

## CSI3131 – Operating Systems

### Tutorial 9 –File Systems

1. Consider a file currently consisting of 100 blocks. Assume that the file control block (and the index block, in the case of indexed allocation) is already in memory. Calculate how many disk I/O operations are required for contiguous, linked, and indexed (single-level) allocation strategies, if, for one block, the following conditions hold. In the contiguous-allocation case, assume that there is no room to grow in the beginning, but there is room to grow in the end. Assume that the data of the added block is stored in secondary memory.

Condition	Contiguous	Linked	Indexed
The block is added at the beginning.			
The block is added in the middle.			
The block is added at the end.			
The block is removed from the beginning.			
The block is removed from the middle.			
The block is removed from the end.			

2. Why must the bitmap for file allocation be kept on mass storage, rather than in main memory?
3. Explain how the VFS layer allows an operating system easily to support multiple types of file systems.

4. Examine the listing on page 5 that provides the contents of a MINIX file system. The listing was generated using *ls* with the following arguments:

- l : Long listing format, displays many of the attributes found in the inode.
- i: Provide the inode number
- a: List all entries in directories (including the hidden files that start with the “.”).
- R: Recurse into subdirectories (the listing starts at the root of the MINIX file system, attached at */root/mnt1*).

(a) Given the following information and the information in the listing, complete the information found in the inode table shown on page 3. Be sure to fill in the index block if an indirect index is used in the inode.

- The numerical user id for *root* is 0 and for *test1* is 500. Use these values in the *i\_uid* field of the inode.
- The numerical group id for *root* is 0 and for *test1* is 500. Use these values in the *i\_gid* field of the inode.

- The *i\_mode* field is a 16 bit field that contains the following pattern:

tttt sss rwx rwx rwx

where the lower 9 bits represent the user, group, and other permissions, the three bits *sss* are special system bits, and the upper four bits *tttt* identify the type of inode. For the *i\_mode* values in the table below provide a bit pattern where the permission bits reflects the permissions of the file, the special bits *sss* are set to zero, and the upper four bits *tttt* are set to one of the following:

1010 – if the inode is associated to a symbolic link

1000 – if the inode is associated to a regular file

0100 – if the inode is associated to directory

- The *i\_size* field of the inode reflects the number of bytes stored in the file/directory/symbolic link.
- The *i\_time* field of the inode is a 32 bit number that gives the time the file was last modified. For this exercise enter the date and time of modification given in the listing.
- The *i\_nlinks* field contains the number of times a file is referenced by the directory or the number of subdirectories (including . and ..) contained in a directory.
- The *i\_zone* field contains 9 block numbers. The first 7 numbers provide the numbers of the first seven data blocks allocated to the inode (i.e. file, directory, etc.). The eight number refers to a block that is an index block (i.e. index array of subsequent data blocks). The ninth number refers to a double indirect index block. Fill in the *i\_zone* with data block numbers given that the first data block is block number 524, the inodes were created in sequence 1 to 13, and that the contents of the files/directories were allocated contiguous blocks. If an index block was used, fill in an index block below the table. Note that no double indirect index block was used.
- Blocks have a size of 1 Kbytes. The inode is 32 bytes long with fields sizes defined as follows: *i\_mode* 2 bytes; *i\_uid* 2 bytes; *i\_size* 4 bytes; *i\_time* 4 bytes; *i\_gid* 1 byte; *i\_nlinks* 1 byte; *i\_zone* 18 bytes (2 bytes per block number).

(b) On page 4, complete the contents of the directory tables and include arrows between tables to show the tree structure. Add files as circles or ellipses and include them in the directory tree.

