VauLSMorg

03 FS - File System & FUSE
Aneka Soal Ujian Sistem Operasi
Rahmat M. Samik-Ibrahim et.al.

1. **2016-1a**

   Circle or cross: "**T**" if True – "**F**" if False.

   T / F A file is logical storage unit (Silber9).

   T / F A volume (of file system) may be "a subset of a device", or "a whole device", or "multiple devices linked together into a disk array set" (Silber9).

   T / F Microsoft Windows' volume label "C:" is usually reserved for the main disk. Label "A:" and "B:" were once reserved for the floppy disks.

   T / F The implementation of File Systems on Virtual Machines is called Virtual File Systems (VFS (Silber9).

   T / F One disadvantage of linked allocation method (of disk space) is external fragmentation (Silber9).

   T / F A unified buffer cache can not solve the problem of double caching (Silber9).

2. **2017-1**

   Circle or cross: "T" if True – "F" if False.

   T / F There is no external fragmentation in a file system with linked allocation.

   T / F The Deadline I/O Scheduler (Linux) gives the **Read Queues** a higher priority.

   T / F In a distributed file system, it is possible to write unnoticed by others for a short time.

   T / F Doubling the block size in a indexed allocation disk space system will exactly double the maximum file size.

3. **2017-2**

   (Adapted from JJ Pfeiffer, "Writing a FUSE Filesystem: a Tutorial", NMSU, licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License)

   One of the real contributions of Unix (and later Linux) has been the view that "everything is a (01)". A tremendous number of radically different sorts of objects, have been mapped to the (02). One of the more recent directions this view has taken, has been (03) or also known as (04). A (05) is a program that listens on a (06) for file operations to perform, and performs them. With FUSE, (07) users can create their own file systems. The idea here is that users can write a FUSE file system to provide interaction with an object in terms of a (08) and (09). A user just has to write codes that implements file operations like (10), (11), and (12).

   Match the number of the sentence above with these following phrases:

[ ] directory structure	[ ] file	[ ] file abstraction	[ ] filesystem operations
[ ] Filesystems in User Space	[ ] FUSE	[ ] FUSE filesystem	[ ] non-privileged
[ ] open()	[ ] read()	[ ] socket	[ ] write()

4. **2018-1**

(1) or (2) is a (3) interface that lets (4) Unix-like users create their own (5) without modifying the kernel code. It is particularly useful for writing (6) filesystems – which don't actually store data themselves. It is available for a variety of systems like (7), (8), and (9).

Match the number of the sentence above with these following phrases:

[ ] Android [ ] filesystems [ ] Filesystem in Userspace [ ] FUSE [ ] Linux

[ ] MacOS [ ] non-privileged [ ] software [ ] virtual [ ] Windows-10

5. **2018-2 (47%)**

(01) Nonvolatile Memory (NVM) Devices is frequently used in a disk-drive-like container, in which called a (02). Until today (2018), (03) are cheaper per megabyte than (04). (05) scheduling is generally used in NVMs, because NVMs do not contain (06). (07) provides access to storage across a network, whereas (08) is a private network connecting servers and storage units. A (09) is a pointer to an entry in the per-process file-system table. A significant drawback of hash table is its (10). A log-based transaction-oriented file systems is also known as a (11) file systems. The design goal of the Apple File System (APFS) is to run on (12) Apple devices. The (13) is a temporary file system that is created in (14). These following information is required for mounting a file system: (a) The name of the (15) containing file system, (b) file system (16) and (c) the (17). The (18) layer provides mechanisms for uniquely representing files. When a user is mounting (19), his/her programs are able to access the data using the (20) file operation system calls.

