文件系统详细设计报告

一、主要功能简介

文件系统参照 FAT16 的设计，将一个 64M 的 2 进制文件作为磁盘。实现在这个模拟磁盘上进行创建、删除目录；创建、删除文件；读、写文件；提供虚拟内存的操作，并且给用户提供了 ls、cd 等命令。

文件系统的内部操作包括：
磁盘级操作：生成磁盘文件、初始化磁盘
文件系统级操作：启动文件系统、退出文件系统、格式化文件系统
文件分配表 FAT 及物理块相关操作：初始化文件分配表、更新文件分配表、申请物理块、释放物理块、获取文件所有的块号
系统打开文件表的操作：定位，创建，删除，更新打开文件表项
进程打开文件表的操作：定位，创建，删除，更新进程打开文件表以及表项
对目录的操作：创建，删除目录、显示目录、改变当前工作目录
对目录项的操作：创建，删除，更新目录项
对普通文件的操作：创建，删除，打开，关闭，读，写文件
交互操作：即虚拟内存的页面置换

二、主要数据结构设计

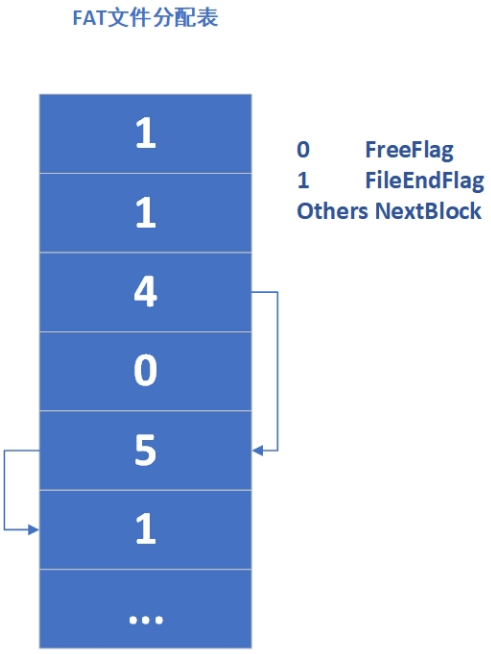
1.磁盘

引导区 1 block	保留块 1 block	FAT1 128 blocks	FAT2 128 blocks	根目录区 2 blocks	数据区

磁盘结构：类似 FAT16 的磁盘结构，将一个 64M 的 2 进制文件作为磁盘
文件系统的物理块大小为 1KB,共计 2^16 块，即磁盘空间是 64MB
磁盘分为引导区(1 块)，保留块（1 块），FAT1（占 128 块）,FAT2(备份，占 128 块)，根目录区（2 块）,数据区（大小可变）
只有文件系统有权直接操作磁盘。

```
//磁盘（文件形式）
fstream disk;
```

2.FAT 文件分配表



文件分配表中每一项为 16 位，记录了对应的物理块相连接的下一块的位置

若物理块空闲，该项为 0，表示可用，若物理块为文件的最后一块，该项为 1，表示结束块

FAT1 文件分配表大小为 128k，占 128 个物理块

FAT2 是 FAT1 的备份，与 FAT1 完全相同

FAT 会在启动文件系统时读入，在退出文件系统时写回磁盘。

```
//文件分配表中每一项为 16 位
typedef unsigned short fid;
fid fat[FATENTRYNUM];
```

3.FCB 文件控制块

文件控制块，用以描述文件的属性，并作为目录项，每个 FCB 占 32 字节。

每个文件控制块包含：文件名、文件权限、文件属性、起始块号、文件长度、文件大小、文件创建时间、文件最后修改时间这些属性。

```
/*文件控制块，每个目录项占 32 字节(14+1+1+2+2+4+4+4)*/
typedef struct FCB
{
    char fileName[14];          //文件名&扩展名
```

```
    char rights;                //文件权限：0 不可读写，1 只读，2 可写
    unsigned char attribute;    //属性，暂定 0 表示目录，1 表示普通文件
    unsigned short startBlock; //文件开始的磁盘块号
    unsigned short length;      //文件的长度，指占多少个磁盘块
    unsigned int size;          //文件大小，单位字节
    unsigned int createTime;    //文件创建时间
    unsigned int updateTime;    //文件最后一次修改时间
} fcb;
```

