



Computer Engineering II

Exercise Sheet Chapter 6

Basic

1 HDDs

- Which tracks are favored if SSTF is used for disk scheduling, assuming a random access workload?
- Explain how starvation can occur for SSTF, SCAN, C-SCAN, SPTF; how does F-SCAN solve the problem?
- If we always serve disk requests in the order they occur, we avoid starvation. Which disadvantages does this have?
- Assume you have two HDDs, one with 5ms average seek and 9000 rounds per minute, and one with 3ms average seek and 5400 rounds per minute. Both disks have a peak transfer rate of 120 MB/s. Which offers a better rate of I/O for a random access workload, assuming a 4KB sector size? How long is the total I/O time for a 200MB random access workload? Assume 1MB = 1000KB.

2 SSDs

| Block | 0 | | | | 1 | | | |
|---------|---|---|---|---|---|---|---|---|
| Page | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Content | | | | | | | | |
| State | i | i | i | i | i | i | i | i |

Perform the following writes and erases for an SSD, once for a direct mapped SSD, once for a page-level mapped one. Assume that a block consists of 4 pages, and each write is exactly the size of one page. By “garbage collect” we mean “pick a block to erase, issue necessary writes, update validity information”.

- write(0) content A
- write(6) content B
- write(4) content C
- write(1) content D
- write(0) content E
- garbage collect (only page-level mapped)
- write(4) content F

- garbage collect (only page-level mapped)

Below you'll find templates for the exercise. Show the SSD after each set of writes as well as after each block erasure.

Direct mapped SSD:

write(0) content A

| | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|
| Content | | | | | | | | |
| State | | | | | | | | |

write(6) content B

| | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|
| Content | | | | | | | | |
| State | | | | | | | | |

write(4) content C

| | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|
| Content | | | | | | | | |
| State | | | | | | | | |

write(1) content D

| | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|
| Content | | | | | | | | |
| State | | | | | | | | |

write(0) content E

| | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|
| Content | | | | | | | | |
| State | | | | | | | | |

| | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|
| Content | | | | | | | | |
| State | | | | | | | | |

write(4) content F

| | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|
| Content | | | | | | | | |
| State | | | | | | | | |

| | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|
| Content | | | | | | | | |
| State | | | | | | | | |

Page-level mapped SSD:

write(0) content A

| | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|
| Table | | | | | | | | |
| Content | | | | | | | | |
| State | | | | | | | | |

write(6) content B

| | | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|--|
| Table | | | | | | | | | |
| Content | | | | | | | | | |
| State | | | | | | | | | |

write(4) content C

| | | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|--|
| Table | | | | | | | | | |
| Content | | | | | | | | | |
| State | | | | | | | | | |

write(1) content D

| | | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|--|
| Table | | | | | | | | | |
| Content | | | | | | | | | |
| State | | | | | | | | | |

write(0) content E

| | | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|--|
| Table | | | | | | | | | |
| Content | | | | | | | | | |
| State | | | | | | | | | |

garbage collection

| | | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|--|
| Table | | | | | | | | | |
| Content | | | | | | | | | |
| State | | | | | | | | | |

| | | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|--|
| Table | | | | | | | | | |
| Content | | | | | | | | | |
| State | | | | | | | | | |

write(4) content F

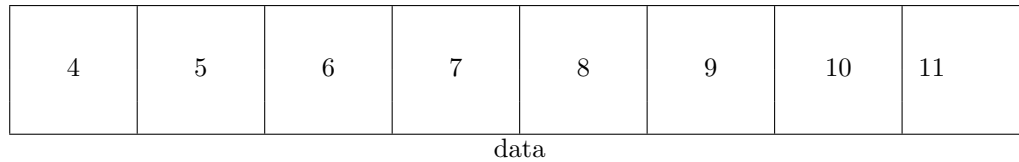
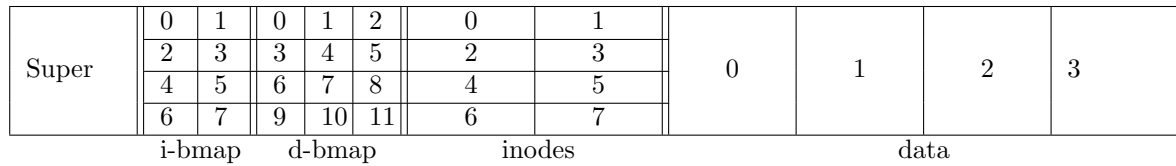
| | | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|--|
| Table | | | | | | | | | |
| Content | | | | | | | | | |
| State | | | | | | | | | |

garbage collection

| | | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|--|
| Table | | | | | | | | | |
| Content | | | | | | | | | |
| State | | | | | | | | | |

| | | | | | | | | | |
|---------|--|--|--|--|--|--|--|--|--|
| Table | | | | | | | | | |
| Content | | | | | | | | | |
| State | | | | | | | | | |

