Operating Systems, Spring 2020, Exercise 5

1. (Problems 11.7, 11.12 [Stal12] modified)
   Disks, RAID
   a. How much disk space (in sectors, track, surfaces) will be required to store 300,000 120-byte logical records if the disk is fixed sector with 512 bytes/sector, with 96 sectors per track, 120 tracks per surface, and 16 usable surfaces? Ignore any file header record(s) and track indexes, and assume that records cannot span two sectors.
   b. Consider a 4-drive, 200 GB-per-drive RAID array. What is the available data storage capacity for each of the RAID levels, 0, 1, 3, 4, 5, and 6?
   c. Assume that you need to update one character in one disk block. How many disk transactions will it require from RAID-5 system? How do you compute new parity blocks. Which disk ops can occur concurrently?

2. Disk cache. See Fig 11.9 (slide 55 in Ch 11)
   a. Which problem does the algorithm in Fig. 11.9 (a) solve?
   b. What happens if the "New section" in Fig. 11.9 (a) is too small or too large?
   c. Which problem in Fig. 11.9 (a) algorithm does the algorithm in Fig. 11.9 (b) solve?
   d. What happens if the "Middle section" in Fig. 11.9 (b) is too small or too large?
   e. How do the algorithms in Fig. 11.9 take into consideration that some blocks might be dirty (written)?
   How would you make the algorithms work better in this regard?

3. Linus Elevator
   a. How does the (basic) Linus Elevator Scheduler (LES) differ from standard Elevator algorithm?
   b. Why is there the Deadline Scheduler better than LES? What problem does the Deadline Scheduler try to solve? How does it solve it?
   c. Why is the Anticipatory I/O Scheduler better than either one of the previous algorithms? What problem does Anticipatory I/O Scheduler try to solve? When do you wait that (e.g.) 6 ms and how do you get that time back?

4. (Problem 12.2 [Stal12], modified)
   Consider a file with 4 million records. Find how many record (file) accesses, on average, are required to search for a particular key (say X) for
   a. A sequential file with all 4 million records.
   b. An indexed sequential file with index containing 2000 entries (with the keys in the index more or less equally distributed over the main file).
   c. Multiple levels of indexing in indexed sequential file: a lower-level index with 40,000 entries is constructed and a higher-level index that is constructed into the lower-level index of 200 entries.

5. Assume that you are using B-trees to implement indexed file. Assume that you have some 100000 records of flight tickets, that are indexed with 3 different keys (e.g., customer id "cust", ticket id "ticket", and credit card number "card"). Each customer can have many flights, and each credit card can be used in many flights for many customers. Each ticket id is unique.
   a. How do organize the file structure now? What type of B-trees do you have? What is stored in each node?
   b. How do you locate all records for customer 883, ticket number 43267, or credit card 123-5432?
   c. What happens when you delete one record (e.g., ticket 33221)? What needs to modified and how?
   d. What happens when you update one record (e.g., ticket 44221)? What needs to modified and

how?

e. What happens when you add one record (e.g., ticket 55221)? What needs to modified and how?

6. (Problem 12.13 [Stal18], modified)
   Consider the organization of a UNIX file as represented by the inode (see Fig 12.15, slide 66 in Ch 12). Assume there are 12 direct block pointers, and a singly, doubly, and triply indirect pointer to each inode. Further, assume the system block size and the disk sector size are both 8 KB.If the disk block pointer is 32 bits, with 8 bits to identify the physical disk and 24 bits to indetify the physical block, then: ue.
   a. What is the maximum file size supported by this system?
   b. What is the maximum system partition supported by this system?
   c. Assuming no information other than the the file inode is already on the main memory, how many disk accesses are required to access the byte in position 13,423,956?
   d. Draw an image, similar to Fig 12.15 and 12.16 (slides 63 and 66 in Ch 12), to show where all data blocks (directory and file data) needed to access a small (2 block) file /AAA/BBB/test.c are located.