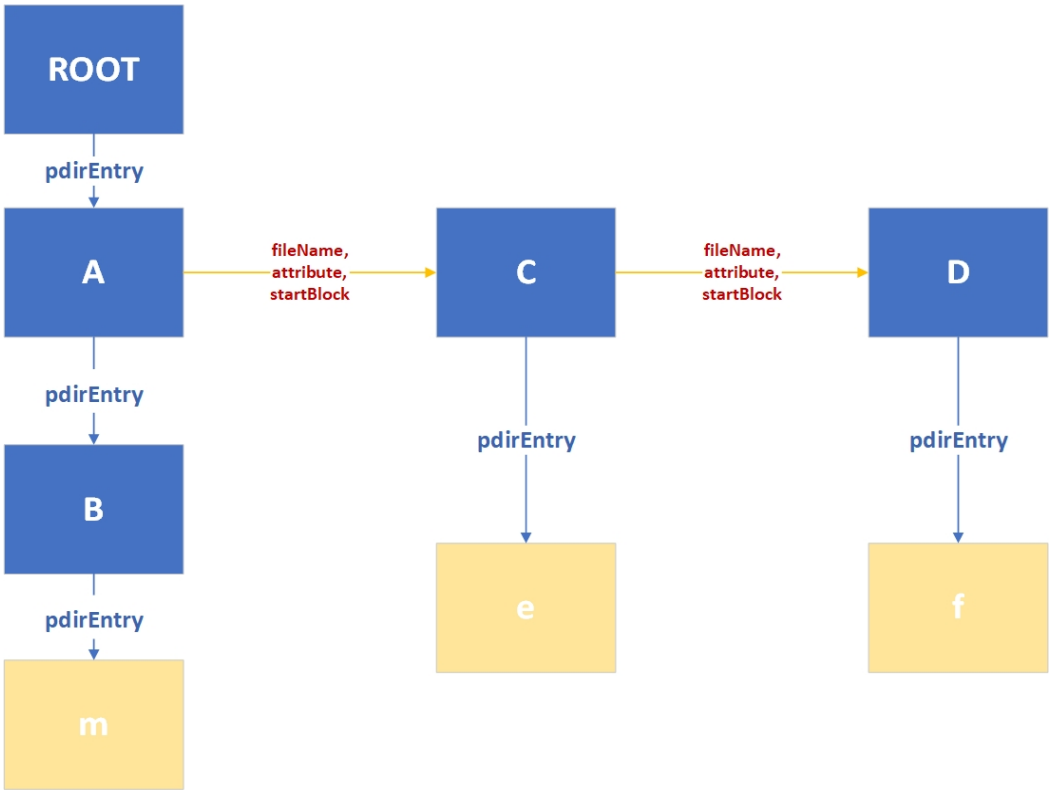
4.目录

目录逻辑上是二叉树型结构，使用 vector 下标访问同级文件（目录）的目录项，通过同级目录项起始块号（FCB 中的 startBlock）来访问下一级目录内容。

根目录有固定的磁盘块位置和长度，其他目录大小固定，均占 1 个磁盘块，但位置不固定，根据 FAT 表随机分配空闲块。

目录被视为一种特别的文件，除了根目录有固定位置，其他目录均在数据分区。

文件系统会维护当前工作目录，记录当前工作目录的目录项。



访问举例:

如图，若当前工作目录为根目录，需要对 D 目录下的 f 文件进行读写。

首先根据根目录找到目录项 A（通过文件名，获得下标)。

根据 A 的 FCB ，找到存储 A 目录开始的物理块（fileName 和 atribute 指示该目录项指向一个目录，startBlock 指示起始块号)。

然后生成 A 的目录结构（即创建 A 的目录项 vector)。

同上，接着找到 C 目录和 D 目录，最终搜索 D 目录，找到文件 f 的目录项。

根据 f 的 FCB，获取文件长度和起始物理块号，参照 FAT，即可获得完整的文件 f。

```
//目录逻辑上的二叉树型结构，使用向量下标连接同级文件（目录），通过同级目录起始块号（FCB 中的 startBlock）连接下一级目录内容
typedef vector<fcb> directory;
//当前工作目录名
string tempDirName;
//当前工作目录
directory tempDir;
```

