1 File access

Files will be composed of a number of blocks. Files are **sequential** or **random** access. Random access is essential for example in **database systems**.

2 Contiguous allocation

Contiguous file systems are similar to *dynamic partitioning* in memory allocation:

- Each file is stored in a **single group** of adjacent blocks on the hard disk E.g. 1KB blocks, 100KB file, we need 100 contiguous blocks
- Allocation of free space can be done using **first fit, best fit, next fit, etc**.

2.1 Advantages

- **Simple to implement**: only location of the first block and the length of the file must be stored (in the directory entry)
- **Optimal read/write performance**: blocks are co-located/clustered in nearby/adjacent sectors, hence the seek time is minimised (remember the example in lecture on disks!)

2.2 Disatvantages

- The exact size of a file (process) is not always known beforehand: what if the file size exceeds the initially allocated disk space
- Allocation algorithms needed to decide which free blocks to **allocate** to a given file (e.g., first fit, best fit, etc.)
- Deleting a file results in *external fragmentation*: de-fragmentation must be carried out regularly (and is slower than for memory)

Contiguous allocation is still in use: CD-ROMS/DVDs External fragmentation is less of an issue here since they are write once only

3 Linked list allocation

To avoid *external fragmentation*, files are stored in separate blocks (similar to paging) that are **linked** to one another. Only the **address** of the first block has to be stored to locate a file. Each block contains a **data pointer** to the next block (which takes up space)

3.1 Advantages

- **Easy to maintain**: only the first block (address) has to be maintained in the directory entry
- File sizes can grow **dynamically** (i.e. file size does not have to be known beforehand): new blocks/sectors can be added to the end of the file
- Similar to paging for memory, **every** possible block/sector of disk space can be used: i.e., there is no external fragmentation!
- Sequential access is straightforward, although more seek operations/disk access may be required

3.2 Disadvantages

- Random access is **very slow**, to retrieve a block in the middle, one has to walk through the list from the start
- There is **some** internal fragmentation; on average the last half of the block is left unused
- **Internal fragmentation** will reduce for smaller block sizes
- May result in **random disk access**, which is very **slow** (remember the example in lecture on disks)
- Larger blocks (containing multiple sectors) will be **faster**
- Space is lost within the blocks due to the pointer, the data in a block is no longer a power of 2!
- **Diminished reliability**: if one block is corrupt/lost, access to the rest of the file is lost

4 File allocation tables

Advantages:

- Block size remains power of 2, i.e., no space is lost due to the pointer
- Index table can be kept in memory allowing **fast** non-sequential/random access (one still has to walk through the table though)

Disadvantages:

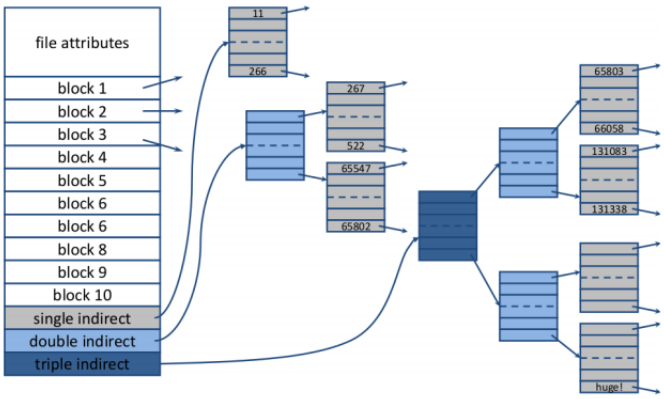
- The size of the file allocation table grows with the number of blocks, and hence the size of the disk
- For a 200GB disk, with a 1KB block size, 200 million entries are required.
- Assuming that each entry at the table occupies 4 bytes, this requires 800MB of main memory!

5 I-nodes

Each file has a small data structure (on disk) called **I-node** (index-node) that contains its attributes and block pointers.

- In contrast to FAT, an I-node is **only loaded** when the file is open (stored in system wide open file table)
- If every I-node consists of n bytes, and at most k files can be open at any point in time, at most nk bytes of main memory are required

I-nodes are composed of **direct block pointers** (usually 10), **indirect block pointers**, or a combination thereof (e.g., similar to multi-level page tables)



5.1 Directories

- In UNIX, all information about the file (type, size, date, owner, and block pointers) is stored in its i-node.
- Therefore, directory tables are very simple data structures composed of file name and a pointer to the i-node.
- Note that directories are no more than a special kind of file, so they have their own i-node.

5.2 Lookups

Opening a file requires the disk blocks to be located. Absolute file names are located relative to the **root** directory. Relative file names are located based on the **current working directory**.

Locate the root directory of the file system
- Its i-node sits on a fixed location at the disk (the directory itself can sit anywhere)
Locate the directory entries specified in the path:
- Locate the i-node number for the first component (directory) of the path that is provided
- Use the i-node number to index the i-node table
and retrieve the directory file
- Look up the remaining path directories by
repeating the two steps above
Once the files directories have been located, locate the files i-node and cache it into memory

5.3 Sharing files

There are two approaches to **share a file**, e.g. between directory B and C, where C is the real owner:

- Hard links: maintain two (or multiple) **references to the same i-node** in B and C. (In Unix: ln /C/file1 /B/file2) the i-node link reference counter will be set to 2
- Symbolic links: The **owner maintains a reference** to the i-node in, e.g., directory C The referencer maintains **a small file** (that has its own i-node) that contains the location and name of the shared file in directory C. (In Unix: ln -s /C/file1 /B/file2)

Reference section

external fragmentation

External fragmentation is the various free spaced holes that are generated in either your memory or disk space.