盘的作业池中选择进程调入内存以准备执行，后者从准备执行的进程中选择进程并为之分配CPU。查阅资料后，我们了解到 UNIX 和 Windows 的分时操作系统均没有长期调度程序，而只依赖短期调度程序维护系统进程，所以我们的 PD-OS 同样采取**只有短期调度**的方式。

a) FCFS 算法

先请求 CPU 的进程先分配到 CPU。

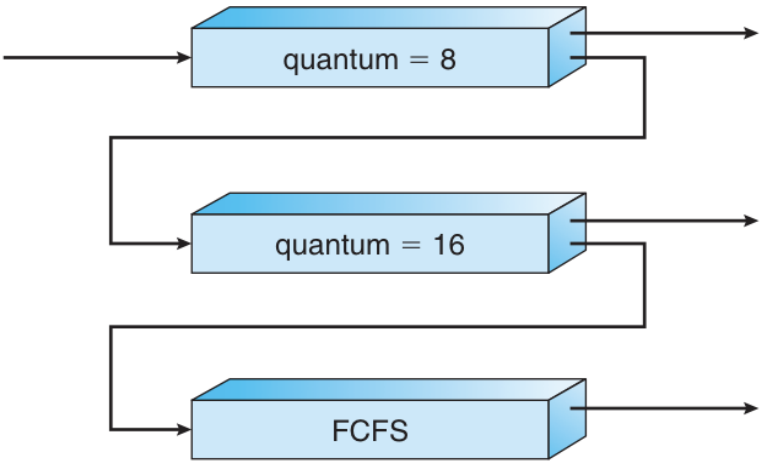
该算法利用 FIFO 队列来实现：当一个进程进入到就绪队列，其 PCB 链接到队列的尾部。当 CPU 空闲时，CPU 分配给位于队列头的进程，接着该运行进程从就绪队列中删除。

b) SJF 算法——支持抢占、非抢占两种模式

具有最短 CPU 区间的进程先分配到 CPU，若两进程具有相同的 CPU 区间长度，则按照 FCFS 原则来调度。

在非抢占模式下，只有当 CPU 空闲时才会使用调度算法从就绪队列中选择进程来运行；而在抢占模式下，CPU 空闲、有新的进程加入到就绪队列两种情况都会启用调度算法。

c) 多级反馈队列调度



我们将进程的优先级分为三等：0、1、2（0 为优先度最高），按照优先度分为 3 个队列。其中，队列 0 和队列 1 采用 RR 轮转算法，队列 0 每个进程拥有 8 个时间片，队列 1 则是 16 个（具体时间片数量和比例将根据今后 PD-OS 的实际运行情况进行调整）；而队列 2 采用 FCFS 算法。调度程序将首先执行队列 0 中的所有进程，只有 0 为空时，才能执行 1，类似的有队列 1 和队列 2。到达队列 1 的进程会抢占队列 2 的进程，类似有队列 0 和队列 1。队列 0 中的进程如果不能在规定的时间片中完成，就会被移到队列 1 的尾部；而如果队列 0 为空，队列 1 的头部进程将得到时间片，不能完成则会被抢占，并放到队列 2 中；当队列 0 和 1 均为空时，队列 2 按照 FCFS 执行进程。

二、 进程同步的实现

在我们设计的操作系统中，多个进程的临界区问题发生在对同一文件的读写操作时。为了解决这一问题，对每一个文件都设置一个状态量 is_writing，当 is_writing == 1 时表示有进程正在对该文件进行“写”的操作，此时不允许其他进程再次访问该文件。当“写”的操作结束后，将该状态量更改为 0，以此避免并发访问过程中可能的冲突。

三、 中断机制

每个进程管理块 PCB 都包含一个指令寄存器，记录该进程正在执行的指令。

在我们的操作系统中，发生中断的情况有以下几种：

- 时间片中断——时间片用尽但进程并未结束，指令寄存器的值修改为下一条指令，将该进程移入就绪队列重新等待调度，等到再次被调度时从指令寄存器显示的指令处开始执行。
- I/O 中断——修正当前进程指令寄存器的值，将该进程由运行态移入阻塞态，等待 I/O 请求完成后将其移入就绪队列等到再次被调度时从指令寄存器显示的指令处开始执行。
- 缺页中断——当前进程需要访问磁盘中的文件，修正当前进程指令寄存器的值，将该进程由运行态移入阻塞态，等待文件系统的返回信息，此后过程同上。
- 用户手动设置中断——该功能主要用于中断机制的展示，我们在 UI 界面上设置一个中断按钮，用户可在任意时间点击此按钮中断当前正在进行的进

程，系统开始执行事先设置好的一个中断处理程序，此后该进程被唤醒继续执行。我们将向用户动态展示这个过程。

对于每一种中断情形都设置对应的中断处理程序，并制成中断向量表。

4.1.2 进程管理数据存储说明

进程的代码以文件形式存储，扩展名为.exe，里面包含一个进程要执行的指令。

4.2 内存管理设计说明

4.2.1 内存管理数据结构说明

①long storage[]; //内存
②struct process_storage{
    int ID;
    int* data_segment;
    int* code_segment;
}ProcStorage；//进程存储位置
③int free_partition_table[10][4]; //连续分配空闲分区表，最多有十个分区，四列分别是分区号，分区大小，起始地址，状态
④map<int, int> page_table; //连续分配页表<页号，块号>
⑤int virtual_page_table[10][5]; //分页存储页表：内存块号，状态位，访问字段，修改位，外存地址
⑥int idx; //页表偏移量