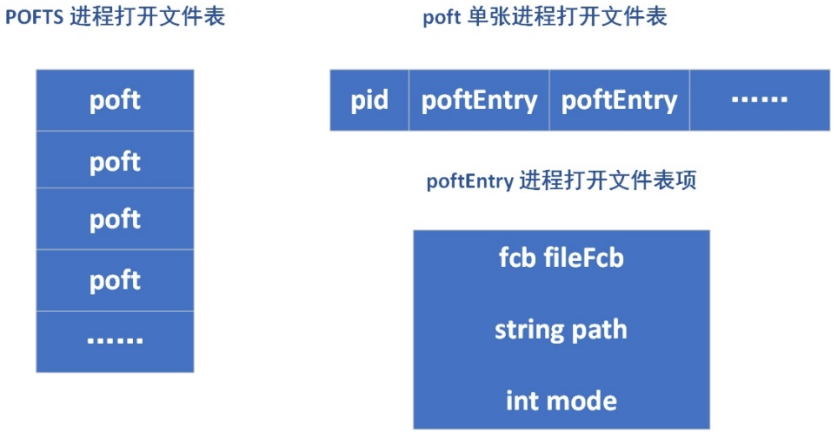
5.系统打开文件表

系统打开文件表项以文件为主体，在进程打开和关闭文件时，对表项进行操作。

系统打开文件表项记录了文件的 FCB、文件路径、被多少个进程打开、各个进程的 pid、打开模式（读或者写）。

```
//系统打开文件表项
typedef struct softEntry
{
    fcb fileFcb;      //文件 fcb
    string path;      //路径名,以防有重名文件
    int pn;           //表明当前使用该文件的进程数
    vector<int> pid; //进程标识符数组，用于表明哪些进程在使用该文件
    int mode;         //文件打开模式，标明各个进程打开文件的方式
} softEntry;
vector<softEntry> soft; //系统级打开文件表
```

6.进程打开文件表



进程打开文件表(pofts)以进程为主体，在进程打开和关闭文件时，对表和表项进行操作。每个在使用文件的进程都会维护一张进程打开文件表（poft）,他们集体组成了进程打开文件表（pofts）

进程打开文件表项(poftEntry)记录了文件的 FCB、文件路径、打开模式（读或者写）。

单张进程打开文件表(poft)包含一个 pid 和若干个表项。

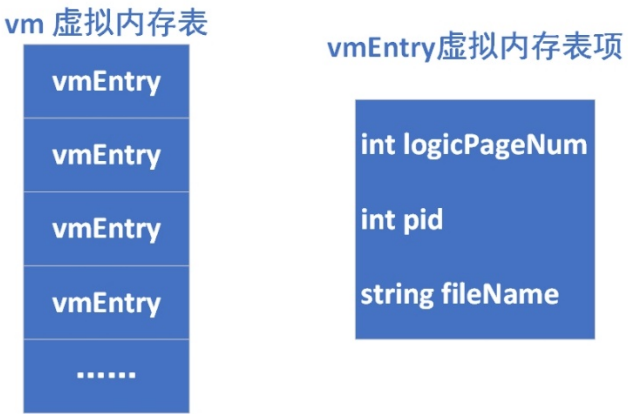
```
//进程打开文件表项
typedef struct poftEntry
{
```

```
    fcb fileFcb; //目录项
    string path; //路径名,以防有重名文件
    int mode; //文件打开模式
} poftEntry;

//单个进程文件打开表
typedef struct poft
{
    int pid;                    //表明进程 id
    vector<poftEntry> entries; //进程打开文件表项数组
} poft;

vector<poft> pofts; //进程打开文件表
```

7. 虚拟内存



虚拟内存是一片与内存交互的区域，大小为 32KB，设计上将其放在了 FAT2 的磁盘块上，在文件系统启动时，将 FAT2 的磁盘块前 32 块清空作为虚拟内存，在退出文件系统时，才将 FAT 的内容备份到 FAT2 的磁盘块。这样的设计，实现了 FAT2 磁盘块的复用，提高了磁盘的利用率，并节省了磁盘空间。

文件系统会维护一张虚拟内存表 vm，用来进行页面置换。

虚拟内存表项记录了逻辑页号，进程 pid 和文件名，当逻辑页号为 0，表明对应的虚拟内存块是空闲的。

```
//虚拟内存表项
typedef struct vmEntry
{
    int logicPageNum; // 0 表示该块空闲
    int pid;
    string fileName;
} vmEntry;
vmEntry vm[VMSIZE]; //下标对应磁盘位置
```

