CSI3131 – Operating Systems
Tutorial 9
File Systems – Solution

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.	**201**	**1**	**1**
The block is added in the middle.	**101**	**52**	**1**
The block is added at the end.	**1**	**3**	**1**
The block is removed from the beginning.	**198**	**1**	**0**
The block is removed from the middle.	**98**	**52**	**0**
The block is removed from the end.	**0**	**100**	**0**

2. Why must the bitmap for file allocation be kept on mass storage, rather than in main memory?

   In case of system crash (memory failure) the free-space list would not be lost as it would be if the bit map had been stored in main memory.

3. Explain how the VFS layer allows an operating system easily to support multiple types of file systems.

   VFS introduces a layer of indirection in the file system implementation. It borrows techniques from object-oriented programming. System calls can be made generically (independent of file system type). Each file system type provides its function calls and data structures to the VFS layer. A system call is translated into the proper specific functions for the target file system at the VFS layer. The calling program has no file-system-specific code, and the upper levels of the system call structures likewise are file system-independent. Thus VFS hides the details of file system implementation and offers a common view to all user programs/processes for all types of file systems.

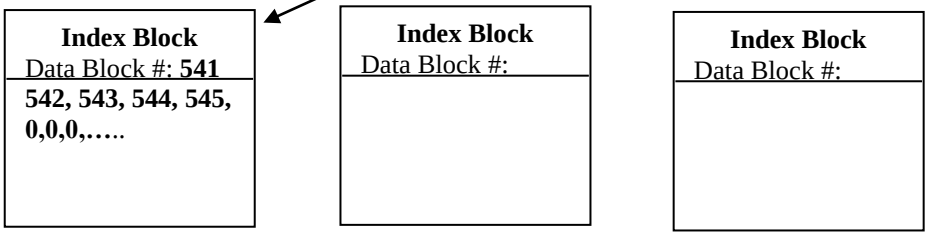
4. Examine the listing on page 5 that provides the contents of a MINIX file system. The listing was generated using *ls* with the following arguments:
   -l : Long listing format, displays many of the attributes found in the inode.
   -i: Provide the inode number
   -a: List all entries in directories (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*).

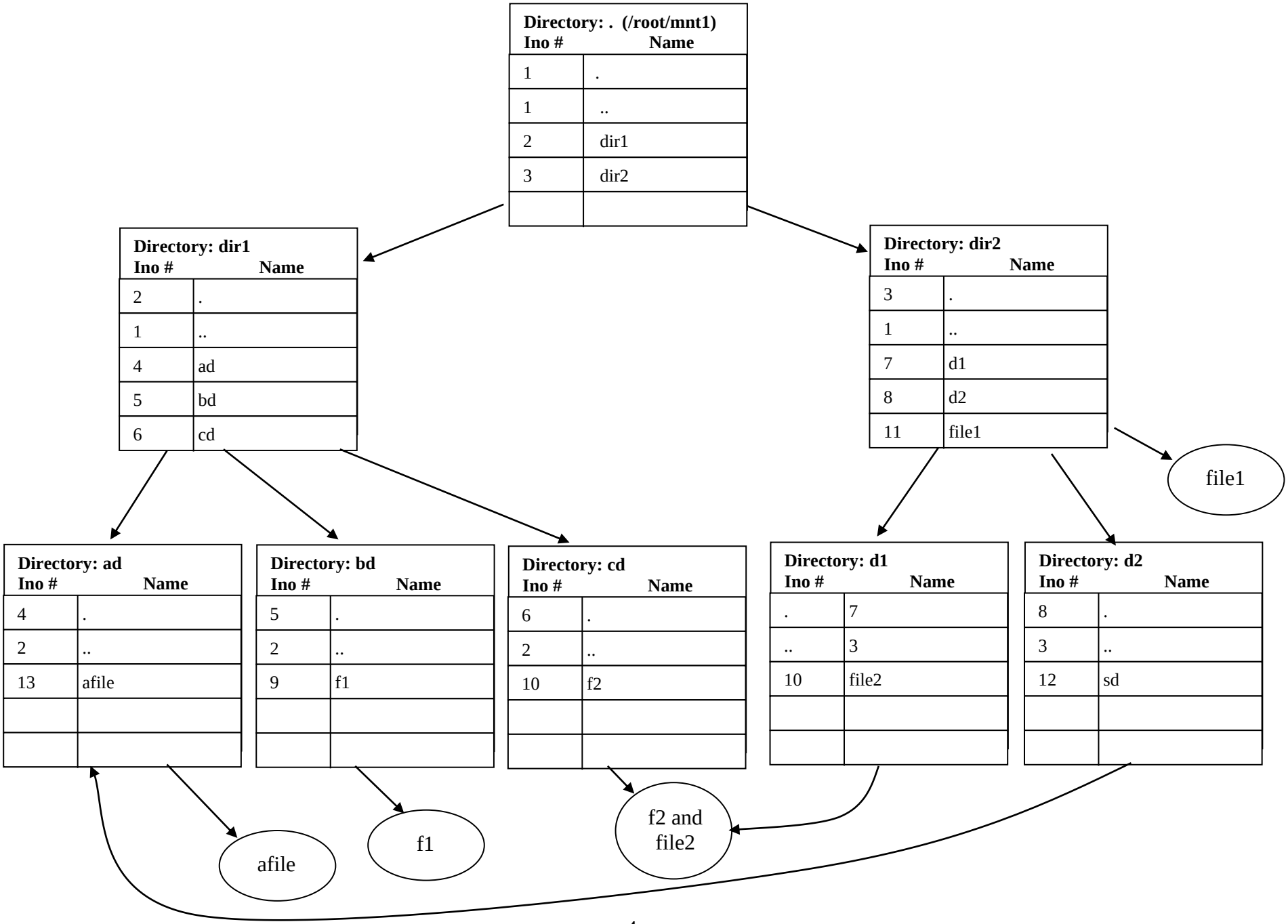
   (a) Given the following information and the information in the listing, complete the information found in the inode table shown on page 3. Be sure to fill in the index block if an indirect index is used in the inode.
   - The numerical user id for *root* is 0 and for *test1* is 500. Use these values 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 referes 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 field 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).