Match the number of the sentence above with these following phrases:

[ ] all current (60%)	[ ] device (50%)	[ ] file handle (10%)	[ ] First Come First Served (60%)
[ ] fixed size (60%)	[ ] Flash-memory-based (40%)	[ ] FUSE (00%)	[ ] Hard disk drives (50%)
[ ] journaling (40%)	[ ] mount point (30%)	[ ] moving disk heads (40%)	[ ] Network-attached storage (90%)
[ ] solid-state disk (50%)	[ ] solid-state disk (60%)	[ ] standard (30%)	[ ] storage-area network (90%)
[ ] tmpfs (60%)	[ ] type (80%)	[ ] virtual file system (10%)	[ ] volatile main memory(40%)

6. **2019-1 (45.8%)**

This problem assumes that you are familiar with the "Allocation Methods section" in Silberschatz's book. Consider a fictitious file system called "Little Brother File System" (**LBFS**). LBFS is using the "linked allocation method". LBFS consists of 12 blocks (block 0 to block 11). Each block size is 1kByte (1024 bytes). Inside each block is a "1-byte pointer" that points to the next block number. The first block of a file pointer number can be reached from the "file directory". The "file directory" is in block 0. Inside the "file directory" is a special file called "FREE BLOCK LIST" which is a linked list to all available free blocks. Pointer number "**255**" in block 11 means "end of the linked list".

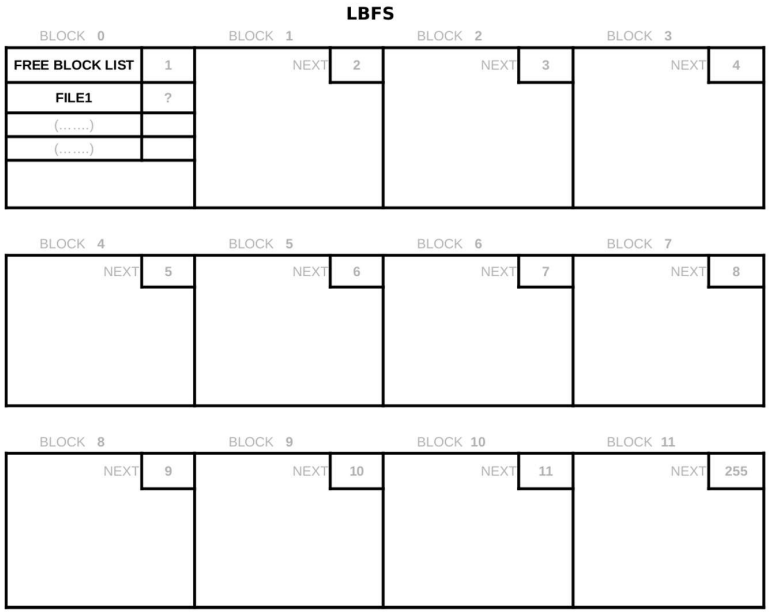
Initially, "FREE BLOCK LIST" holds blocks: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and 11.

(a) (46%) Create a file "`FILE1`" of size 2kBytes (2048 bytes). The blocks for "FILE1" are taken from the "FREE BLOCK LIST". How many blocks are allocated for "FILE1" (excluding the directory block)?

(b) (47%) Specify the linked block list of "FILE1". Starting from the directory block, "FILE1" will **hold blocks**:

(c) (47%) Specify the remaining linked block list of file "FREE BLOCK LIST". Starting from the directory block, "FREE BLOCK LIST" will **hold blocks**:

You **may or may not** use these following diagram as a worksheet. It will **not** affect your grade!