三、主体函数设计

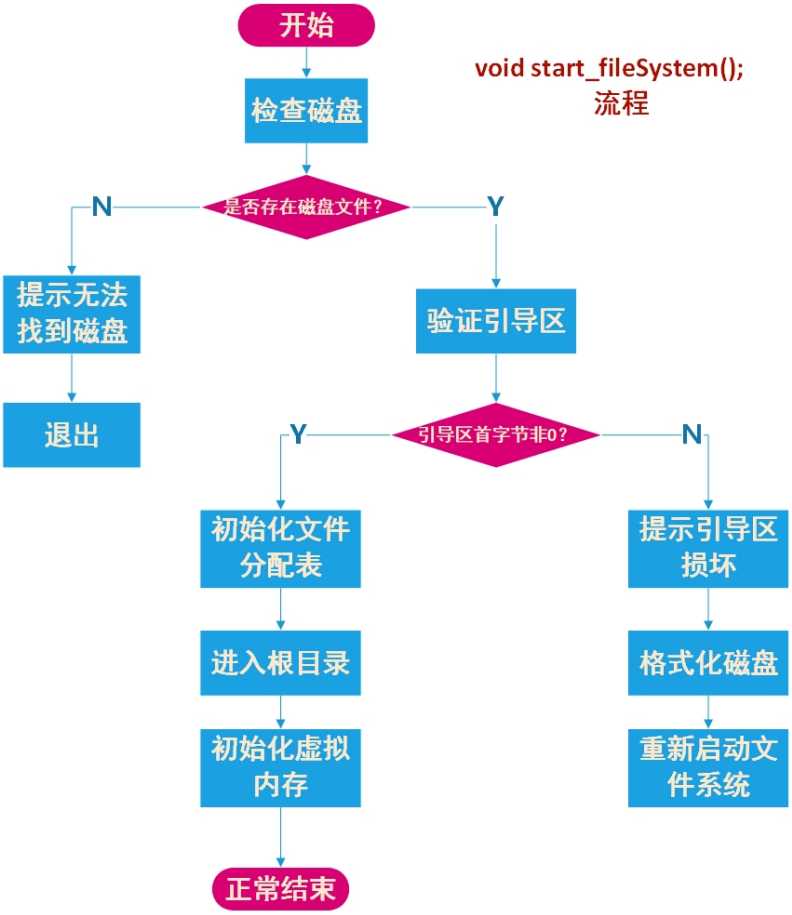
1、启动文件系统

void start_fileSystem();

第一步：检查磁盘，检测不到则提示无法找到磁盘，并退出

第二步：验证引导区，即验证首字节是否非 0，验证失败则提示引导区损坏，并格式化磁盘，重新启动文件系统

第三步：进入根目录，初始化文件分配表和虚拟内存



2、void exit_fileSystem();　//退出文件系统

第一步：关闭所有打开的文件

第二步：保存文件分配表（写回磁盘 FAT1）

第三步：备份文件分配表（写回磁盘 FAT2）

void exit_fileSystem()
流程

3、读文件

int read_file(string fileName, vector<unsigned char*>& mem, int size);