3 File System



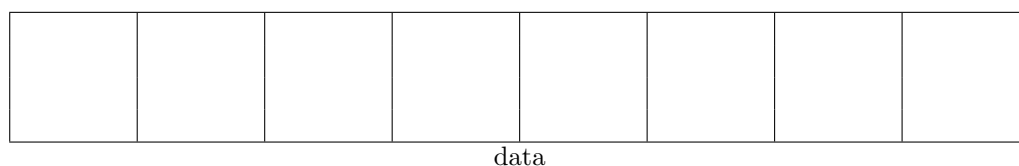
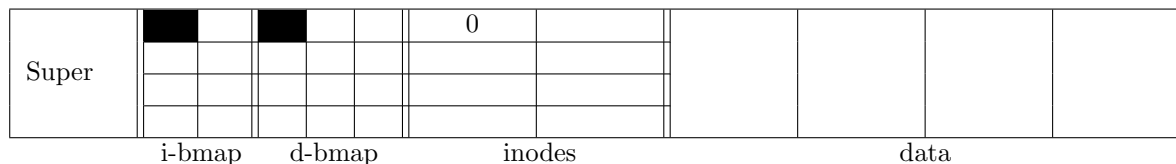
Assume we have a file system as depicted above with an inode table consisting of two blocks that can hold a total of 8 inodes, a data region that can hold 12 blocks, and uses bitmaps as allocation structures.

- a) Starting with only the root directory /, the following actions are performed:
- Create directory /a
 - Create directory /b
 - Create file /b/a.txt, containing "This is a long text file". Assume that this file needs two blocks.
 - Create file /a/b.txt, containing "This is another long text file". Assume that this file needs two blocks.
 - Delete file /b/a.txt
 - Create file /a/a.txt, containing "Hi". Assume that this file needs one block.

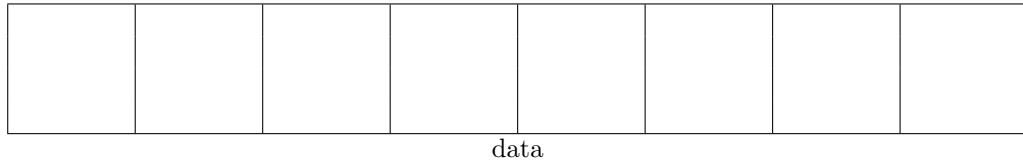
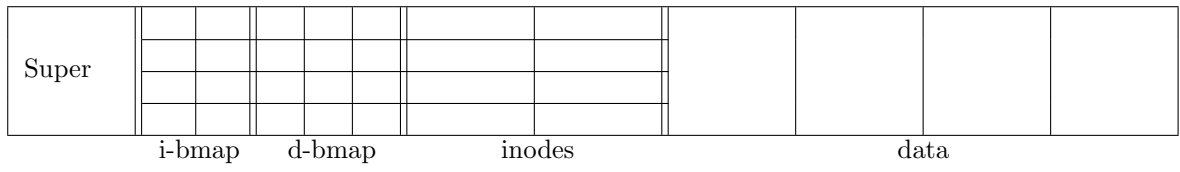
List the commands you can use in a UNIX terminal to perform these actions and show the state of the file system after each of these. For the inodes, list the indices of the referenced data blocks; for the directory blocks, list names and inode indices of the contained files. You find templates below.

- b) How does the file system resolve the address /a/b.txt?
 c) What is the difference between soft and hard links from an implementation view?

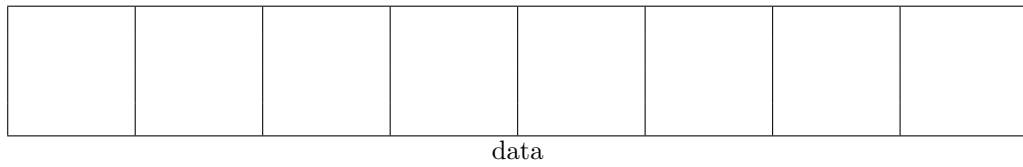
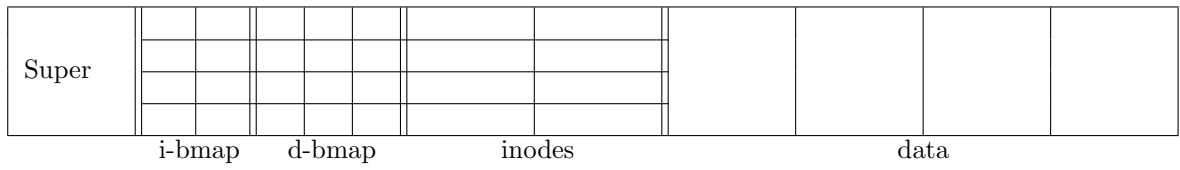
Starting state



