CS4411 Operating Systems Homework 3 Solutions
Spring 2019

1. **[10 points]** Suppose a hard disk has an average access time of 8ms, a rotational speed of 7200 rpm, and 63 sectors per disk.

   (a) Compute the average time to read a sector.

   (b) Compute the average time to read 10 sectors if they are contiguous (on the same track).

   (c) What is the average time if the 10 sectors are located randomly on the disk?

   Answer: The *average time to read a section* T_{sector} is

   $$T_{\text{sector}} = T_s + T_r + T_x$$

   where T_s is the average seek time, T_r is the average rotational delay, and T_x is the time to transfer one sector. The average seek time is given as 8 ms (*i.e.*, $T_s = 8$ ms). The rotational delay T_r is the time for the disk to spin one-half a rotation. The transfer time T_x is the time required to the disk to spin one sector, of 1/63 rotations. Using the given parameters, this is

   $$8\text{ms} + (0.5\text{ revolutions})\left(\frac{60000}{7200}\text{ms/rev.}\right) + \left(\frac{1}{63}\text{revolutions}\right)\left(\frac{60000}{7200}\text{ms/rev.}\right)$$

   To read 10 contiguous sectors would take

   $$8\text{ms} + (0.5\text{ revolutions})\left(\frac{60000}{7200}\text{ms/rev.}\right) + \left(\frac{10}{63}\text{revolutions}\right)\left(\frac{60000}{7200}\text{ms/rev.}\right)$$

   and to read 10 randomly located sectors would take

   $$10\left[8\text{ms} + (0.5\text{ revolutions})\left(\frac{60000}{7200}\text{ms/rev.}\right) + \left(\frac{1}{63}\text{revolutions}\right)\left(\frac{60000}{7200}\text{ms/rev.}\right)\right]$$

   ■

2. **[10 points]** Suppose that a disk drive has 5000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 2150, and the previous request was at cylinder 1805. The queue of pending requests, in FIFO order, is 2069, 1212, 2296, 2800, 544, 1618, 356, 1523, 4965, 3681. Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests, for each of the following disk scheduling algorithms:

   (a) First-Come-First-Serve (FCFS)

   (b) Shortest-Seek-Time-First (SSTF)

   (c) SCAN

   (d) C-SCAN

   (e) LOOK

   (f) C-LOOK

   Answer: The following table shows the access order of tracks and the total number of cylinders crossed.

Method	1	2	3	4	5	6	7	8	9	10	11	12	13	*Total*
FCFS	2150	2069	1212	2296	2800	544	1618	356	1523	4965	3681			13011
SSTF	2150	2069	2296	2800	3681	4965	1618	1523	1212	544	356			7586
SCAN	2150	2296	2800	3681	4965	**4999**	2069	1618	1523	1212	544	356		7492
C-SCAN	2150	2296	2800	3681	4965	**4999**	**0**	356	544	1212	1523	1618	2069	9917
LOOK	2150	2296	2800	3681	4965	2069	1618	1523	1212	544	356			7424
C-LOOK	2150	2296	2800	3681	4965	356	544	1212	1523	1618	2069			9137

In SCAN and C-SCAN the **4999** indicates that the disk head has to moved to the last track 4999. In the C-SCAN, the disk head only scan in one direction. As a result, after the desk heads visited the last track **4999** it has to be moved back to the first track **0** and scan in the same direction. ∎

3. **[10 points]** A file currently has 5 blocks, from block 0 to block 4. Assume that its FCB and directory are already in memory. There are a few more assumptions as shown blow:

   - In the contiguous allocation case, the directory entry has a start pointer and a length value.
   - In the linked allocation, a directory entry has a start pointer and an end pointer.
   - In the FAT allocation, the FAT table is in memory and a directory entry has a start pointer to the FAT table.
   - In the case of the indexed allocation, the index block is in memory and is large enough for further expansion.

   How many disk I/O operations (*i.e.*, reads and writes) are required for contiguous, linked, FAT, and indexed (single-level) allocation strategies, if block 2 to be removed. Assume that the directory and index will **not** be written back to disk after this insertion is done.

   Answer:

   - **Contiguous Allocation**: The contiguous allocation knows the location of the first block (*i.e.*, block 0), and can easily locate block 2. Because the allocation has to be contiguous, block 3 and block 4 have to be moved one block forward. In this case, for each of block 3 and block 4, one read and one write are needed, and four I/O operations are needed. If the directory has to be saved back to disk because the length has changed, one more I/O operation is needed.
   - **Linked Allocation**: To access block 2, the system has to **(1)** read in block 0 to obtain the address of block 1; **(2)** read block 1 to obtain the address of block 2; **(3)** read block 2 to obtain the address of block 3; and **(4)** replace the address field in block 1 with the address of block 3, and write block 1 back to disk. This is exactly the same as updating a link list. Therefore, we need 3 reads and 1 writes. no need to update directory for this case.
   - **FAT**: This only requires an update the pointer in FAT. One may have to write the portion of the FAT back to disk, and this requires one more disk operation.
   - **Index Allocation**: For the indexed allocation, because the index is already in memory, all block numbers starting with block 3 in the index block are moved forward. In this case, no I/O operation is needed. If the index block has to be written back to disk, one write is needed.

4. **[10 points]** Suppose a Unix inode has 10 direct disk block pointers, one single indirect block pointer, one double indirect block pointer, and one triple indirect block pointer. For convenience, each index table has $256 = 2^8$ entries and each block has 1K= 2^{10} bytes. What is the maximum size of a Unix file?

   Answer: Please refer to slide 21 of `10-Storage-System.pdf`. Let us examine each of the four cases as follows:

   - **Direct Disk Block Pointer**: There are 10 direct disk block pointers each of which points to a 2^{10} byte blocks. Therefore, this part supports a file of size 10×2^{10} bytes of $10 \times 1K = 10K$ bytes.
   - **Single Indirect Block Pointer**: Because each index table has $256 = 2^8$ entries and each block has 2^{10} bytes, the total size is $2^8 \times 2^{10} = 2^{18}$ bytes or 2^8K bytes.
   - **Double Indirect Block Pointer**: The first level has 2^8 entries of pointers, each of which points to an index table of 2^8 block pointers. Therefore, there are $2^8 \times 2^8 = 2^{64}$ blocks, and the file size of this part is $2^{64} \times 2^{10}$ bytes or $2^{64}K$ bytes.
   - **Triple Indirect Block Pointer**: Each of the second level indirect block is the same as the double indirect block, and supports $2^{64}K$ bytes. Because there are 2^8 indirect tables at the second level, the total number of block pointers is $2^8 \times 2^{64} = 2^{512}K$.

   In summary, the maximum file size is $10K + 2^8K + 2^{64}K + 2^{512}K = 10 + 2^8 + 2^{64} + 2^{512}$K. ∎