Exercises from old exams

Mass memory and File System (exercises on ch 11, 13 and 14)

1. Consider two file systems F1 and F2, on two different volumes, based on FAT and inode, respectively. Pointer (indexes) have size 32 bits, disk blocks have size 4KBytes, and both volumes are split into a part reserved to metadata (directories, FAT e FCB (File Control Block) or Inode, index blocks, etc.) and a part reserved to data blocks. The part reserved to data blocks have the same size in F1 and F2. The FAT in F1 have size 150 MB.

   How many data blocks can be present in the two file systems?

   - N blocks in F1: Each entry in the FAT corresponds to a data block, so N_{F1} *= 150 MB / 4B = 37,5 M blocchi*

   - N blocks in F2: $N_{F2} = N_{F1}$ *= 37,5 M blocks (by definition)*

   Which is the maximum number of available free data blocks?

   - F1: *(same as the used data blocks) 37,5 M blocks*

   - F2: 37,5 *M blocks (as for F1)*

   Now suppose that free blocks in F1 are handled by a free list.

   - Can the free list in F1 be implemented directly within the FAT?

     (YES/NO, motivate) YES. The FAT contains both the lists of blocks used by files and the free list. There is no reason to look for alternative strategies, as the index (in the FAT) corresponding to free blocks are available (not used for files) so they can be exploited in order to implement the free list.

   Let's handle free blocks in F2 by a linked list of index blocks, each one containing a pointer to the next index block in the list and N_{Free} pointers to free data blocks.

   - Is this kind of representation possible/allowed (while the inode standard representation is adopted for files)?

     (YES/NO, motivate) YES. The inode representation doesn't imply any constraint on the way free blocks are organized. So, though free blocks could be handled as an additional file, handled by means of an inode, a list of index nodes is also possible. As an alternative, a bitmap as well as a list of linked free blocks are also possible.

   - (If YES, at previous question)
     - *Which is the value of* N_{Free}*? N_{Free} = 4KB/4B-1 = 1023 (total number of pointers in a block - one pointer, to the next index block in the linked list)*
     - *How many index blocks are needed, at maximum, for the free list in F2? Each index block points to N_{Free} = 1023 free data blocks. The maximum number of free data blocks is thus N_{F2} = 37,5 M blocks. The maximum number of index blocks in the free list is $N_{IndFree}$ = 37,5 M / 1023 = 37,5 K *1024/1023 ≈ 37,5 K*

   - ~~(If NO, at previous question)~~
     ~~Provide an alternative way of organizing free blocks~~
     ……
     ……

   Assume now that file "a.mp4" (in F1) has size 317 MB and file "b.mp4" (in F2) has size 751 MB.

   - How many data blocks are needed/allocated for a.mp4 ? *317 MB / 4 KB = 317/4 K = 79,25 K blocks*

   - How many indexes in the FAT are used for a.mp4 ? *79,25 K (same as the number of data blocks)*

   - How many data and index (inode excluded) blocks are used/allocated for b.mp4 ?
     DATA: 751 MB / 4 KB = 751/4 K = 187,75 K
     INDEX: 10 data blocks are directly indexed/pointed by the inode. The single indirect index block points to 1024 = 1K data blocks. 1 index block is needed.
     Double indirect: the outer level is made by 1 index block, the inner level has ⌈(187,75 K-10) / 1K⌉ - 1 = 187 index blocks (double inder)
     Total: 1 (singole) + 1 (duble outer) + 187 (double inner) = 189

2. Consider a kernel module that handles the scheduling problem for disk access requests.

3a) Why the SCAN and C-SCAN policies are not meaningful/needed with SSD disks?

Because they are policies oriented to reduce the total distance traveled by the read/write head (computed as the sum of differences between block/cylinder indexes): this is meaningful for magnetic disks (HDDs), characterized by seek time. As SSDs do not pay any cost for moving across different block indexes, the policy is not needed, nor would it have any impact.

Is it possible for a system to combine a magnetic disk (HDD) A of size 500GB and an SSD disk B of size 200GB, organized as a single volume?

YES. The disk technology is a problem characterizing lower driver layers. Disk partitioning and formatting allow both to generate multiple volumes on a single device, as well as to join multiple devices in a single volume. Of course, performance will vary depending on the device.

(If YES was the answer) what is the size of the volume? 500GB + 200GB = 700GB

3b) The system (*WARNING: this is another sub-question, unrelated from the previous one*) needs to support paging and swapping. Which solution can you consider as more efficient, between a raw swap partition and a swapfile in an existing filesystem? (motivate)

The raw partition is more efficient, in terms of overall transfer time, as it allows direct handling of disk blocks, without the overhead of file system management.

3c) Let's aim at implementing a new filesystem, supporting shared files and directories (so the directory tree is a DAG!).

Why possible cycles represent a problem for the graph of directories?

Exactly because cycles prevent a graph from being a DAG, which means that a directory would have an parent/predecessor descendant/successor in the DAG (so not a DAG any more) This has to be avoided.