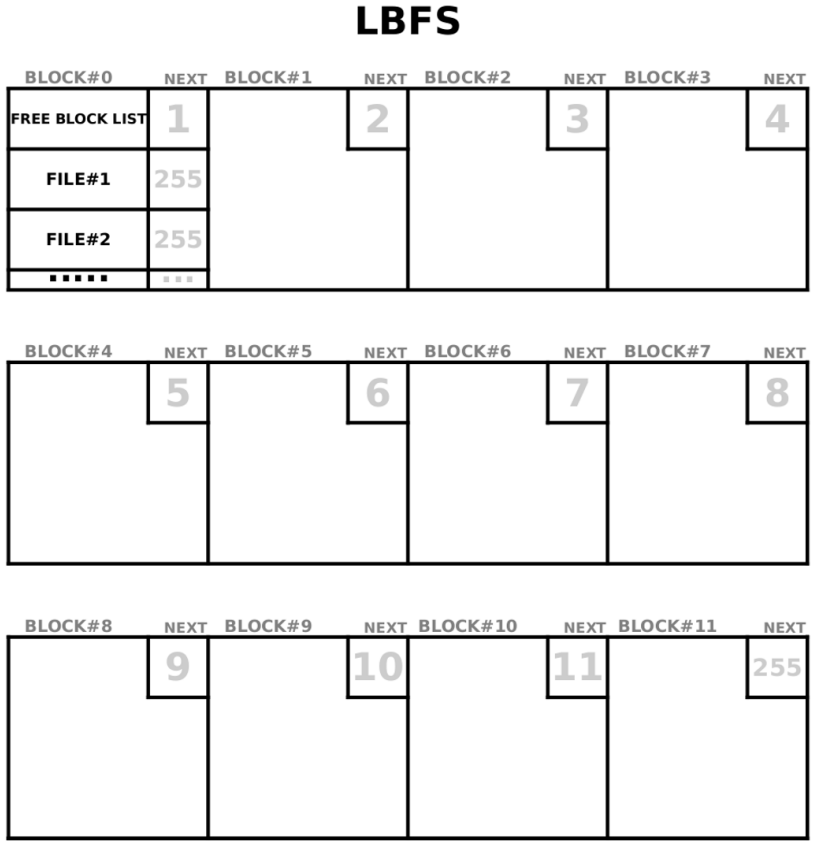


7. **2019-2 (44%)**

This problem assumes that you are familiar with the last semester exam problem as well as "Allocation Methods section" in Silberschatz's book. Consider a fictitious file system called "Little Brother File System" (**LBFS**). LBFS is using the "linked allocation method". LBFS consists of 12 blocks (Block#0 to Block#11). Each block size is 1kByte (1024 bytes). The pointer size is 1 byte. Pointer number "**255**" means "end of the linked list".

Block#0 is the "Directory Block" (DB). The first DB entry is a special file called "FREE BLOCK LIST" which is a linked list to all available free blocks.

Initially, "FREE BLOCK LIST" holds blocks in sequence: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and 11.



To create a new file:

- alocate "free file block(s)" from the "Free Block (linked) List".
- terminate the "free file block(s)" with pointer 255.
- place the "new file name" and the "linked-list header" in a DB entry.
- file name examples are `File#1, File#2`, etc.

To delete a file:

- return the "file block(s)" to the **end** of "Free Block (linked) List".
- put value 255 to the "linked-list header" of the file DB entry.

(a) Create file "`FILE#1`" of size 2 kBytes (2048 bytes). The blocks for "`FILE#1`" are taken from the "FREE BLOCK LIST". How many blocks are allocated for "`FILE#1`" (excluding the directory block) (50%)?

(b) Specify the linked block list of "`FILE#1`" starting from the directory block (46%).

(c) Create another file (`FILE#2`) of size 1 kBytes (1024 bytes). How many blocks are allocated for "`FILE#2`" (excluding the directory block) (37%)?

(d) Specify the linked block list of "`FILE#2`" starting from the directory block (30%).

(e) Delete "`FILE#1`". Specify the linked block list of file "FREE BLOCK LIST" starting from the directory block (13%).

(f) Adjust the LBFS diagram (above) (42%).