# MINIX File System.

Boot	SB	Inode Map	Zone Map	Inode Table	Data Zones
------	----	--------------	-------------	-------------	------------

inode #	i_mode	i_uid	i_size	i_time	i_gid	i_nlinks	i_zone[0..8]
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14	0	0	0	0	0	0	0

<b>Index Block</b> Data Block #:

<b>Index Block</b> Data Block #:

<b>Index Block</b> Data Block #:

Directory:	
Ino #	Name

Directory:	
Ino #	Name

Directory:	
Ino #	Name

Directory:	
Ino #	Name

Directory:	
Ino #	Name

Directory:	
Ino #	Name

Directory:	
Ino #	Name

Directory:	
Ino #	Name

Inode number  
 Inode type:  
 -: file  
 d: directory  
 l: symbolic link  
 Permissions  
 Link count  
 Group Id  
 Size (bytes)  
 Time modified  
 Owner Id  
 Name

```

[root@site:dev mnt1]# ls -laR .
.:
total 7
 1 drwxr-xr-x  4 root    root    128 Mar 27 09:24 .
96770 drwxr-x--- 8 root    root    4096 Mar 27 09:47 ..
 2 drwxr-xr-x  5 test1   244    160 Mar 27 09:24 dir1
 3 drwxr-xr-x  4 test1   244    160 Mar 27 09:30 dir2

./dir1:
total 5
 2 drwxr-xr-x  5 test1   244    160 Mar 27 09:24 .
 1 drwxr-xr-x  4 root    root    128 Mar 27 09:24 ..
 4 drwxr-xr-x  2 test1   244     96 Mar 27 09:36 ad
 5 drwxr-xr-x  2 test1   244     96 Mar 27 09:29 bd
 6 drwxr-xr-x  2 test1   244     96 Mar 27 09:30 cd

./dir1/ad:
total 6
 4 drwxr-xr-x  2 test1   244     96 Mar 27 09:36 .
 2 drwxr-xr-x  5 test1   244    160 Mar 27 09:24 ..
13 -rw-r--r--  1 test1   244   3159 Mar 27 09:36 afile

./dir1/bd:
total 4
 5 drwxr-xr-x  2 test1   244     96 Mar 27 09:29 .
 2 drwxr-xr-x  5 test1   244    160 Mar 27 09:24 ..
 9 -rw-r--r--  1 test1   244   1053 Mar 27 09:29 f1

./dir1/cd:
total 14
 6 drwxr-xr-x  2 test1   244     96 Mar 27 09:30 .
 2 drwxr-xr-x  5 test1   244    160 Mar 27 09:24 ..
10 -rw-r--r--  2 test1   244   10530 Mar 27 09:30 f2

./dir2:
total 7
 3 drwxr-xr-x  4 test1   244    160 Mar 27 09:30 .
 1 drwxr-xr-x  4 root    root    128 Mar 27 09:24 ..
 7 drwxr-xr-x  2 test1   244     96 Mar 27 09:31 d1
 8 drwxr-xr-x  2 test1   244     96 Mar 27 09:36 d2
11 -rw-r--r--  1 test1   244   2106 Mar 27 09:30 file1

./dir2/d1:
total 14
 7 drwxr-xr-x  2 test1   244     96 Mar 27 09:31 .
 3 drwxr-xr-x  4 test1   244    160 Mar 27 09:30 ..
10 -rw-r--r--  2 test1   244   10530 Mar 27 09:30 file2

./dir2/d2:
total 3
 8 drwxr-xr-x  2 test1   244     96 Mar 27 09:36 .
 3 drwxr-xr-x  4 test1   244    160 Mar 27 09:30 ..
12 lrwxrwxrwx  1 test1   244     13 Mar 27 09:36 sd -> ../../dir1/ad
  
```

5. Using the directory from question 4 (do not include f2 and sd), complete the FAT table and directory tables below to show how the FAT file system would represent the same directory. Assume that clusters (blocks) are 1Kbyte in size. Note that in this case the attributes have been included in the directory tables. Also note that it is not possible to represent hard links in FAT.

