Assignment 23

Consider three different File Allocation Table (FAT)-based file systems: the first has only 10 KB clusters, the second has only 1 KB clusters, the third has only 0.5 KB clusters:

1. In this file system, 1000 files of size 0.5 KB, 100 files of size 5 KB, and 10 files of size 50 KB shall be stored at the same time. How large (in KB) is for each of the three above cluster sizes the amount of storage that cannot be used due to internal fragmentation?
2. The FAT shall be used to manage clusters that comprise altogether exactly 1500 KB. (The space required for storing the FAT itself shall not be included in the 1500 KB.)
   (a) For each of the above three cluster sizes, investigate whether it is at all possible to manage all the files from part 1. using the above FAT? Justify your answer for each cluster size!
   (b) Now, the FAT itself shall be stored in clusters: How many clusters are occupied by the above FAT at each of the three cluster sizes provided that cluster numbers are stored within the FAT using 16 Bits ("FAT16")?

Solution

1) A cluster is an individual unit of storage on the hard disk. When you save a file to the hard disk, the file is saved in clusters. The hard disk can't work with partial clusters. Therefore, even a 1-byte file occupies an entire cluster. The amount of space that's wasted in such a situation is determined by the cluster size.

Three different FAT systems with size clusters:
10 KB = 10240 bytes
1 KB = 1024 bytes.
0.5 KB = 512 bytes.

0.5 KB = 512 bytes. 512 * 1000 = 512.000 bytes.
5 KB = 5120 bytes. 5120 * 100 = 512.000 bytes.
50 KB = 51200 bytes. 51200 * 10 = 512.000 bytes.

If file size is not a multiple of block size, the last block is only partially used (internal fragmentation). Disadvantage of huge cluster sizes: larger internal fragmentation!
For 10 KB size cluster:
Start storing the files larger than cluster size as there is no fragmentation. Each 50KB file is stored in 5 * 10KB clusters with no internal fragmentation.
Each file of size 5KB occupying a 10KB cluster leaves 5KB internal fragmentation for each 5KB file. 100 * 5KB = 500KB internal fragmentation.

Each file of size 0.5KB occupying 10KB cluster leaves 9.5KB internal fragmentation. 1000 * 9.5KB = **9.500KB internal fragmentation.**

Therefore there is 500KB + 9.500KB = 10.000KB internal fragmentation using 10KB size clusters.

For 1KB size clusters:
Start by storing files larger than cluster size as there is no internal fragmentation (FAT uses pointers to store files in non-contiguous blocks/clusters). Each 50KB file is stored in 50 * 1KB clusters with no internal fragmentation.

Each 5KB file is stored in 5 * 1KB clusters with no internal fragmentation.

Each 0.5KB file stored in 1KB cluster leads to 0.5KB internal fragmentation. 1000 * 0.5KB = **500KB internal fragmentation.**

For 0.5KB clusters there is no internal fragmentation as each file size is larger or same size as cluster, so no unused space in a cluster.

2) 1500KB * 1024 = 1.536.000 bytes.

a) 10KB clusters require 5 clusters for every 50KB file, there are 10 * 50KB files, so therefore 5 * 10 = 50 clusters needed.
10KB clusters require 100 clusters for 5KB files, and 1000 clusters for 0.5KB files. **Total of 1150 clusters needed.**

Since 1150 clusters are required, and each cluster is 10KB, the total size 10*1150 = 11.500KB far exceeds 1500KB and does not work.

1KB clusters require 50 clusters for every 50KB file, 10 * 50 = 500 clusters for the 50KB files. 5KB files require 5 clusters each, 100 * 5 = 500 clusters required for 5KB. Then 1000 clusters needed for the 0.5KB files. **Total = 2.000 clusters required.**

Since 2.000 clusters are required, and each cluster is 1KB, and 2.000KB > 1.500KB, it does not work.

0.5KB clusters require 100 clusters for every 50KB file, 10*100 = 1000 clusters. Every 5KB file requires 10 clusters, 10*100 = 1000 clusters. Then 1000 clusters for the 0.5KB files. **Total = 3.000 clusters required.**

3.000 clusters * 0.5KB = 1.500KB works.

b) 16 bits = 2^4 bits = 2 bytes. Each cluster contains a pointer to the next block, taking up 2 bytes of space.
1150 clusters for 10KB clusters. 2000 clusters for 1KB clusters. 3000 clusters for 0.5KB clusters.

Cluster entry number is 2 bytes.
For cluster size 10KB : 1150 * 2 = 2300 bytes is the total size for the entry bit. Each cluster is 10240 bytes, so 2300/10240 = 0,2246 approximates to 1.
For cluster size 1KB : 2000 * 2 = 4000 bytes is the total size for entry bit. Each cluster is 1024 bytes, so 4000 / 1024 = 3,90625 approximates to 4.
For cluster size 0.5KB : 3000 * 2 = 6000 bytes, each cluster is 512 bytes, so 6000 / 512 = 11,71875 which approximates to 12.

Assignment 24

Consider an I-node-based file system:

1. Each I-node supports storing 98 direct block pointers, one pointer to a single indirect block, and one pointer to a double indirect block. Each indirect block supports storing 100 block pointers. Each block has a size of 1 KB. What is the maximum file size that can be managed in such a file system?

2. Consider the absolute path /usr/bin/vi. Which disk accesses are required to check whether the file vi really exists in the directory /usr/bin? Assume that the root directory has already been loaded into a cache of the operating system and thus no disk access is required to know the contents of the root directory. Note: Do not refer to abstract disk accesses such as "read directory", but mention really each individual disk access that may be needed to load data blocks (which contain, e.g., file contents or directory contents), metadata such as I-nodes, or anything else; in addition to mentioning the disk accesses themselves, describe what kind of information is used from each of these accesses.

3. Consider the following system calls and describe for each the involved file system updates:

   (a) link("ParentDirectory/existingFile","ParentDirectory/newFile") to cre- ate a hard link newFile that points to existingFile.

   (b) symlink("ParentDirectory/existingFile","ParentDirectory/newFile") to create a symbolic link newFile that points to existingFile.

   (c) unlink("ParentDirectory/existingFile") that deletes the additional hard link ParentDirectory/existingFile to a file (i.e. after deleting this hard link, still other hard links remain pointing to the same file).

Note: Describe the filesystem updates in terms of "Allocate a data block XYZ with content ABC", "Create an I-node pointing to data block XYZ", "Add to ABC a directory entry pointing to XYZ", "Update part ABC of I-node XYZ", "Update part ABC of directory entry XYZ", etc.