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. 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.

4. 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.