Reliablefilesystem

2021219113 2021213595 沈尉林

1、

Q: The FastFile file system uses an inode array to organize the files on disk. Each inode consists of a user id (2 bytes), three time stamps (4 bytes each), protection bits (2 bytes), a reference count (2 byte), a file type (2 bytes) and the size (4 bytes). Additionally, the inode contains 13 direct indexes, 1 index to a 1st-level index table, 1 index to a 2nd-level index table, and 1 index to a 3rd level index table. The file system also stores the first 436 bytes of each file in the inode.

- Assume a disk sector is 512 bytes, and assume that any auxilliary index table takes up an entire sector, what is the maximum size for a file in this system.
- Is there any benefit for including the first 436 bytes of the file in the inode?
  A:

计算

```python
bytes_per_sector = 512
index_size = 4
bytes_in_inode = 436
direct_index_blocks = 13

size_direct_indexes = direct_index_blocks *
bytes_per_sector

entries_per_table = bytes_per_sector // index_size

size_first_level_index = entries_per_table *
bytes_per_sector

size_second_level_index = (entries_per_table ** 2) *
bytes_per_sector

```

```
    size_third_level_index = (entries_per_table ** 3) *
    bytes_per_sector

    max_file_size = size_direct_indexes +
    size_first_level_index + size_second_level_index +
    size_third_level_index + bytes_in_inode
```

Output

```
1   max_file_size = 1082203060 (B)
```

将文件前436字节包含在 inode 中的好处

将文件的前436字节包含在inode中有几个潜在的好处：

1. **快速访问小文件**：对于小于或等于436字节的文件，可以直接从inode访问整个文件，无需读取任何额外的数据块。这可以显著加快对小文件的访问速度。
2. **减少小文件操作的磁盘I/O**：对于只需要读取文件开头的小文件或操作（如文件预览），这种设计避免了额外磁盘I/O的开销。
3. **小文件的高效空间利用**：将小文件直接存储在inode中可以更节省空间，因为它不需要为文件数据分配单独的块。
4. **提高文件元数据操作的性能**：由于文件开头与其元数据一起存储在 inode 中，因此需要同时元数据和文件数据开头的操作（如带预览的文件列表）更为高效。

2、

Q: When user tries to write a file, the file system needs to detect if that file is a directory so that it can restrict writes to maintain the directory's internal consistency. Given a file's name, how would you design a file system to keep track of whether each file is a regular file or a directory?

- In FAT
- In FFS
- In NTFS
  A:

In FAT

- **文件和目录区分**：在 FAT 中，每个文件或目录由父目录中的一个目录项表示。这些目录项包含关于文件或目录的元数据，包括其名称、大小、时间戳和属性。

- **属性字段**：目录项中的一个字段是属性字段。该字段包含一组定义文件或目录特性的位。
- **“目录”属性位**：这些位中的一个用来指示该项是目录还是常规文件。当创建或重命名文件时，文件系统会检查这个位来确定文件的类型。
- **写入限制**：如果文件系统操作尝试向标记为目录的文件写入，FAT文件系统可以轻松检查这个属性，并限制操作以维护目录的完整性。

In FFS

- **Inode 结构**：在 FFS 中，每个文件（包括目录）由一个 inode 表示。inode 包含关于文件的详细元数据。
- **Inode中的类型字段**：inode中的一个字段是类型字段。这个字段指定inode表示的是常规文件、目录、符号链接还是其他文件类型。
- **目录操作**：当执行文件操作时，文件系统会检查inode的类型字段。如果inode类型是目录，某些操作，如直接向其写入数据，可以被限制以确保一致性。

In NTFS

- **MFT 条目**：在 NTFS 中，每个文件或目录由主文件表 (MFT) 中的一个条目表示。每个 MFT 条目包含有关文件或目录的详细元数据。
- **文件属性**：NTFS在每个MFT条目中使用一组属性来描述文件或目录。在这些属性中，有一个指定文件的类型。
- **访问控制和文件类型**：NTFS 以其强大的访问控制机制而闻名。当对文件进行操作时，NTFS 不仅检查文件类型属性，还可以基于文件类型、用户权限和其他安全描述符执行安全策略。

3、

Q: Suppose a variation of FFS includes in each inode 12 direct, 1 indirect, 1 double indirect, 2 triple indirect, and 1 quadruple indirect pointers. Assuming 6 KB blocks and 6-byte pointers.

- What is the largest file that can be accessed with direct pointers only?
- What is the largest file that can be accessed in total

A:

```python
block_size = 6144
pointer_size = 6
num_direct_pointers = 12
num_indirect_pointers = 1
num_triple_indirect_pointers = 2
```

```
num_quadruple_indirect_pointers = 1

pointers_per_block = block_size // pointer_size

largest_file_direct = num_direct_pointers * block_size

total_size = largest_file_direct

total_size += num_indirect_pointers * pointers_per_block
* block_size

total_size += num_indirect_pointers *
(pointers_per_block ** 2) * block_size

total_size += num_triple_indirect_pointers *
(pointers_per_block ** 3) * block_size

total_size += num_quadruple_indirect_pointers *
(pointers_per_block ** 4) * block_size
```

Output

```
largest_file_direct=73728(B)
total_size=67686000294051840(B)
```

4、

Q: Consider a disk queue holding requests to the following cylinders in the listed order: 116, 22, 3, 11, 75, 185, 100, 87. Using the elevator scheduling algorithm, what is the order that the requests are serviced, assuming the disk head is at cylinder 88 and moving upward through the cylinders?
A: 请求将按以下顺序被服务:

1. 从88开始，向上移动，首先服务大于88的请求。按照距离从近到远的顺序：100，116，185。
2. 达到列表中最高的请求后，磁头改变方向，向下移动。
3. 接着服务剩余的小于88的请求。按照距离从近到远的顺序：87，75，22，11，3。

因此，请求的服务顺序是：100，116，185，87，75，22，11，3。

5、

Q: Search for how different RAID versions (at least 5) work differently and list a table to compare them.

A:

RAID 类型	描述	容错能力	随机读写性能	磁盘利用率	磁盘数量要求	应用场景
RAID 0	条带	无	高	高（100%）	n≥1	个人用户
RAID 1	镜像	高	低	低（50%）	2	重要数据存储
RAID 5	分布式奇偶校验条带	有	低	中（(n-1)/n）	n≥3	存储性能与数据安全兼顾
RAID 6	双重奇偶校验条带	有	一般	中（(n-2)/n）	n≥4	数据中心，对数据安全要求高
RAID 10	镜像加条带	有	低	低（50%）	4	性能和安全性的平衡