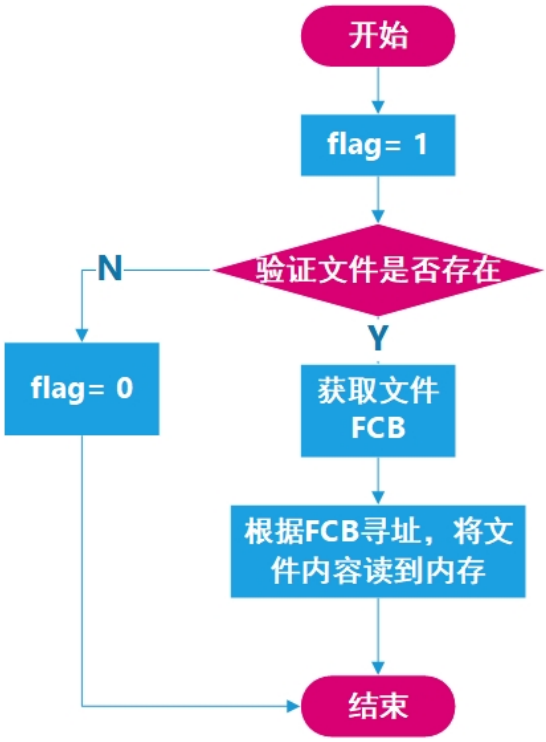
返回值：0 表示失败，1 表示成功

参数：fileName 文件绝对路径及文件名、mem 内存地址容器、size 读取大小（字节为单位）

第一步：验证文件是否存在，不存在 flag 置 0，直接结束

第二步：根据文件的绝对路径和文件名定位文件的 FCB

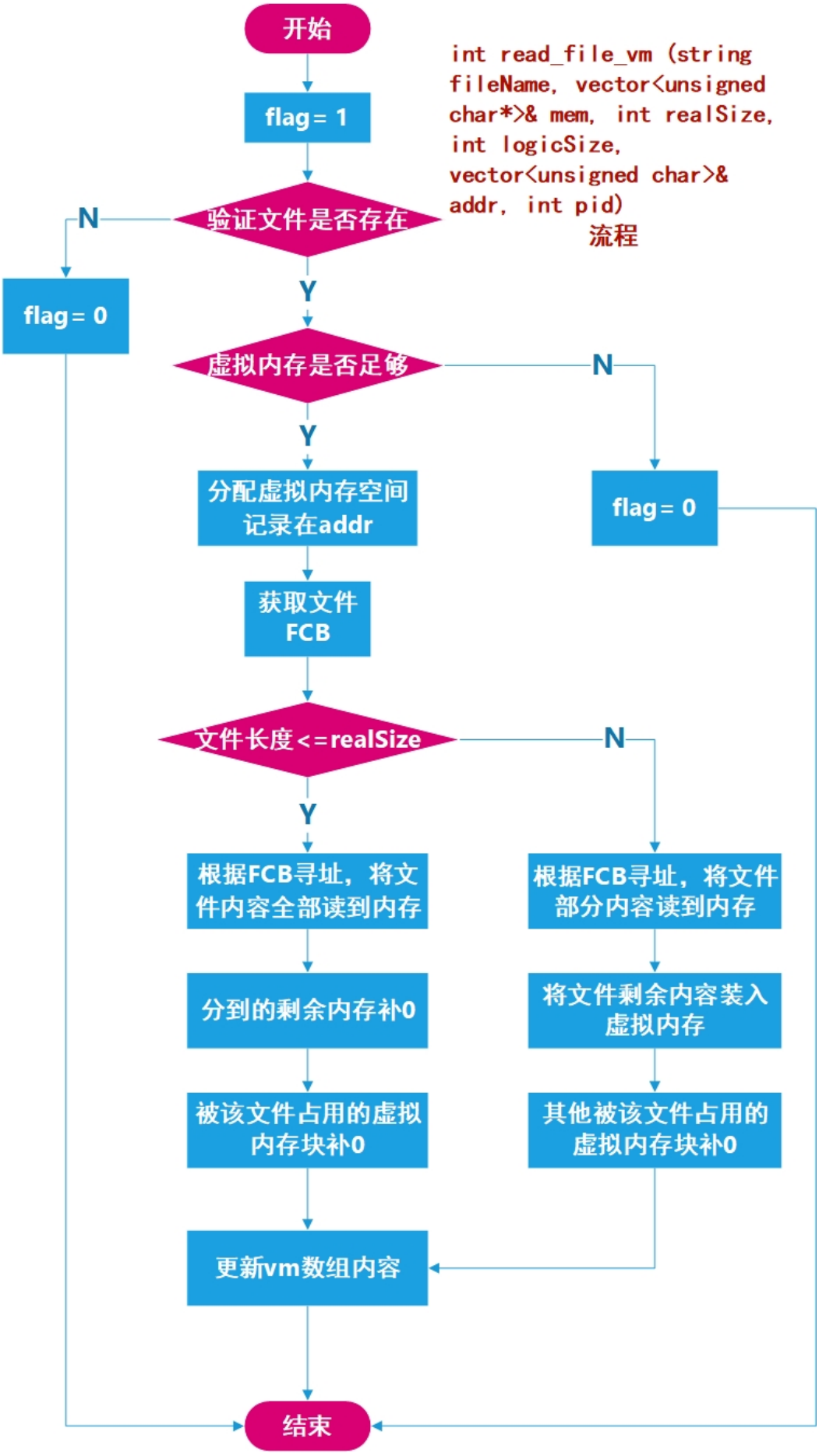
第三步：根据 FCB 的 startBlock 和 length 定位文件在磁盘上的位置，寻址并读出文件内容到 mem,返回 flag



int read_file(string fileName, vector<unsigned char*>& mem, int size);
流程

int read_file_vm(string fileName, vector<unsigned char*>& mem, int realSize, int logicSize, vector<unsigned char>& addr, int pid)　//调用虚拟内存的读文件
返回值：0 表示失败，1 表示成功
参数：fileName 文件绝对路径及文件名、mem 内存地址容器、realSize 目标内存大小（块为单位）,logicSize 逻辑大小（比 realSize 大，多出来的部分暂存在虚拟内存），addr 该文件占用的虚拟内存下标，pid 进程符号

第一步：验证文件是否存在，不存在 flag 置 0，直接结束
第二步：判断虚拟内存是否有足够的空间，不够则 flag 置 0，直接结束
第三步：分配虚拟内存空间，将对应的虚拟内存标记为占用，下标记录在 addr
第四步：判断文件真实大小和目标内存大小的关系，将文件读到内存，文件小于目标内存则剩余空间补 0，大于则将文件剩余部分装入虚拟内存
第五步：虚拟内存部分按情况补零，并更新 vm 数组，返回 flag


