Tutorial-9 Solution

File Structure

1. What problems could occur if a system allowed a file system to be mounted simultaneously at more than one location?
   Answer: There would be multiple paths to the same file, which could confuse users or encourage mistakes (deleting a file with one path deletes the file in all the other paths).
2. Consider a system that supports the strategies of contiguous, linked, and indexed allocation. What criteria should be used in deciding which strategy is best utilized for a particular file?
   Answer:
   Contiguous—if file is usually accessed sequentially, if file is relatively small.
   Linked—if file is large and usually accessed sequentially.
   Indexed—if file is large and usually accessed randomly
3. Why is it advantageous for the user for an operating system to dynamically allocate its internal tables? What are the penalties to the operating system for doing so?
   Answer: Dynamic tables allow more flexibility in system use growth — tables are never exceeded, avoiding artificial use limits. Unfortunately, kernel structures and code are more complicated, so there is more potential for bugs. The use of one resource can take away more system resources (by growing to accommodate the requests) than with static tables.
4. Explain how the VFS layer allows an operating system to support multiple types of file systems easily.
   Answer: VFS introduces a layer of indirection in the file system implementation. In many ways, it is similar to object-oriented programming techniques. 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. The translation at the VFS layer turns these generic calls into file-system-specific operations.
5. Consider a file currently consisting of 100 blocks. Assume that the filecontrol 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.
   a. The block is added at the beginning.
   b. The block is added in the middle.
   c. The block is added at the end.

d. The block is removed from the beginning.
e. The block is removed from the middle.
f. The block is removed from the end.

Answer:
The results are:

	Contiguous	Linked	Indexed
a.	201	1	1
b.	101	52	1
c.	1	3	1
d.	198	1	0
e.	98	52	0
f.	0	100	0

6. How do caches help improve performance? Why do systems not use more or larger caches if they are so useful?
   Answer: Caches allow components of differing speeds to communicate more efficiently by storing data from the slower device, temporarily, in a faster device (the cache). Caches are, almost by definition, more expensive than the device they are caching for, so increasing the number or size of caches would increase system cost.
7. One problem with contiguous allocation is that the user must reallocate enough space for each file. If the file grows to be larger than the space allocated for it, special actions must be taken. One solution to this problem is to define a file structure consisting of an initial contiguous area (of a specified size). If this area is filled, the operating system automatically defines an overflow area that is linked to the initial contiguous area. If the overflow area is filled, another overflow area is allocated. Compare this implementation of a file with the standard contiguous and linked implementations.
   Answer: This method requires more overhead then the standard contiguous allocation. It requires less overhead than the standard linked allocation.
8. Consider a file currently consisting of 150 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 block information to be added is stored in memory.
   Solution:

Assumptions:

- Each I/O operation reads or writes a whole block.
- For linked allocation, a file allocation table (FAT) is not used, i.e., only the address of the starting block is in memory.
- The blocks are numbered 1 to 150 and the current positions of these blocks are also numbered 1 to 150.

- All preparation of a block (including putting in the data and any link value) is done in main memory and then the block is written to disk with one write operation.
- The file control block does not have to be written to disk after a change (this is typical where many operations are performed on a file).
- At most one index block is required per file and it does not have to be written to disk after a change.
- For linked allocation, assume that no I/O operations are necessary to add a freed block to the free list.

a)

The block is added in the middle:

Contiguous: Assume that in the middle means after block 75 and before block 76. We move the last 75 blocks down one position and then write in the new block.

$75(1r + 1w) + 1w = 75r + 76w = 151$ I/O operations

Linked: We cannot find block 75 without traversing the linked list stored in the first 74 data blocks. So, we first read through these 74 blocks. Then we read block 75, copy its link into the new block (in main memory), update block 75's link to point to the new block, write out block 75, write new block.

74r + 1r + 1w + 1w = 75r + 2w = 77 I/O operations
block75 new

Indexed: Update the index in main memory. Write the new block.

1w = 1 I/O operation

b)

The block is removed from the beginning.

Contiguous: Simply change the starting address to 2.

0 I/O operations

Linked: Read in block 1 and change the starting address to the link stored in this block.

1r = 1 I/O operation

Indexed: Simply remove the block's address from the linked list in the index block.

0 I/O operations

9. Consider a file system on a disk that has both logical and physical block sizes of 512 bytes. Assume that the information about each file is already in memory. For the contiguous strategy, answer these questions:
    a) How is the logical-to-physical address mapping accomplished in this system? (For the indexed allocation, assume that a file is always less than 512 blocks long.)
    b) If we are currently at logical block 10 (the last block accessed was block 10) and want to access logical block 4, how many physical blocks must be read from the disk?

    Solution:

Let L be the logical address and let P be the physical address. 2. The assumption in part (a) is poorly given. It's more reasonable to simply assume that the index is small enough to fit into a single block. In fact, a 512 block file will probably require more than a single 512 byte block because block addresses typically require 3-4 bytes each.

(a) Overview The CPU generates a logical address L (a relative offset in a file) and the file system has to convert it to a physical address P (a disk address represented by a block number PB and an offset in this block). For convenience of calculation, we assume that blocks are numbered from 0. In any approach, we can determine the logical block number LB by dividing the logical address L by the logical block size (here 512). Similarly, the offset, which will be the same for logical and physical addresses since the block sizes are identical, is determined by applying modulus. The offset is the same in all approaches.

LB := L div 512
offset := L mod 512

Contiguous: Assume S is the starting address of the contiguous segment. Then a simple approach to mapping the address is:

P = S + L

If we prefer to consider the block level,

PB = SB + LB

(b) If we are currently at logical block 10 and we want to access logical block 4 ...

Contiguous: We simply move the disk head back by 6 blocks (from physical block 10 to physical block 4) because the space allocated to the file is contiguous. Then we read block 4, for a total of one read.