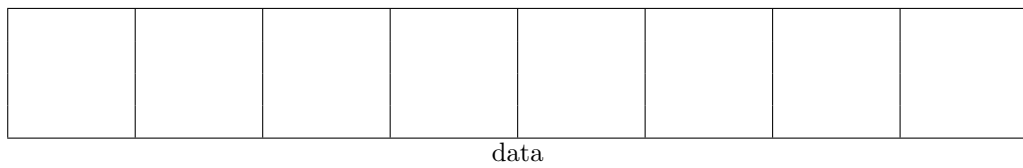
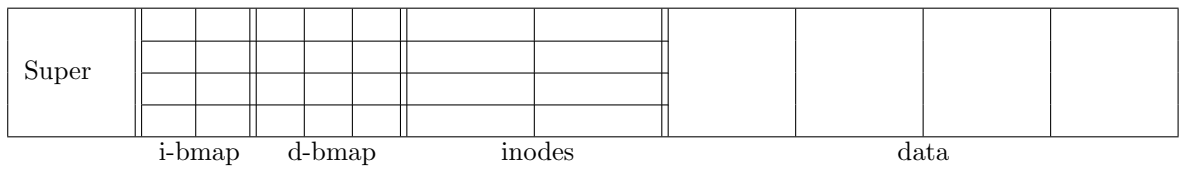
Command:



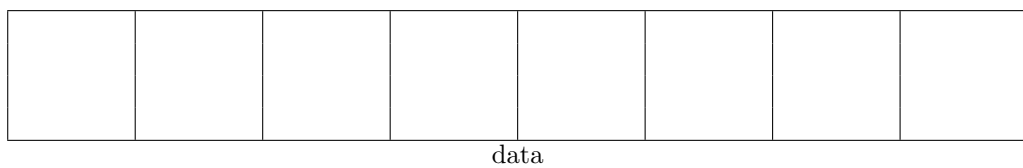
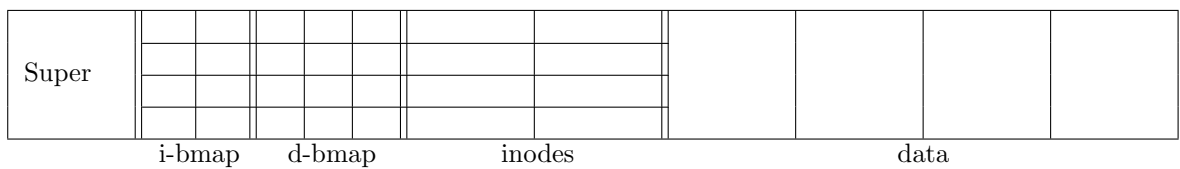
Command:



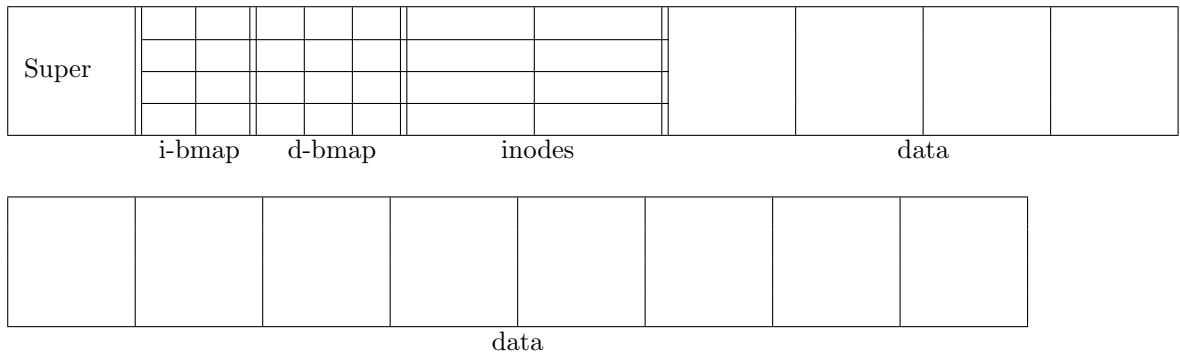
Command:



