1.
The problem is, if the new file is created in the same storage area or with the same absolute path name, Old files will be overwritten. Then the user can use the original link to access the new file, that is Wrong operation. a) Making sure all related to the file links are deleted when a file is deleted. b) and making file can be deleted when no link points to the file can avoid these problems.

2.
Only one table is maintained that contains references to all the files that the user is currently accessing, and only an open count is stored to indicate how many processes or users have opened the file. This file could be removed or closed when all processes or users perform this file operation, which means the open count is zero.
Yes, because the OS should keep track of each process-independent information.

3.
Contiguous:
Sequential: it would be efficient, Contiguous allocation great for sequential because Only need to find the first address, and then one by one continuous access.
Random file access: it would be efficient, because you can easily lactate to the first address of one process, add a logical address then you can get the physical address.

Linked:
Sequential: it would be efficient, Linked allocation good for sequential access, because you can simply follow the list pointer from one block to next block.
Random file access: it would not be efficient, because Access to the ith block requires i block reads.

Indexed:
Sequential: it would be efficient, simply access each block by following each index pointer in “index block”.
Random file access: it would be efficient, simply access the block by following each index pointer in “index block”.

4. idk
5.
It would decrease the internal fragmentation and memory would be best utilized. 5 kb file could be allocated a 4 kb block and 2 contiguous 512-byte blocks. Maintaining a bitmap of free blocks and maintaining extra state to allocate subblocks and coalesce the subblocks to obtain the large block when all subblocks are free.

6.
16kb x 1024 = 16384 bytes
16384 / 8 = 2048
12 + 2048 + 2048^2 + 2048^3 * 8kb = 64TB