Notes:

- The DOS name is an 8 character field (bytes) that is padded with spaces. The first character has special interpretations: set to 0x00 for a directory entry never used, set to \$E5 for not currently used entry (i.e. deleted files/directories).
- The DOS extension field is a 3 character field (3 bytes) that is padded with space characters (i.e. contains 3 space characters when no extension is specified).
- The type field (also called the attribute field) contains eight bits interpreted as follows:
  - Bits 7 and 6 (most significant bits): not used
  - Bit 5 – Archive bit
  - Bit 4 – Sub directory bit
  - Bit 3 – Volume Label bit
  - Bits 2 and 1 – Not used
  - Bits 0 – Read only bit
- Time/Date fields contains access and modification times for the entry. You need not fill in these fields for this exercise.
- The First Cluster field provides the cluster identifier (block number) of the first allocated cluster to the entry (file/directory). The number is also used as an index into the FAT to get the other clusters allocated.
- The size field is used to provide the sizes of files. It is not used for subdirectory entries.
- There are a number of types of directory entries defined in FAT as follows:
  - File Entry (for recording files)
  - Volume Label Entry (used to record a file system name, found only in the root directory))
  - Sub-directory Pointer Entry (for defining subdirectories)
  - Sub-directory Identifier Entry (. entry – always first directory entry)
  - Sub-directory Parent Pointer Entry (.. entry – always second directory entry)
  - Not-currently-in-use Entry (DOS name field starts with 0xE5)
  - Never-used Entry (DOS name field starts with 0x00)
- File Entry specifics
  - The volume label and sub-directory bits are set to zero and the DOS name field starts with a valid character. Set the type field to 00 for file entries.
- Sub-directory Pointer Entry
  - The sub-directory bit is set to 1 and the DOS name field starts with a valid character. The Volume Label field is set to zero. Set the type field to 10 for sub-directory pointer entries.
  - Sub-directory pointer entries cannot occupy the first or second entries in the directory table.
- Sub-directory Identifier Entry
  - The sub-directory bit is set to 1 and appears as the first directory entry. The DOS name field contains “.” (padded with space characters) and has the

extension set to 3 space characters. This entry is mandatory in all directory tables except for the root directory table. Set the type field to 10 for sub-directory identifier entries.

- Sub-directory Parent Pointer Entry
  - The sub-directory bit is set to 1 and appears as the second directory entry. The DOS name field contains “..” (padded with space characters) and has the extension set to 3 space characters. This entry is mandatory in all directory tables except for the root directory table. Set the type field to 10 for sub-directory parent pointer entries.
  - If the parent directory is the root directory, the first cluster number is set to 0000.
- Setup of the FAT file system:
  - First block is reserved as the master boot record (MBR) that contains fields describing the file system (e.g. size of sectors, number of sectors per cluster, number of FAT tables, number of entries in the root directory etc.).
  - The file system shall be contained in a 64 MByte partition.
  - The FAT table is 256 clusters long (2 bytes X 64 Kentries / (1024 bytes/cluster) )
  - Two FAT tables are recorded.
  - The root directory contains 64 entries (32 bytes longs) and thus occupies 2 clusters (32 bytes/entry X 64 entries / (1024 bytes/cluster).
  - The partition contains 64 Kclusters (64 Mbytes / 1024 bytes/cluster). Thus the data region of the file system contains  $64 \times 1024 - (1+256+256+2) = 65021$  clusters. The clusters are numbered 2 to 65022 (numbers are represented as 16 bit integers).
  - The first four bytes of the FAT are reserved (used to identify the type of FAT); which means that indexes 0 and 1 are not used in the FAT. The first FAT entry starts at index 2 (which is why the data clusters are numbered starting at 2). The valid values for FAT entries include:
    - 0 = the corresponding cluster is free for allocation
    - 2 to 65022 = the corresponding cluster has been allocated and the entry gives the number of the next allocated cluster.
    - 0xFFFF = the corresponding cluster has been allocated and is the last cluster.
    - 0xFFF7 = the corresponding cluster is defective.
    - Other values – not used.

DOS Name	DOS Ext	Type	Create Time	Last Access Date	Last Modified Time	First Cluster (Block)	Size
Directory:							
Directory:							
Directory:							
Directory:							
Directory:							
Directory:							
Directory:							
Directory:							



**File Allocation Table (FAT)**

	0	1	2	3	4	5	6	7	8	9
0	xx	xx								
10										
20										
30										
40										
50										
60										
70										
80										
90										

xx - reserved