开始
flag = 1
验证文件是否存在
N
flag = 0
Y
虚拟内存是否足够
N
flag = 0
Y
分配虚拟内存空间
记录在addr
获取文件
FCB
文件长度<=realSize
N
Y
根据FCB寻址，将文件内容全部读到内存
分到的剩余内存补0
被该文件占用的虚拟内存块补0
根据FCB寻址，将文件部分内容读到内存
将文件剩余内容装入虚拟内存
其他被该文件占用的虚拟内存块补0
更新vm数组内容
结束
int read_file_vm (string fileName, vector<unsigned char*>& mem, int realSize, int logicSize, vector<unsigned char>& addr, int pid)
流程

4、写文件

int write_file(string fileName, vector<unsigned char*>& mem, int size);

返回值：0 表示失败，1 表示成功

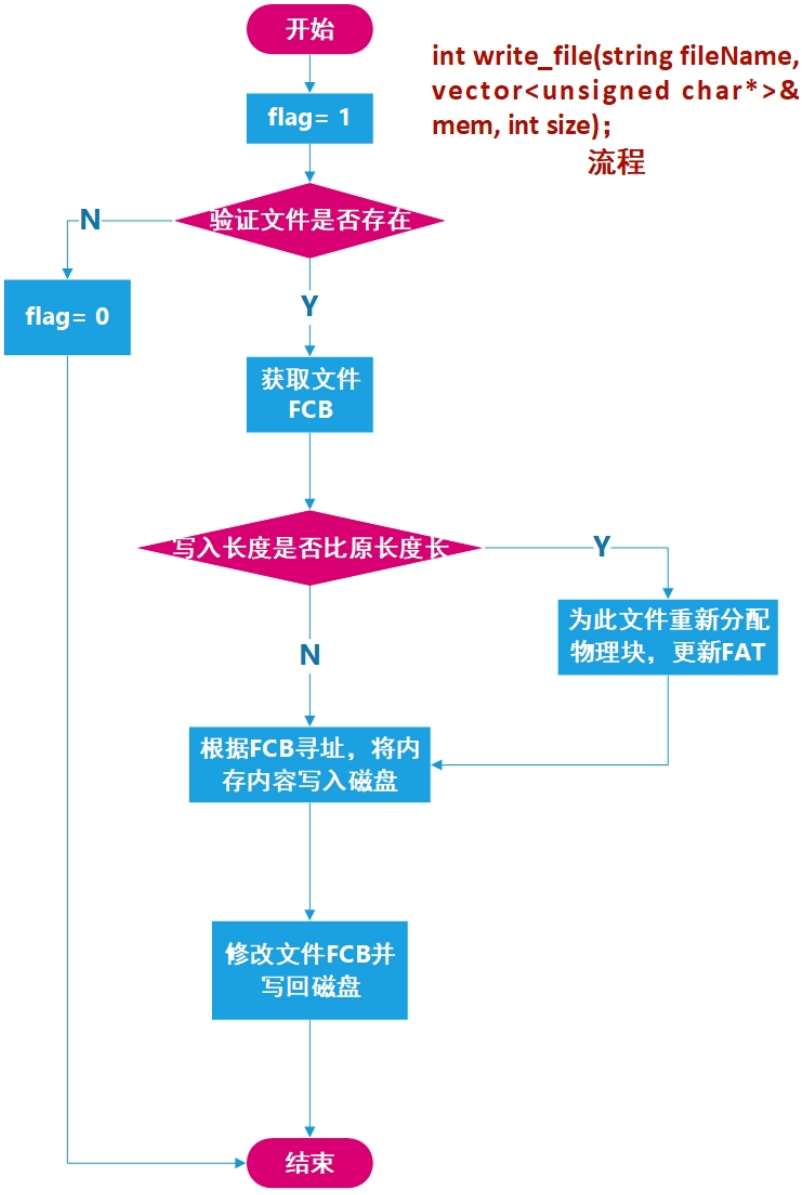
参数：fileName 文件绝对路径及文件名、mem 内存地址容器、size 读取大小（字节为单位）

