
Norwegian University of Science and Technology

# Operating Systems

## Example solutions for Theoretical Exercise 7

Michael Engel

# 7.1 Inode-based file systems

A Unix filesystem has 2 kB (1 kB = 1024 bytes) blocks and 4 byte disk addresses. Each inode contains 10 direct entries, one single-indirect entry and one double-indirect entry.

a. Calculate the maximum possible file size in this file system

10*2 kB + (2048/4)*2 kB + (2048/4) * (2048/4) *2 kB
= 20 kB + 1024 kB + 524288 kB = 525332 kB (or 537939968 bytes or 513.02 MB)

b. Suppose half of all files are exactly 1.5 kB in size and the other half of all files are exactly 2 kB. What fraction of the disk space would be wasted? (Consider only the blocks used to store data)

Both 1.5 kB and 2 kB files will use 2 kB of space. For each 2 kB file, 0 kB is wasted; for each 1.5 kB file, 0.5 kB is wasted.
Therefore, the fraction wasted is (0/2)*50%+(0.5/2)*50% = 12.5%.

c. Based on the same condition as in the previous item above, does it help to reduce the fraction of wasted disk space if we change the block size to 1 kB? Explain your answer

No. Nothing is changed. Both 1.5 kB and 2 kB files will still use 2 kB of space. For each 2 kB file, 0 kB is wasted; for each 1.5 kB file, 0.5 kB is wasted.

Therefore, the fraction wasted is (0/2)*50%+(0.5/2)*50% = 12.5%, which is unchanged.

# 7.2 Simple file system operations

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 at the beginning but there is room to grow at the end. Also assume that the block information to be added is stored in memory.

# 7.2 Simple file system operations

Calculate how many disk I/O operations are required if the block is…

| | Contiguous | Linked | Indexed |
|---|---|---|---|
| added at the beginning | 201 | 1 | 1 |
| added in the middle | 101 | 52 | 1 |
| added at the end | 1 | 3 | 1 |
| removed from the beginning | 198 | 1 | 0 |
| removed from the middle | 98 | 52 | 0 |
| removed from the end | 0 | 100 | 0 |

# 7.3 Free storage management

Why must the bit map for file allocation be kept on mass storage, rather than in main memory?

As with all metadata (see also TE6): 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.

# 7.4 File and directory names in Unix

a. Explain why the slash character (/) is not a valid character in Unix file names.

It is used as path separator. Allowing it inside of file names would allow for ambiguities, e.g. if there is a file foo in directory bar and also a file named "bar/foo" in the same directory

b. What happens when you try to create a file with a slash in the name, e.g. using `creat("my/file", 0777);`?

Looking at the macOS man page, we find:

*"The creat() function is the same as:*

*open(path, O_CREAT | O_TRUNC | O_WRONLY, mode);"*

So creat performs an inode lookup for the passed name "my/file" and interprets is as pathname indicating a file "file" in a directory "my".

If you happen to have a directory "my" in the current directory, creat will create a file "file" inside of "my". Otherwise you will get an error indicating "No such file or directory", which tells you that the directory "my" could not be found.

# 7.4 File and directory names in Unix

c. Give the meaning of the parameter `0777` in the call to `creat` above

This is the mode the created file should be given (0 = octal number follows, 777 = rwx for owner, group and others). However, this mode is xored with a possible *file creation mask,* which can be set and displayed using `umask`. With a umask value of 022, the permissions would be 0777 xor 022 = 0755 (binary: 111 111 111 xor 000 010 010).

d. Are there other characters that are not allowed in Unix file names?

The NULL character is also problematic, since it is hard to generate (since \0 indicates the end of a C string). All other characters except / and \0 are allowed. This sometimes leads to situations in which it can be difficult to access a file with a name that contains e.g. control characters.