RECITATION: 104
NUMBER OF HOURS TO COMPLETE: 6 hours

1. *(5 pts) Explain why the SSTF scheduling policy tends to favor middle cylinders over the innermost and outmost cylinders.*

   This would allow for the shortest seek times because central disks have the shortest average distance to all tracks. The disks on the "edges" have much larger seek times should the scheduler schedule tracks on opposite ends.

2. *(5 pts) Why is rotational latency usually NOT considered in disk scheduling?*

   There is a level of imprecision when measuring the latency, and by the time the scheduler received this information from the disk, the information would already be "outdated" due to the variability in latency. For this reason, most disks just don't communicate this information.

3. *(10 pts) What are the tradeoffs with rereading code pages from the file system versus using swap space to store them.*

   Swap space has faster allocation speeds that the file system, which would allow for much higher data transfer speeds. However, unlike the file system, swap space requires startup time which would add a delay as opposed to simply paging out.

4. *(30 pts) Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4,999. The drive is currently serving a request at cylinder 2,150, and the previous request was at cylinder 1,805. 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) FCFS
   (b) SSTF
   (c) SCAN
   (d) LOOK
   (e) C-SCAN
   (f) C-LOOK

   (a) 13011 - first request processed first
   (b) 7586 - selects request with shortest seek time from head position
   (c) 7492 - moves back and forth across disk processing along the way
   (d) 7424 - like SCAN except it doesn't seek to the ends unless necessary
   (e) 9917 - after moving across disk once, moves back to beginning without processing
   (f) 9137 - same as combining concept from LOOK and C-SCAN

5. *(10 pts) Contrast the performance of the three allocation methods (contiguous, linked, indexed) for both sequential and random file access.*

   - Contiguous

     Sequential: Since the file is stored contiguously, information can easily be accessed sequentially. It allows for a minimal number of seeks.

     Random: Can easily analyze the blocks adjacent to determine which "direction" the intended block is.

   - Linked

     Sequential: Requires more seek time due to the non-contiguous storage.

     Random: Not preferred as it may require following the links to several disk blocks until you arrive at the intended block.

   - Indexed

     Sequential & Random: Works well for both since there exists a table of pointers to the data blocks minimizing any negative effects from random file access.

6. *(5 pts) Consider a system where free disk space is kept in a free-space list. Suppose the pointer to the free-space list is lost. Can the system reconstruct the free-space list? Explain your answer.*

   Yes, the system can reconstruct the free-space list using a bitmap. All you need to do is go through the file system and flip bits (from 0 to 1) for every occupied block (1 meaning occupied). Every bit still 0 at the end is a free block.

7. *(5 pts) Consider a file system similar to the one used by UNIX (indexed allocation). How many disk operations are required to access the contents of a file at* ***/dir1/dir2/file3****? Assume that none of the disk blocks is currently being cached.*

   If we infer that the file takes up some n number of blocks, then it would require $n + 3$ disk operations read the contents of the file. 1 for the root directory, 1 for **dir1**, 1 for **dir2**, and then n to read in the file since it wasn't cached.

8. *(30 pts) You are asked to allocate a file according to either a File Allocation Table (FAT) or multi-level indexed allocation (UNIX inode - triply indirect). Assume that the file is 2000 disk blocks long, there are 4 KB per disk block, each pointer in the FAT occupies 4 bytes, the first index block contains 15 entries (of which 12 are direct, and one each is singly indirect, doubly indirect, and triply indirect - see slides or textbook), every other index block contains 15 entries (may be indirect depending on the nesting level), each index block entry takes 4 bytes, and unused index blocks don't count in the total memory cost, though unused entries in partially filled index blocks do count. How many bytes are used to lay out the file when using a:*

   a *FAT file system?*

   b *UNIX-style file system?*

   Now suppose that you wish to read the 1099'th block of the file. Assume each of the following counts as one search operation:

   - *moving from one element to the next in a linked list*
   - *indexing into an index block*
   - *moving from index block to the next*

   How many searches are need to read block 1099 when using the

   c *same FAT file system as above?*

   d *same UNIX-style file system as above?*

   a $2000 * 4 = 8000$

   b
   - first level index blocks: $(12 * 4)$ bytes
   - singly indirect blocks: $(15 * 4)$ bytes
   - doubly indirect blocks: $(13 * 4) + (15 * 4 * 2)$ bytes
   - triply indirect blocks: $(13 * 4) + (13 * 4 * 2) + (15 * 4 * 4)$ bytes
   - all other blocks: $(12*4)+(15*4)+(13*4)+(15*4*2)+(13*4)+(13*4*2)+(15*4*4)$

   c 1099

   d The fourth.