第一步：验证文件是否存在，不存在 flag 置 0，直接结束

第二步：根据文件的绝对路径和文件名定位文件的 FCB，判断写入的大小是否超过了原先的大小（长度），需要重新分配物理块，更新文件分配表

第三步：按块将内容写入磁盘

第四步：修改该文件 FCB 并写回磁盘，返回 flag



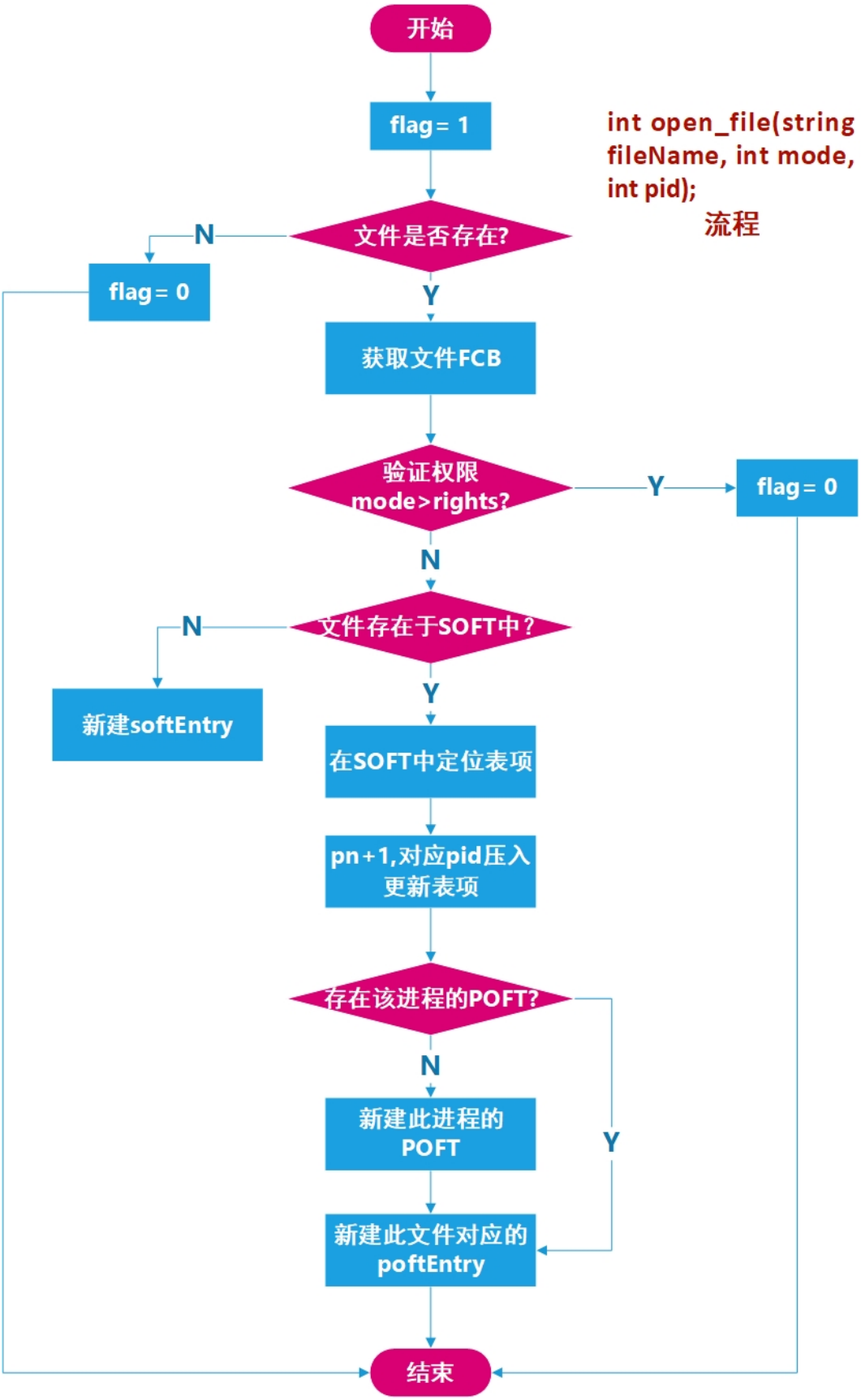
5、打开文件

int open_file(string fileName, int mode, int pid);

返回值：0 表示失败，1 表示成功

参数：fileName 文件绝对路径及文件名，mode 文件打开方式，pid 进程标识符

第一步：验证文件是否存在，不存在 flag 置 0，直接结束
第二步：根据 FCB 验证打开模式，超出权限则 flag 置 0，直接结束
第三步：系统文件打开表中找有无该文件的表项，如果没有，新建这一项。如果有，只能以只读模式再次打开，并修改文件打开表的相关项
第四步：新建进程级文件打开表项，如果程是第一次打开文件，那么新建一张该进程的表