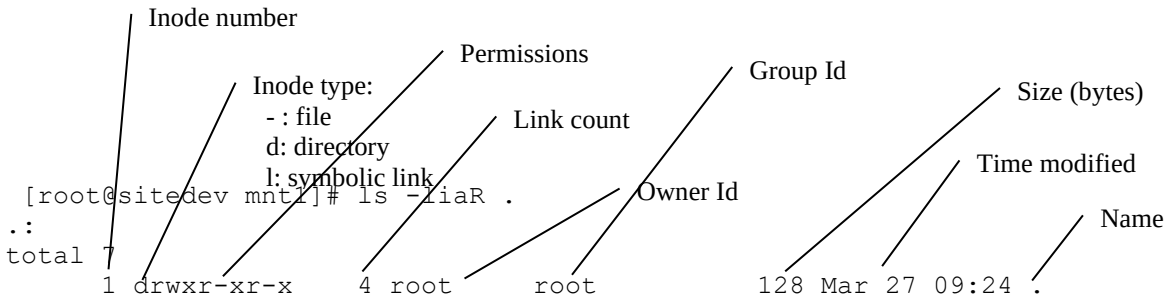
   (b) On page 4, complete the contents of the directory tables and include arrows between tables to show the tree structure. Add files as circles or ellipses and include them in the directory tree.

MINIX File System.

inode #	i_mode	i_uid	i_size	i_time	i_gid	i_nlinks	i_zone[0..8]
1	0100 000 111101101	0	128	Mar 27 09:24:26 2008	0	4	524,0,0,0,0, 0,0,0,0
2	0100 000 111101101	500	160	Mar 27 09:24:34 2008	244	5	525,0,0,0,0, 0,0,0,0
3	0100 000 111101101	500	160	Mar 27 09:30:34 2008	244	4	526,0,0,0,0, 0,0,0,0
4	0100 000 111101101	500	96	Mar 27 09:36:44 2008	244	2	527,0,0,0,0, 0,0,0,0
5	0100 000 111101101	500	96	Mar 27 09:29:27 2008	244	2	528,0,0,0,0, 0,0,0,0
6	0100 000 111101101	500	96	Mar 27 09:30:08 2008	244	2	529,0,0,0,0, 0,0,0,0
7	0100 000 111101101	500	96	Mar 27 09:31:52 2008	244	2	530,0,0,0,0, 0,0,0,0
8	0100 000 111101101	500	96	Mar 27 09:36:09 2008	244	2	531,0,0,0,0, 0,0,0,0
9	1000 000 110100100	500	1053	Mar 27 09:29:27 2008	244	1	532,533,0,0, 0, 0,0,0,0
10	1000 000 110100100	500	10530	Mar 27 09:30:08 2008	244	2	534,535,536, 537,538,538, 540,541,0
11	1000 000 110100100	500	2106	Mar 27 09:30:42 2008	244	1	546,547,548, 0,0,0,0,0,0
12	1010 000 111111111	500	13	Mar 27 09:36:09 2008	244	1	549,0,0,0,0, 0,0,0,0
13	1000 000 110100100	500	3159	Mar 27 09:36:56 2008	244	1	550,551,552, 553,0,0,0,0, 0
14	0	0	0	0	0	0	0




