1. A file requires 256 blocks that are numbered 0, 1, 2, . . . , 255. Suppose that a new block needs to be inserted between Block 63 and Block 64. How many data blocks need to be accessed and how many data blocks need to be relocated, for each of the following allocation schemes.

(a) Contiguous allocation (if there is room for growth only at the end of the file)
(b) Contiguous allocation (if there is room for growth only at the beginning of the file)
(c) Contiguous allocation (if there is room for growth neither at the end nor at the beginning of the file)
(d) Linked allocation without FAT
(e) Linked allocation with FAT
(f) Indexed allocation (single-level, assume that 1024 block addresses can be stored in the index block)

(a) 0, 192
(b) 0, 64
(c) 0, 256 (relocate to another contiguous area of the required size if available, else failure)
(d) 64, 0
(e) 0, 0 (assuming that the FAT is stored in memory)
(f) 0, 0 (assuming that the index block is in memory)

2. An HDD consists of 1 KB block, and has a capacity of 1 TB. How many bits are required to index all the blocks of the HDD? We do not require to allocate a non-multiple of 16 for this indexing. If so, how many bits are required for indexing the blocks of the HDD? What is the largest size of a file if it can be indexed by a single index block? What is the largest size of a file if it can be indexed by two-level indexing? If the list of free blocks is stored in a bitmap, how many blocks are needed to store the bitmap?

1 TB / 1 KB = 1 G = 2^{30}, so 30 bits are needed to index the blocks.
[Rounded up to a multiple of 16] 32 bits (4 bytes).
1 KB block can store 256 block addresses, so largest file size is 256 × 1KB = 256 KB.
For 2-level indexing, largest file size is 256^2 × 1 KB = 64 MB.
(1 G / 8) / 1K = 128 K blocks.

3. Consider a disk of size 256GB, which implements FAT to maintain the file system. The size of the FAT is 16 MB, where each FAT entry is of size 4 bytes, and accessing each FAT entry takes 2ms time (the FAT is buffered in the main memory). Reading each data block takes 100ms. Accessing the directory entry for the file takes 10ms. Compute the time to read a file of size 200KB from the disk.

Number of entries in the FAT table: 16MB/4=4M

Disk block size: 256GB / 4M = 64KB

File size: 200KB requires 4 Blocks

File access:

Directory access (10ms) + first data block (100ms) + FAT for second block (2ms) + second data block (100ms) + FAT for third block (2ms) + third data block (100ms) + FAT for forth block (2ms) + forth data block (100ms) + FAT for fifth block (NULL) (2ms)
= 10ms + 100ms × 4 + 2ms × 4 = 418ms

4. Consider an inode-based organization of a Unix file system. The inode is stored inside a disk block, and in this system, an inode is always accessed from the disk only (that is, an inode never gets buffered in the memory). Assume that 192 bytes of an inode are used to store the file attributes. Inside the inode, there exists one single indirect pointer, one doubly indirect pointer, and one triply indirect pointer. The rest of the inode stores direct block pointers. Disk block size is of 8KB, disk block pointer is 32 bits long, and disk bandwidth is 16KB/Sec. Estimate the minimum and maximum file sizes, whose random/direct access takes exactly 1.5 secs.

Inside inode, pointer storage space 8192 – 192 = 8000 bytes

Number of pointers inside inode = 8000 / 4 = 2000 (total)

So, number of direct pointers = 2000 – 3 = 1997

Bandwidth 16KB/sec. Block size 8KB. Direct access takes 1.5 sec means it requires 3 block access (one inode, one single indirect, one data block)

Minimum and maximum file size which is accessed via single indirect pointers:

Min: 1997 × 8KB + 1 = 1997 × 8KB + 1 byte = 16,358,401 bytes

Max: 1997 × 8KB + 2048 × 8KB = 1997 × 8KB + 2048 × 8KB = 32,360 KB ≈ 31.6 MB

5. A file *foobar.mkv* is of size 987,654,321 bytes. The HDD storing this file is composed of 1 KB blocks. In each of the following cases compute the number of blocks needed to store the metadata and the number of blocks needed to store the data (content of the file).

(a) Linked allocation without a FAT
(b) Linked allocation with a FAT
(c) Indexed allocation (use an many levels as is required)

(a) $\lceil 987{,}654{,}321 / (1024 - 4) \rceil = 968{,}289$ data blocks

(b) $\lceil 987{,}654{,}321 / 1024 \rceil = 964{,}507$ data blocks

(c) Same as (b)

Requirement for metadata: By Exercise 2, maximum file size for 2-level indexing is 64 MB. Our file is bigger than that. Three-level indexing can store a maximum file of size $256^3 \times 1$ KB = 16 GB. Our file fits in that space. The file has 964,507 data blocks. We need $\lceil 964{,}507 / 256 \rceil = 3768$ index blocks in the third level, $\lceil 3768 / 256 \rceil = 15$ index blocks in the second level, and (of course) 1 index block at the first level.

6. Again consider the file *foobar.mkv* of size 987,654,321 bytes, and an HDD with 1 KB blocks. Explain the organization of the file (metadata and data) if the HDD implements Unix's inode-based indexing with 12 direct pointers, and one indirect pointer for each of the three levels.

Suppose that the 123,456,789-th byte of the file is needed. Explain all disk-block accesses required to read this byte. Assume that the top-level inode is stored in main memory. All other blocks need to be read from the disk.

See the EndSem paper of 2024 [Q5(e)] for a similar situation.

7. Think of a hypothetically small hard drive with only 100 blocks (numbered 0 – 99). Assume that each block can hold only four block pointers. At some point of time, the following blocks are free.

47, 89, 25, 33, 3, 21, 40, 94, 6, 61, 44, 37, 97, 80, 73

The grouping method is used to store this information (in the given sequence) in the free blocks themselves.

(a) Describe how this can be achieved using a minimum number of free blocks.

(b) If four free disk-blocks are given to a new file, explain which blocks does this file gets (in what sequence), and how the grouping information changes.

(c) Now, if a file is deleted adding the free blocks 11, 59, and 34, how do you efficiently update the grouping information?

(a) External information: 47
Block 47: 89, 25, 33, 3
Block 3: 21, 40, 94, 6
Block 6: 61, 44, 37, 97
Block 97: 80, 73, NULL, NULL

(b) Free blocks given to the new file: 89, 25, 33, 47

External information: 3
Block 3: 21, 40, 94, 6
Block 6: 61, 44, 37, 97
Block 97: 80, 73, NULL, NULL

(c) The best way to add free blocks is to keep an additional external information (last block number of the free-blocks list).

External information: 3, 97
Block 3: 21, 40, 94, 6
Block 6: 61, 44, 37, 97
Block 97: 80, 73, NULL, NULL

changes to

External information: 3, 59
Block 3: 21, 40, 94, 6
Block 6: 61, 44, 37, 97
Block 97: 80, 73, 11, 59
Block 59: 34, NULL, NULL, NULL