OS HW5

1.

a.

RAID-5

- RMW: A_i and A_p, 2 blocks;
- RRW: read A_2, A_3, A_4, write A_1, A_p, 5 blocks.

RAID-6

- RMW: A_i, A_p, A_q, 3 blocks;
- RRW: read A_2, A_3, write A_i, A_p, A_q, 5 blocks.

b.

RAID-5

- RMW: every block with its check block, 14 blocks;
- RRW: A_1, A_2, A_3, A_4, B_1, B_2, B_3, B_4, A_p, B_p, 10 blocks.

RAID-6

- RMW: every block with its two check blocks, 21 blocks;
- RRW: A_1, A_2, A_3, B_1, B_2, B_3, C_1, C_2, C_3, A_p, A_q, B_p, B_q, C_p, C_q, 15 blocks.

2.

disk-scheduling algorithm	order	total distance
FCFS	2150, 2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681	13011
SSTF	2150, 2069, 2296, 2800, 3681, 4965, 1618, 1523, 1212, 544, 356	7586
SCAN	2150, 2296, 2800, 3681, 4965, (4999), 2069, 1618, 1523, 1212, 544, 356	7492
LOOK	2150, 2296, 2800, 3681, 4965, 2069, 1618, 1523, 1212, 544, 356	7424
C-SCAN	2150, 2296, 2800, 3681, 4965, (4999), (0), 356, 544, 1212, 1523, 1618, 2069	9917
C-LOOK	2150, 2296, 2800, 3681, 4965 , 356, 544, 1212, 1523, 1618, 2069	9137

3.

- 系统实现系统调用open()以便进程可以请求访问文件内容。系统调用open()会首先搜索整个系统的打开文件表——**它包含了所有打开文件的信息**——以确定该文件是否被其他进程使用。若是，则在单个进程的打开文件表中创建一个条目，并让其指向现有整个系统的打开文件表——*这样能够节省大量开销*——若尚未被打开，则根据给定文件名搜索目录结构。当文件不再被使用时(close(), delete())，将被进程关闭，操作系统会将其在打开文件表中的入口移除，单个进程表的条目会被删除；用户关闭它时，整个系统的打开文件表的条目会被删除。
- 通过使用打开文件表，打开文件的信息会被缓存在内存中，而文件的索引将由打开文件表完成，不需要每次访问都进行一次目录的遍历；打开文件表还可以保持一个文件的打开状态以确保相关操作的正确运行，综合来看，可以减少IO的开销并提升系统的性能表现。

4.

```
755 = 111 101 101
111：所有者可以读写执行这个文件；
101：文件所有组和其他人可以读和执行这个文件，不能写。
```

5.

问题

- 连续分配将导致产生外部碎片，使得在创建大文件时连续空间不足；
- 当需要向文件中增加内容时，无法满足文件大小的增加。

解决方法

- 将大文件变成许多小份并储存在固定大小的块中；
- 使用链表满足文件大小的动态增长。

6.

优势

- 随机访问时拥有更好的性能表现——当访问位于文件中间的块时，可以直接在FAT中获取对应位置的指针，而不用顺序遍历所有块去得到目标块的地址。而且大部分的FAT可以缓存在内存中，因而可以避开对磁盘的访问。

问题

- 存在某种空间换时间的trade-off，即如果要缓存FAT将会给内存空间带来浪费。

7.

a.

1. root directory
2. /a in inode
3. /a in disk block
4. /a/b in inode
5. /a/b in disk block
6. /a/b/c in inode
7. /a/b/c in disk block

- 7 in total.

b.

- 7 - 3(inode) = 4 in total.

8.

```
every block: 8KB/4B = 2048 indexes
maxsize = 12 * 8 + 2048 * 8 + 2048 * 2048 * 8 + 2048 * 2048 * 2048 * 8 KB = 68,753,047,648KB
```

9.

hard link

- 指向现有文件的目录条目，和原有的条目指向同一个inode，并不会创建新文件内容；
- 创建一个具有两个路径名的文件；
- 链接数+1。

symbolic link

- 软连接就相当于一个文件，会由新的inode为每个软连接其创建；
- 储存路径名；
- 源文件的连接数不会增加。

10.

Data Journaling

1. Journal write: 写事务内容，包括TxB, data, metadata；
2. Journal commit: metadata, data（包括TxE）；
3. Checkpoint: 将更新内容写在磁盘对应位置。

Metadata Journaling

1. Journal write: 写事务内容，包括TxB, metadata, 并且将更新后内容写在磁盘对应位置。两个过程可以同时进行；
2. Journal commit: 写TxE；
3. Checkpoint: 将metadata的更新写在磁盘对应位置。

区别

- 前者在写入文件系统前会先写日志，并且写两次data；
- 后者将metadata写入日志和将data写入文件系统可以同时进行，然后等待日志提交后再将metadata写入文件系统，共写入一次data。

11.

```
Polling, Interrupt, Direct Memnory Access(DMA)
```

12.

```
IO调度，缓冲，缓存，假脱机和设备预留，错误处理，IO保护，电源管理等。
```