操作系统作业6

姓名:**李昱祁** 学号:**PB18071496**

EX1

- **RAID5**：将数据和奇偶校验分散在所有磁盘上。奇偶校验块只保存其它磁盘的块的奇偶校验。

  a. 若只修改一个数据块，则只需访问该数据块以及该数据块所属条的奇偶校验块即可 故总共需要访问的块个数为：

  $$1+1=2(个)$$

  b. 若从某条的边界开始连续修改7 个数据块： 每条中含有 $5-1=4$ 个数据块，所以此情形下修改的数据块分布于相邻的 2 条，还需要额外修改这两条中的奇偶校验块，故总共需要访问的块个数为：

  $$7+2=9(个)$$

  （注：根据异或运算的性质，只要我们知道了更新前的校验位以及更新的数据，我们可以直接计算出新的校验位，而**不必访问**该条中保持不变的数据块 参考：浅谈 RAID 写惩罚与 IOPS 计算）

- **RAID6**：类似于 RAID5，不同之处为每条有两个校验块（另一个可以为差错纠正码校验等）

  a. 若只修改一个数据块，则只需访问该数据块以及该数据块所属条的2 个校验块即可 故总共需要访问的块个数为：

  $$1+2=3(个)$$

  b. 若从某条的边界开始连续修改7 个数据块： 每条中含有 $5-2=3$ 个数据块，所以此情形下修改的数据块分布于相邻的 3 条，还需要额外修改这 3 条中的 2 个校验块，故总共需要访问的块个数为：

  $$7+2\times 3=13(个)$$

EX2

操作系统保存了一个记录有关全部打开文件信息的表，即**打开文件表**。具体来说，有以下两种：

- **整个系统的**打开文件表：包括每个打开文件的 FCB(文件控制块) 的副本以及其它信息
- **每个进程的**打开文件表：包括一个指向整个系统的打开文件表中的适当条目的指针，以及其它信息

第一点，这些关于打开文件的信息是非常重要的，通过它们，OS 可以获取多个进程访问同一文件时每个进程的各自访问状态、以及通过此信息可以判断是否可以对某一文件进行某些操作等等……

第二点，已经打开的文件的相关信息在**打开文件表**里，缓存在内存中，对这些文件执行操作时可以通过**打开文件表**进行索引，不必再次重新检索，提高了性能

EX3

- **文件**：文件是由 OS 定义的逻辑存储单位，由 OS 映射到物理存储单位上；文件是记录在外存上的相关信息的命名组合，从用户角度看，文件是逻辑外存的最小分配单元
- **目录**：目录是一个特殊的文件，其内部格式与文件系统有关。它可以看作是一个符号表，将文件名称(和文件属性等)转换成目录的 entry

$$(755)_O = 111, 101, 101$$

即权限为 **rwxr-xr-x** ，该文件所有者可以读、写、执行此文件；而组中其他成员，以及所有其余用户都只能读、执行该文件，而不能对其进行写操作

EX4

连续分配主要有以下两个问题：

- 为新文件分配空间：管理空闲空间的系统可能更耗时；为防止外部碎片浪费磁盘空间，需要将文件在磁盘间来回复制，大量的合并需要以时间为代价

- 确定一个文件需要多少空间： 连续分配时，某些情况下，新文件的大小无法估计。若为其分配足够大的空间，会造成浪费；若分配一个较小的空间，可能无法进行原地扩展，需要额外处理；即使总量事先已知，但由于文件大小的动态变化，还可能产生内部碎片等等……

下面介绍两种解决方法：

- 为连续分配添加修正方案：文件现有空间不够时，可以分配扩展(extent)，并且在文件块

记录扩展的相关信息。此种方案可以减小连续分配的缺点

- 采用链接分配(linked allocation)：将文件建立为一个链表，文件的信息放入整个链表的各个块中存储。由于链表这种数据结构是非物理连续的，所以只要磁盘中有可用的空闲块，文件就可以增大。

（通过索引分配也可解决，分析略）

EX5

FAT有以下优点：

- 集中管理指针(块号在链表中的顺序)，可靠性更高
- **减小了随机访问的时间**： 通过读入 FAT 信息，磁头可以找到任何块的位置，而不必从头遍历链表
- 在文件创建/删除、文件大小增/减时，性能优秀

主要问题：需要缓存 FAT，因此会增大内存的开销。

EX6

- **case1**：

  1. 读取根目录，获得目录 a/的 inode 编号
  2. 从 inode 表中读取 a/的 inode 结构体，获得存放目录 a/ 内容的块的编号
  3. 读取该块，即读取目录 a/，获得目录 b/的 inode 编号
  4. 从 inode 表中读取 b/的 inode 结构体，获得存放目录 b/ 内容的块的编号
  5. 读取该块，即读取目录 b/，获得文件 c 的 inode 编号
  6. 从 inode 表中读取文件 c 的 inode 结构体，获得存放文件 c 数据的块的编号
  7. 读取存放文件 c 数据的块，即可读出文件 c 的内容

- **case2**：

  1. 读取根目录，获得目录 a/ 的 inode 编号
  2. 读取存放 a/ 内容的块，获得目录 b/ 的 inode 编号
  3. 读取存放 b/ 内容的块，获得文件 c 的 inode 编号
  4. 读取存放文件 c 内容的块

  简单来说，相比上一种情形，此时所有的 inode 结构体已经在内存中，所以读取 inode、获得其中的指针时，不需要进行磁盘 I/O

EX7

若需要存储最大文件，显然要用尽 12 个直接块和 3 个间接块。

由于采用 4 字节 32 位指针，可直接带入 PPT 中公式，计算如下：

$$MAXSIZE = 12 \times 2^{x} + 2^{2x-2} + 2^{3x-4} + 2^{4x-6}$$

带入 $x = 13$，得 $MAXSIZE \approx 2^{46}\ (B) = 2^{6}\ (TB) = 64\ (TB)$

所以此文件系统最大可存储 $64TB$ 的文件

注： 32 位指针最多可管理 2^{32} 个块，则最大容量不难算出为 $32TB$
参考StackOverflow:Understanding the concept of Inodes，关于 32 位指针，8KB block 的文件系统最大可存储文件的大小，该问题下有如下回答：

> If a file grows still larger, then the driver allocates a triple indirect block. Each of the 2048 pointers in a triple indirect block points to a double block. So, under the 32-bit addressing scheme with 32-bit addresses, files up to about 64 TiB could be addressed. Except that you've run out of disk addresses before that (32 TiB maximum because of the 32-bit addresses to 8 KiB blocks).
> So, the inode structure can handle files bigger than 32-bit disk addresses can handle.

而在《现代操作系统》第 4 版第 10 章 6.3 小节，关于Linux ext2 文件系统的介绍中（原书第 448 页，第 6 行），也有如下描述：

> ……(磁盘地址长度为 4 字节）对于块大小是 8 KB 的情况，这个寻址方案能够支持最大 64 TB的文件

此处并没有提及 32 位指针的最大编址空间问题。

所以，**理论上** 文件系统中通过inode索引可支持的最大的文件大小为 $64TB$