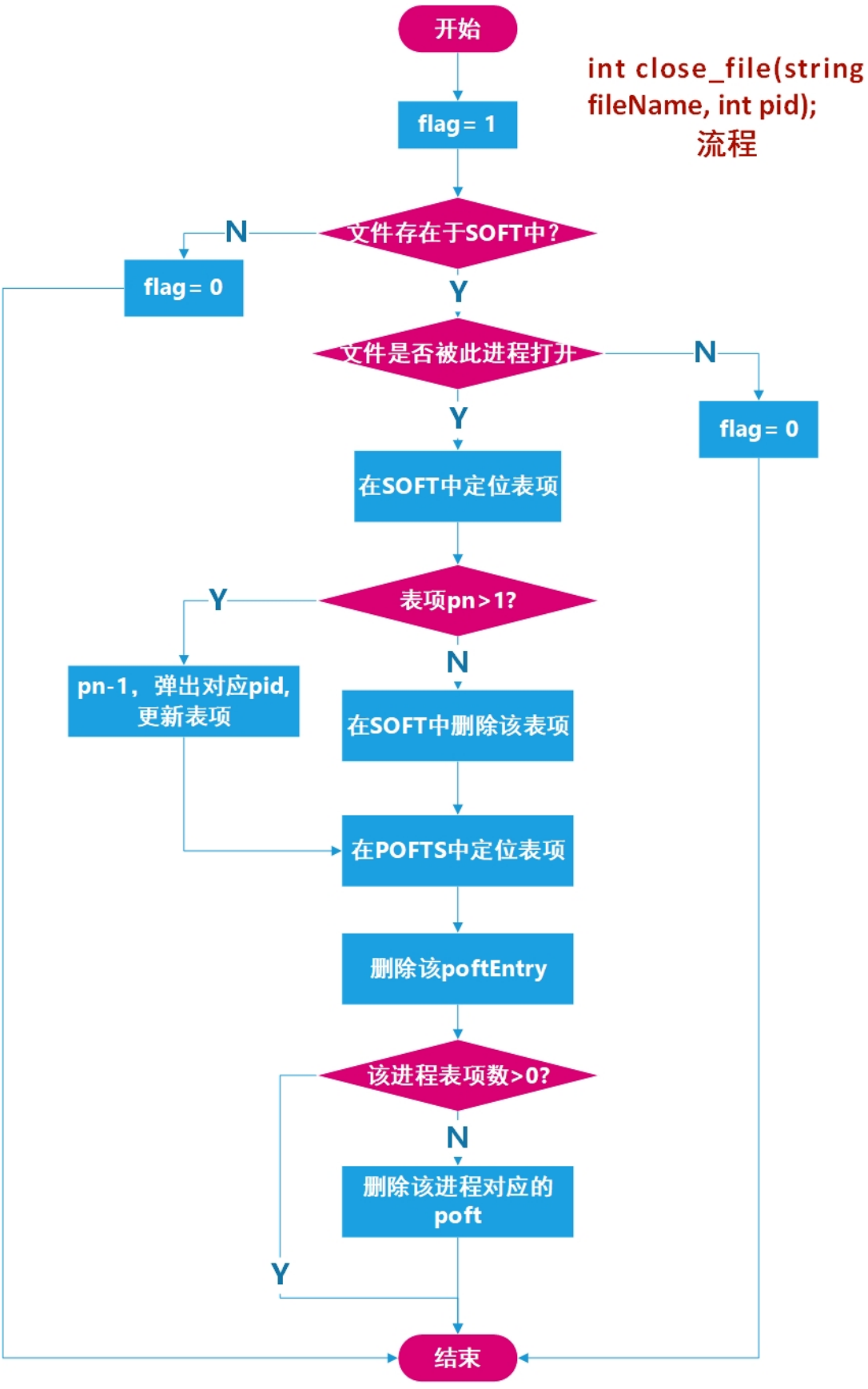


6、关闭文件

int close_file(string fileName, int pid);

返回值：0 表示失败，1 表示成功
参数：fileName 文件绝对路径及文件名，pid 进程标识符

第一步：验证文件是否存在于系统打开文件表中，不存在 flag 置 0，直接结束
第二步：验证文件是否被此进程打开，没有则 flag 置 0，结束
第三步：在 SOFT 中定位该表项，若 pn>1，则 pn-1,弹出对应 pid。pn 若等于 1，删除该表项
第四步：在 POFTS 中删除该表项，若此进程的 poft 没有表项了，删除此进程对应的 poft


开始
int close_file(string fileName, int pid);
流程
flag= 1
文件存在于SOFT中?
N
flag= 0
Y
文件是否被此进程打开
N
flag= 0
Y
在SOFT中定位表项
表项pn>1?
Y
pn-1，弹出对应pid,更新表项
N
在SOFT中删除该表项
在POFTS中定位表项
删除该poftEntry
该进程表项数>0?
N
删除该进程对应的poft
Y
结束

7、页面置换

int vm_swap(unsigned char* mem, unsigned char pageNum);
返回值：0 表示失败，1 表示成功
参数：mem 内存地址，pageNum 虚拟内存（页号）下标

将内存某一页的内容和虚拟内存某一页的内容进行交换