8. **2020-1**

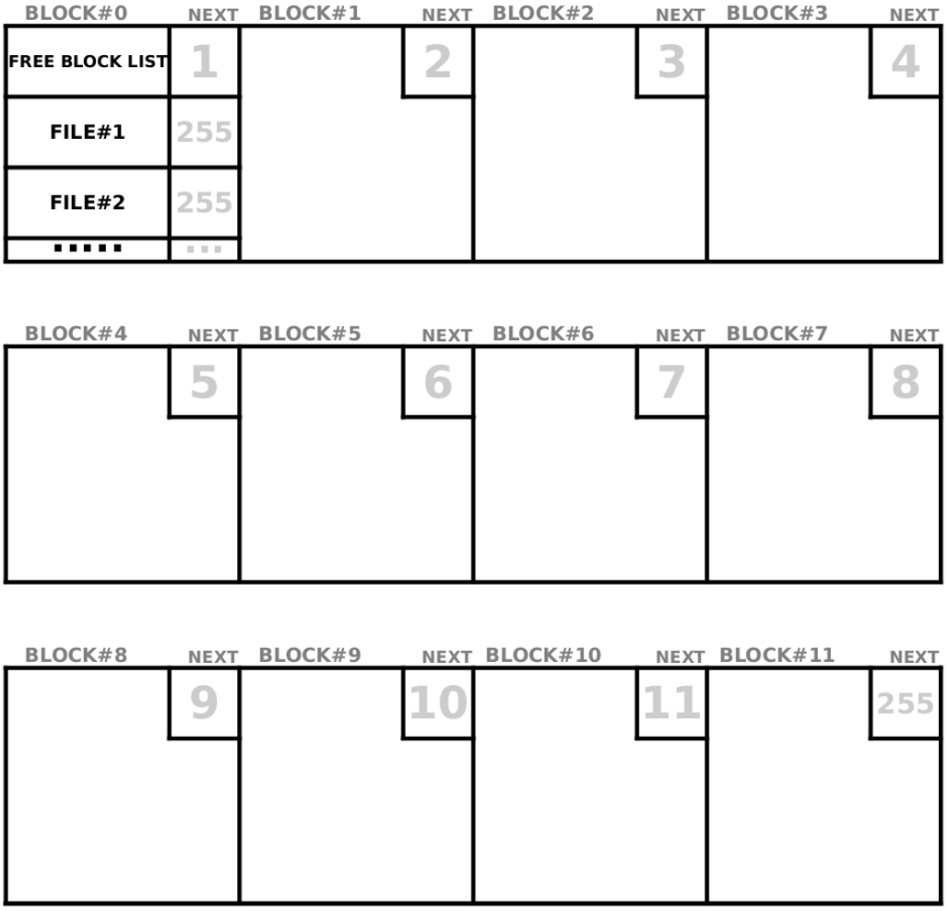
This problem assumes that you are familiar with the last semester exam problem as well as "Allocation Methods section" in Silberschatz's book. Consider a fictitious file system called "Little Brother File System" (**LBFS**). LBFS is using the "linked allocation method". LBFS consists of 12 blocks (Block#0 to Block#11). Each block size is 1kByte (1024 bytes). The Pointer size is 8 bits and pointer number "**255**" means "end of the linked list".

Block#0 is the "Directory Block" (DB). The first DB entry is a special file called "FREE BLOCK LIST" which is a linked list to all available free blocks.

Consider two functions of LBFS:

- **myopen** ("FileName", "BlockAllocation")
  - **IF** "FileName" exists, just open it. Ignore "BlockAllocation".
  - **IF** "FileName" does NOT exists and "FreeBlocks" $\geq$ "BlockAllocation":
    - allocate "BlockAllocation" from the "Free Block (linked) List".
    - place the "FileName" and the "BlockAllocation" header in a directory entry. "FileName" examples are "File#1", "File#2", etc.
    - terminate the "FileName" linked list with 255.
    - adjust the "Free Block (linked) List" header with the reminder (unused) blocks.
  - **ELSE** do nothing.
- **mydelete** ("FileName")
  - **IF** "FileName" exists:
    - **RETURN** the "block(s)" of "FileName" to the end of "Free Block (linked) List".
    - Put value 255 to the "Linked-List Header" of "FileName" directory entry.

LBFS



Initially,

- "FREE BLOCK LIST" holds blocks in sequence: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}.
- "FILE#1" does not exists or: {}.
- "FILE#2" does not exists or: {}.

Use the curly braces format "{}" like the initial example above. What will be the "FREE BLOCK LIST", "FILE#1", "FILE#2" after the following sequences.

(a) myopen ("FILE#1", 9)
- "FREE BLOCK LIST":
- "FILE#1":
- "FILE#2":

(b) myopen ("FILE#2", 1)
- "FREE BLOCK LIST":
- "FILE#1":
- "FILE#2":

(c) mydelete ("FILE#1")
- "FREE BLOCK LIST":
- "FILE#1":
- "FILE#2":

(d) mydelete ("FILE#2")
- "FREE BLOCK LIST":

- "FILE#1":
- "FILE#2":

(e) myopen ("FILE#1", 5)

- "FREE BLOCK LIST":
- "FILE#1":
- "FILE#2":

(f) Adjust the LBFS diagram (above). Please re-write **ALL** pointers!