4.2.2 内存管理算法及流程说明

一、内存管理算法

1.页式分配

采用页式分配时可以指定页面数量及页面大小，并记录每一页的分配情况（占用空间/占用进程 ID/分配 ID），页式分配算法在进行分配与回收时都是以页面为单位；

2.best-fit 分配算法

使用数组结构分别记录内存的分配情况（基址/分配大小/占用进程 ID/分配 ID）及空闲区的情况（基址/空闲区域大小），在每一次分配时，遍历空闲记录表，挑选符合条件的空闲区进行分配，在每一次回收时，需要考虑是否有与释放区相邻的空闲区，若有，则需要将该释放区合并到相邻的空闲区中，并修改该区的大小和首址，否则，将释放区加入空闲区的记录表中。

1）页面替换的 LRU 算法：

依据进程页表判断当前所访问的虚拟页是否已经被调入到物理内存中：

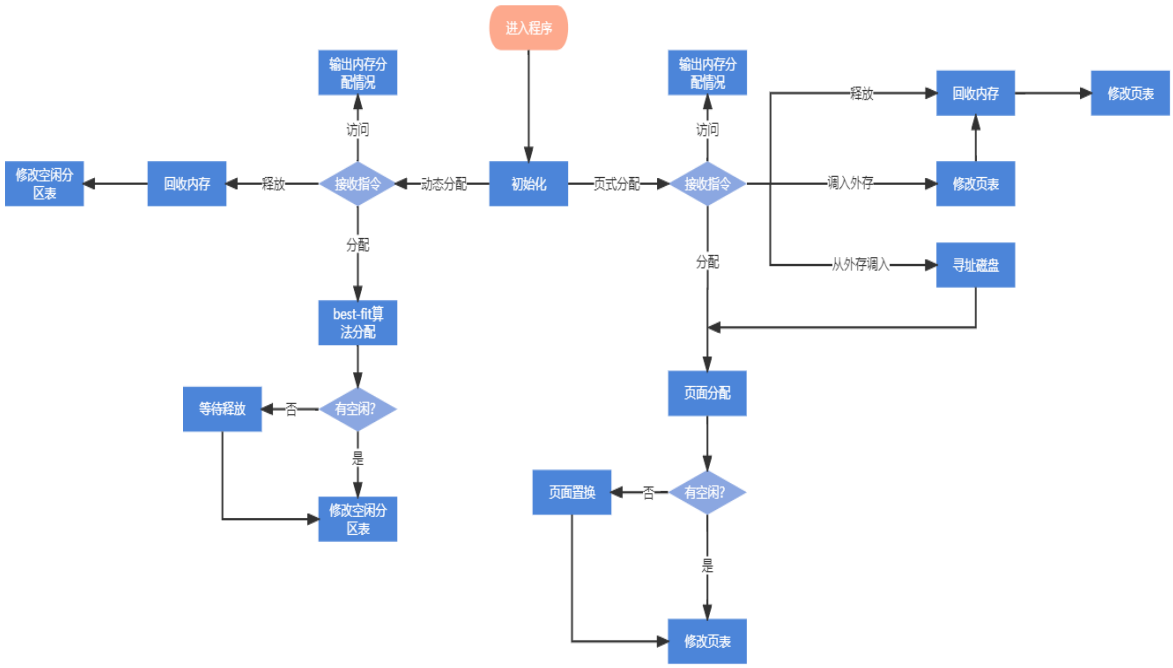
- 若已在物理内存中，则将该页放在调度队列尾部（表示最近访问）；
- 若未在物理内存中，则判断物理内存是否空闲，若空闲，则将访问页面调入物理内存中，并将该页放在调度队列尾部；若不空闲，则将调度队列头部的页面替换出去，将该页面所在页表的有效位置为-1（无效），并将该页面从调度队列中删除，后将访问页面调入物理内存，并将访问页面放在调度队列尾部。
-

2）页面替换的 FIFO 算法：

依据进程页表判断当前所访问的虚拟页是否已经被调入到物理内存中：

- 如已在物理内存中，则不做任何操作；
- 若未在物理内存中，则判断物理内存是否空闲，若空闲，则将访问页面调入物理内存中，并将该页放在调度队列尾部；若不空闲，则将调度队列头部的页面替换出去，将该页面所在页表的有效位置为-1（无效），并将该页面从调度队列中删除，后将访问页面调入物理内存，并将访问页面放在调度队列尾部。