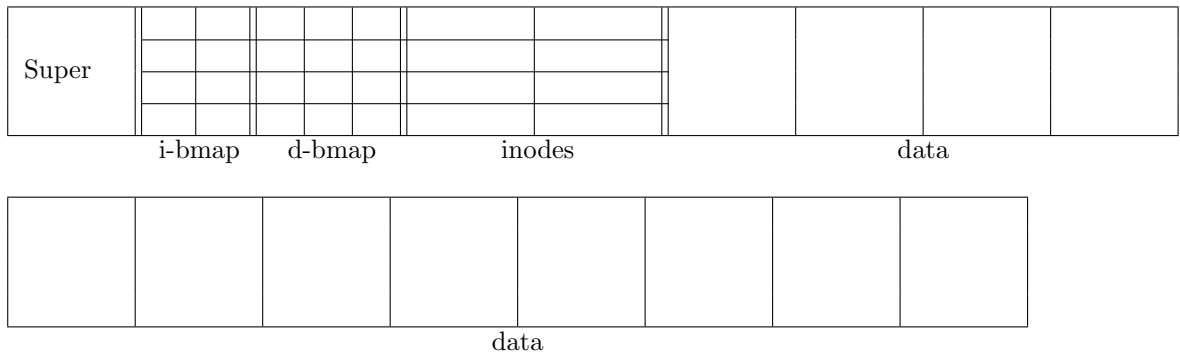
Command:



Command:



Command:



4 Permissions

Consider the following directory tree:

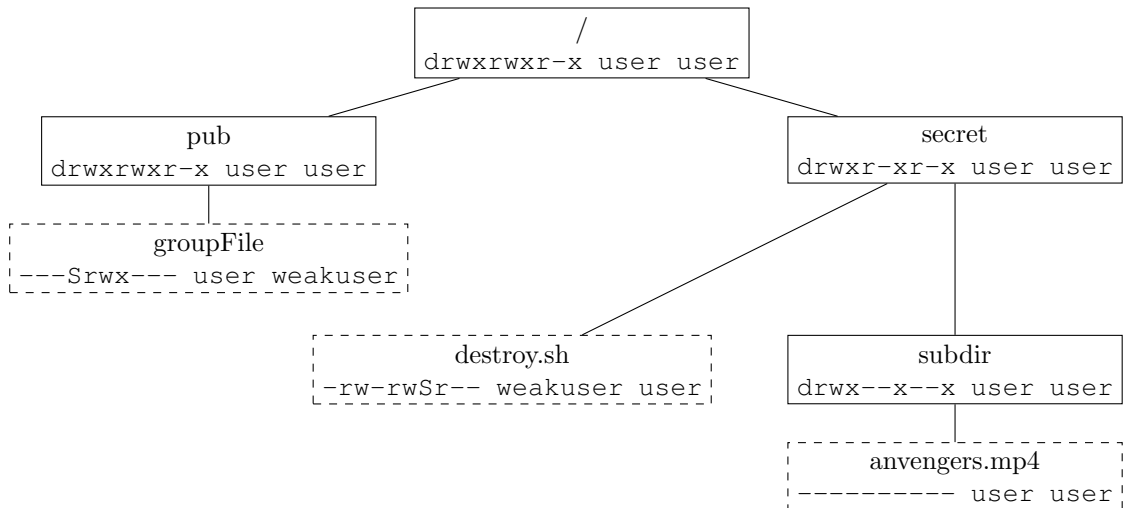


Figure 1: A small directory tree with ownership and permissions. The top line is the name of the file, the bottom line is the *permissions string*, then owner, then group owner.

Each letter in a permission string is also called a “bit” since it is implemented as a single bit (1 for “permission is granted”, 0 for “not granted”). To explain the permissions strings:

- The first letter tells us which type of file it is; `-` means “normal file”, `d` means “directory”. See https://en.wikipedia.org/wiki/Unix_file_types for the other available UNIX file types.
- The three letters after that (the *first triad*) specify the permissions of the owner of the file. The first letter of the triad is “read”, the second “write”, the third “execute” permissions. For example, `-r-x-----` means the owner has read and execute permissions, but not write.
- The second and third triad specify permissions for *group owner* and *other*.
- An execute bit of `s` means the `suid/sgid` bit and the respective execute bit are set, while `S` means only the `suid/sgid` bit but not the respective execute bit is set.

Assume for the following tasks that `user` is only in the group `user`, and `weakuser` is only in the group `weakuser`.

- a) `weakuser` wants to execute `/secret/destroy.sh`, but the OS says `Permission denied`. How would the permissions string have to be different so he can execute it? What about `user`?
- b) Can `weakuser` read, write, or execute `/pub/groupFile`?
- c) The command `ls -l /secret/subdir` should list the contents of that directory. Why doesn't it when `weakuser` runs that command?
- d) We want to make it possible for `weakuser` to play `/secret/subdir/avengers.mp4`. What are the minimal required changes to the permission strings such that this is possible? Who can set the respective bits in the permission strings?
- e) Try to rebuild a directory structure like this on a UNIX machine. To change permissions, use the command `chmod` with the appropriate parameters; to see the contents of a directory with permission strings and owner/group owner, use `ls -l` or `ll`.