Directory: . (/root/mnt1)
Ino # Name
1 .
1 ..
2 dir1
3 dir2
Directory: dir1
Ino # Name
2 .
1 ..
4 ad
5 bd
6 cd
Directory: dir2
Ino # Name
3 .
1 ..
7 d1
8 d2
11 file1
file1
Directory: ad
Ino # Name
4 .
2 ..
13 afile
Directory: bd
Ino # Name
5 .
2 ..
9 f1
Directory: cd
Ino # Name
6 .
2 ..
10 f2
Directory: d1
Ino # Name
. 7
.. 3
10 file2
Directory: d2
Ino # Name
8 .
3 ..
12 sd
afile
f1
f2 and file2




```
 [root@sitedev mnt1]# ls -liaR .
.:
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 contain 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:
    - The 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 K clusters (64 Mbytes / 1024 bytes/cluster). Thus the data region of the file system contains 64 X 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: root							
dir1		**10**				**2**	**-**
dir2		**10**				**3**	**-**
Directory: dir1							
.		**10**				**2**	
..		**10**				**0**	
ad		**10**				**4**	
bd		**10**				**5**	
cd		**10**				**6**	
Directory: dir2							
.		**10**				**3**	
..		**10**				**0**	
d1		**10**				**7**	
d2		**10**				**8**	
file1		**00**				**9**	**2106**
Directory: ad							
.		**10**				**4**	
..		**10**				**2**	
afile		**00**				**12**	**3159**
Directory: bd							
.		**10**				**5**	
..		**10**				**2**	
f1		**00**				**16**	**1053**
Directory: cd							
.		**10**				**6**	
..		**10**				**2**	
Directory: d1							
.		**10**				**7**	
..		**10**				**3**	
file2		**00**				**18**	**10530**
Directory: d2							
.		**10**				**8**	
..		**10**				**3**	

File Allocation Table (FAT)

	0	1	2	3	4	5	6	7	8	9
0	xx	xx	**FFFF**	**FFFF**	**FFFF**	**FFFF**	**FFFF**	**FFFF**	**FFFF**	**10**
10	**12**	**FFFF**	**13**	**14**	**15**	**FFFF**	**17**	**FFFF**	**19**	**20**
20	**21**	**22**	**23**	**24**	**25**	**26**	**27**	**28**	**FFFF**	**0**
30	0	0	0	0	0	0	0	0	0	0
40	0	0	0	0	0	0	0	0	0	0
50	0	0	0	0	0	0	0	0	0	0
60	0	0	0	0	0	0	0	0	0	0
70	0	0	0	0	0	0	0	0	0	0
80	0	0	0	0	0	0	0	0	0	0
90	0	0	0	0	0	0	0	0	0	0

xx – reserved

6. Consider the following simplified file system
   - the disk has 32 blocks
   - each block contains 8 bytes/items
   - block 0 is the boot block
   - block 1 contains the root directory
   - each directory block contains 4 entries in format (name, block#). If the name begins with a lower case letter, it references normal file and the block# is the block containing the inode of that file. If the name begins with capital letter, it means subdirectory and the block # refers to the block of that directory. For convenience, names longer than 1 byte are used, despite the fact that they could not fit in the 8 byte directory block. Empty directory entries are omitted.
   - remaining blocks contain files/subdirectories or are empty
   - the first 5 entries in the inode contain information like time stamps, access rights, owner, which are not so important for us at this moment (marked as '_'); the next two entries are the direct block references, and the last one is the indirect block, -1 means unused.
   - the indirect block contains 8 entries

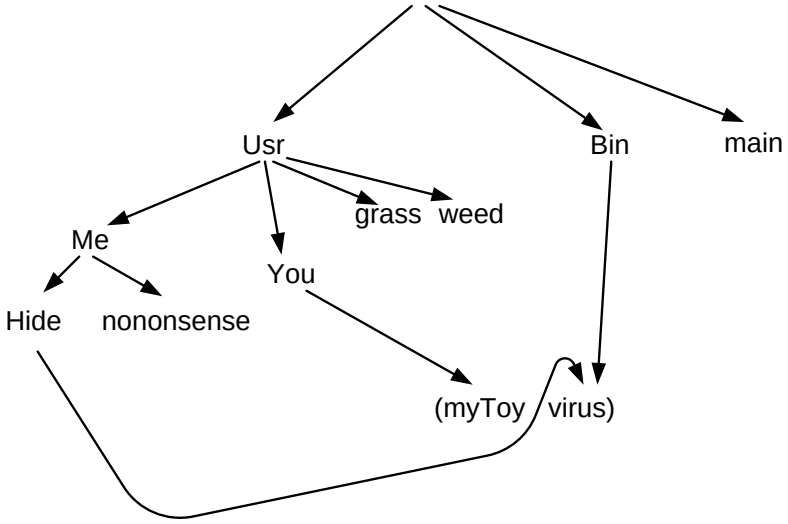
   a) Fill in the first column (What) of the table below, using one of the following descriptions (replace items enclosed in < > with significant values such as directory name, bloc number, file name, etc):

   "Directory <dirname>"
   "block #<number> of <filename>"
   "inode of <filename>"
   "index block of <filename>"
   "empty" (for unused/unreferenced blocks)

Assume the following content of the disc (the 8-character strings describe the content of the file (and empty) blocks, the content of the directory blocks is shown in a human-readable format):

What	Block #	Content
Boot block	0	<booting code>
Root directory	1	<Usr:6; Bin:18; main: 28>
Empty	2	Water_is
Index block of virus and myToy	3	<10,12,-1,-1,-1,-1,-1,-1>
Empty	4	bit_dirt
Inode of weed	5	<_, _, _, _, _, 31, 30, -1>
Directory Usr	6	<Me:14; You:24; grass:7; weed: 5>
Inode of grass	7	< _, _ , _, _, _, 8, -1,-1>
Block 0 of grass	8	Green!!!
Empty	9	maybe_a_
Block 2 of virus and myToy	10	belong_t
Empty	11	y,_but_wh
Block 3 of virus and myToy	12	o_us!!!!
Block 0 of nonsense	13	Little_n
Directory Me	14	<nonsense:25; Hide:18>
Empty	15	not?????
Block 1 of virus and myToy	16	base_are
Empty	17	o_cares!
Directory Bin and Hide	18	<virus: 20>
Empty	19	Maybe_yo
Inode of virus and myToy	20	<_,_,_,_,_, 21,16,3>
Block 0 of virus and myToy	21	All_your
Empty	22	u_should
Block 1 of nonsense	23	onsense.
Directory You	24	<myToy:20>
Inode of nonsense	25	<_,_,_,_,_,13,23,-1>
Block 0 of main	26	The_main
Block 1 of main	27	file?Oh!
Inode of main	28	<_,_,_,_,_, 26,27,-1>
Empty	29	or_maybe
Block 1 of weed	30	stuff!!!
Block 0 of weed	31	Seriuos_

b) Draw the directory tree structure of the disc



c) Which disc blocks, and at which order, will be read when executing the following (pseudo)code snippet:

```
fd = open("/Usr/Me/Hide/virus", "r");   // open that file for reading
seek(fd, 9);                            // position the file pointer to offset 9 within the file
read(fd, buf, 12);                      // read 12 bytes from the file
```

open call: 1, 6, 14, 18, 20 (reads inode from block 20)
seek call:
read call: 16, 3, 10

d) Assume that string "And yet more nonsense from him!" has been appended to the end of the file "/Usr/Me/nonsense". List all changed blocks and their new content.

What	**Block #**	**Content**
index block of nonsense	2	<4, 9, 11, 15>
Block #2 of nonsense	4	And_yet_
Block #3 of nonsense	9	more_non
Block #4 of nonsense	11	sense_fr
Block #5 of nonsense	15	om_him!

e) Represent the same(original, before c)) the situation using FAT (assume that FAT is stored at blocks 2,3,4,5).

What	**Block #**	**Content**
Boot block	0	<booting code>
Root directory	1	<Usr:6 ; Bin:18; main: 26>
FAT block #0	2	___,___,___,___,___,___, -1,___
FAT block #1	3	-1,___,12,___,-1, 23, -1,___
FAT block #2	4	10,___, -1,___,___,16,___, -1
FAT block #3	5	-1,___,27,-1,___,___, -1, 30
. . .	. . .	. . .
	6	<Me: 14; You: 24; grass: 8; weed: 31>
. . .	. . .	. . .
	14	<nonsense: 13; Hide:18>
. . .	. . .	. . .
	18	<virus:21>
. . .	. . .	. . .
	24	<myToy:21>
. . .	. . .	. . .