二、内存管理工作流程图



4.3 设备管理设计说明

4.3.1 设备管理模块算法及流程说明

FCFS 算法：将申请同一设备的进程存入一个队列中，即请求队列，每次将设备分配给

该队列内储存的第一个进程，当该进程使用完设备或屏幕已经打印出相关信息后，则可以将设备分配给下一个请求队列中的进程。

流程说明：当有新设备时可以添加新设备；

对各个设备按 FCFS 调度算法依次处理对设备的请求进程，修改分配给进程的设备的占用状态，待使用结束后回收、修改状态、再分配；

当用户用键盘输入后，UI 将输入框中内容传给设备管理中的缓存，设备管理再将 pid 与缓存中的内容传给内存管理；

当向用户展示设备状态时，打印出所有设备的名称以及是否正在被使用，若是，则还要输出正在使用该设备的进程号。

4.4 文件管理设计说明

4.4.1 文件管理数据结构说明

①模块内的全局变量
static int MAX_INDEX;　　//iNode 最多有多少直接索引
文件属性（文件控制块）
②class FCB
{
string name;　　//文件名，在每个目录下唯一
char type;　　//目录(.f)、程序文件(.e)、用户文件(.t)
pointer;　　//从根目录开始的绝对路径，指向所在的目录
int size;　　//文件长度，每次写操作后都需要更新
int auth;　　//权限，无限制（auth=0），不可删除（auth=1），只读（auth=2）
vector<int> index;　　//索引，第 i 个元素表示文件第 i 块地址，限制最多 MAX_INDEX 块
int singleIndirect;　　//一级间接索引，对大文件可用
成员函数见 4.4.2
}
③class OftEntry
{
FCB file;　　//文件控制块副本，包含文件信息
int readFlag;　　//读文件标识符，计数信号量，标识正在读的进程数量
int writeFlag;　　//写文件标识符，二元信号量，一次只允许一个进程写入
成员函数包括更改两个信号量的函数，以及返回 FCB 中的一些信息
}
④class OpenFileTable
{
vector<OftEntry> OFT;　//打开文件表的主体，包括所有打开文件的 FCB 副本、读写情况

成员函数：新增/删除条目
}
⑤目录
逻辑上：树形结构，根节点为根目录，每个节点的孩子标识目录下的文件，叶子结点标识可执行文件或文本文件
存储上：哈希表
class directoryTree
{
unordered_map<string, void *> directoryTree
每个目录下的内容用哈希表存储，key 是文件名，内容是一个无类型指针：
若文件为目录，则指针类型为 unordered_map*，指向另一个哈希表；
若文件为.t 或.e，则指针类型为 FCB*，指向文件控制块
成员函数见 2.3.2 接口部分
}
⑥空闲空间表
vector<int> freeSpaceList
用位向量 bit vector 实现，位数等于磁盘块数，每位标识一个磁盘块
每位只能为 0 或 1，0 表示块不可用，1 表示块为空
⑦磁盘
暂定为 Vector 的三层数组
Disk[i][j][k]：标识第 i 个柱面第 j 道第 k 分区。
暂定每个磁道扇区数量相同，且一个文件块对应一个扇区
不确定：是否实现扇区的数据结构？（头+数据块+尾）

4.4.2 文件管理算法及流程说明

一、存储逻辑结构

- 选用索引分配，为简化操作，将索引块并入 FCB。
- 每个索引块的最后一条为一级间接块。考虑到本 OS 规模，应该不需要更多层次的间接块。
- 与真实操作系统不同的是，真实 OS 需要内存先读入索引再根据索引读文件，但因为我们限制了文件操作的复杂度，所以文件系统会遍历索引段将文件块全部载入内存。
-

二、存储物理结构

磁盘寻道算法（选一些合适的）：FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK

三、数据存储说明

一方面，磁盘具有非易失的特点，要求每一次重启操作系统，磁盘上的数据不应丢失或损坏；另一方面，频繁的文件操作效率低、性能差。因此我们决定在每一次关闭操作系统时，将文件树和所有文件以文件的形式写入程序所在文件夹；重新启动时，作为初始化，将文件树和存储的文件载入内存，后续的磁盘读写操作在程序内存中实

现。

需要存储的文件：目录结构、对应每个文件的 txt 文档。

5 总结

在概要设计的这两周里，我们共组织三次项目会议，对 PD-OS 各个模块之间的配合做出了详尽的讨论和设计。我们考虑到计算机以进程的形式完成自己各个功能，于是决定以进程的方式完成文件相关操作，在一次次的讨论中深刻理解了在一个简单的文件操作过程中涉及了进程管理、内存管理、文件管理、设备管理的相互配合和相互调用。以小见大，我们在讨论的过程中对操作系统的运行过程都有了更加深刻的了解。

目前，我们的设计还有许多缺陷，比如进程操作的指令集还没有确定，进程在内存中创建时申请数据段和堆栈的细节还没有确认清楚等等。同样，我们还出现了成员只对自己负责的模块有较深的了解，而对于别的模块实现认识不清，以至于接口方面出现了种种矛盾的问题。在接下来的时间里，我们会统一再对操作系统课程做进一步的回顾与复习，争取做到每个成员都对系统整体有较深的了解。除此之外，我们还打算对 Linux 操作系统做进一步的深入研究，从实例中汲取经验教训和值得学习的亮点。