Which strategies could be adopted in order to face the above problem (cycles)?

a) *Share just files (leaves), not directories (but this would limit the initial request: "*shared files and directories*")*
b) *Garbage collection: cycles are accepted (momentarily, in order to avoid checks after every creation of a link/sharing to an existing file/directory), but a check function is activated form time to time, in order to find and remove possible cycles in the graph*
c) *After every creation of a new link (to an existing file/directory) a check is done, which means traversing the graph in order to fine a possible newly generated cycle.*

3. Consider a Unix-like file system, based on inodes, with 13 pointers/indexes (10 direct, 1 single indirect, 1 double indirect and 1 triple indirect). Pointers/indexes have size 32 bits, and disk blocks have size 2KB. We know that the file system contains 1000 files of average size 15MB, and that the total internal fragmentation is 1MB. Compute the maximum possible number of files with triple indirect indexing (N3) and with double indirect indexing, that cam be present in the file system.

R1	***General observations*** *An index block contains 2KB/4B = 512 pointers/indexes.* *The total size (net, independent of the chosen file system) of the files is 15MB * 1000 = 15000MB (=* 15GB approximately) *The disk occupancy for data (so the number of allocated data blocks, which depends on the adopted file system), considering the known internal fragmentation, is 15001MB = 15001*512 blocks.* ***Computing maximum numbers N2/N3*** *(for files with* double/triple indirect indexing) In order to compute the maximum number of files, we need to consider the minimum occupancy. *Let's use MIN_2 and MIN_3 for minimum occupation of the two kinds of file:* $MIN_2 =$ (10 + 512 + 1) blocks $MIN_3 =$ (10 + 512 + 512^2 +1) blocks N2 = $\lfloor$15001 * 512 / 523$\rfloor$ = 14685 N3 = $\lfloor$15001 * 512 / (523 + 512^2)$\rfloor$ = 29 *N2 is greater than the known number of files in the file system (1000), so the maximum number of double indirect files (N2) is limited to 1000.*

Which is the maximum possible number of files without (so 0) internal fragmentation (*motivate)*?

R2	***Observations*** *The total internal fragmentation is known (1MB): this could be split in any number of files from 0 to 1000, as some files could have 0 fragmentation. The maximum number of files with 0 fragmentation ($N0_{MAX}$) can be obtained by assuming that the internal fragmentation (1MB) is split over the minimum possible number of files N_{min}, so ty assuming that they have a maximum internal fragmentation of 1 block – 1 Byte each* ***Computation*** *$N_{min} = \lceil 1MB / (2KB-1B) \rceil = \lceil 1MB / 2047 \rceil = 513$* *$N0_{MAX} = 1000 – 513 = 487$*

We know that a file "a.dat" needs 2000 index blocks. How many data blocks does the file occupy? Which is the size of "a.dat" *(if the size cannot be uniquely computed, compute for it a minimum and maximum value)*?

R3	***Observations*** *The number of index blocks needed by a file requiring the double indirect indexing is, when fully using the inner level: 1 (single indirect) + 1 (double outer) + 512 (double inner).* *2000 index blocks thus require a triple indirect file, of such a size that in needs 2000 – 514 = 1486 index blocks for the "triple" part.* ***Computation*** *We need to compute how many triple index blocks are needed at first (this is simple, just 1), second and third level:* *$\lceil 1486/512 \rceil = 3$ => 1 (triple, I lev.) + 3 (triple, II lev.) + 1482 (triple, III lev.)* ***Occupancy:** number of data blocks NBL (the minimum and maximum depend on how many indexes are actually present in the last index block (min 1 – max 512)):* *$NBL_{min} = 10$ (direct) + 512 (single) + 512*512 (double) + 1481*512 + 1 (triple) = 1020939* *$NBL_{MAX} = NBL_{min} + 511 = 1021450$* ***Size:** Minimum size is attained assuming maximum internal fragmentation (2047 Bytes) – Maximum size assuming 0 fragmentation.* *$DIM_{min} = NBL_{min} * 2KB – 2047B$* *$DIM_{MAX} = NBL_{MAX} * 2KB$*

4. *Consider a* "solid state" (SSD) disk. What operation is limiting its life/duration?

The duration of an SSD disk is limited by the maximum number of allowed cancel/erase operations. Erase operations are done per blocks/pages, and they have an impact on the total number of writes, as a write (re-write) has to be preceded by a cancel/erase.

Now suppose an SSD disk of size 1TB (TeraByte) has a 3 years warrantee, ant the disk supports a maximum of 5TB rewrites per day. Which is the (total) maximum number of Bytes that can be read and written in the entire life of the disk? (two answers needed, one for read and one for write operations)

Read: no limit, as read operations are not a problem
Write: the limit is given by the daily limit, multiplied by the days in 3 years.
$N_{write,max} = 5TB * 3*365 = 5*1095 TB \approx 5.5*10^3 TB$

Let's consider a RAID disk structure. What do we mean, on that structure and the related failure rate and Mean Time To Failure (MTTF), with the following terms?

- Mean Time To Repair (MTTR):

-

Mean Time needed in order to repair (and recover data) a disk after a failure.

- Mean Time To Data Loss (MTTDL):

Mean Time (inverse of rate/probability) occurring between two data losses (or until the first data loss): data loss occurs when a second failure happens during repair time, for a first failure (before the device had been repaired).

Consider a RAID structure with 2 discs in *"mirrored" arrangement*. Suppose each of the 2 disks can have a failure (independently each other), with MTTF = 50000 hours and MTTR = 20 hours, compute the resulting MTTDL

*(for a detailed explanation, see Silberschatz slides: **ch11plus**)*
$MTTDL = MTTF^2/(2*MTTR) = 50000^2/(2*20)$ hours $= 25/4*10^7$ hours $= 6.25*10^7$ hours