John Hodson
Principles of Computer System Design
Homework 3

a) Problem set 1 (Bigger Files)

1. Which of the following adjustments will allow files larger than the current 1-gigabyte limit to be stored?

   Answer choice B. will result in an overflow and these will not be indexable (Increase just the number of bytes per block from 512 to 2048 bytes)

   D. Replace one of the direct block numbers in each inode with an additional triple-indirect block number.

   9 direct blocks -> 10 blocks * 512 bytes/block = 5120 bytes

   10^{th} block is a single indirect -> 128 * 512 =

   11^{th} block is a triple indirect -> 128 double-indirect blocks/triple-indirect block * 128 indirect blocks/double-indirect block * 128 blocks/indirect block * 512 bytes/block = 128^3 * 512 blocks = 1073741824 bytes

   12^{th} block is a double indirect = 128^2 * 512 = 8388608 bytes

13th block is a triple indirect = 1073741824 bytes

2155942400 bytes total in this proposed file

2. Which of the following adjustments (without any of the modifications in the previous question), will allow files larger than the current approximately 1-gigabyte limit to be stored?

   B. Decreasing the size of a block number from 4 bytes to 3 bytes.

   If you were to increase the size of a block number, this would make file size smaller. Thus, we know it must be either B or C. Answer choice C results in not being able to index files of 1 gigabyte in size, even if you were to use every available block (2^16). Thus, B.

b) Problem set 4 (EZPark)

1. Which of these statements is true about the problems with Ben's design?

A. There is a race condition in accesses to available[], which may violate one of the correctness specifications when two find_spot () threads run.

C. There is a race condition in accesses to numcars, which may violate one of the correctness specifications when more than one thread updates numcars.

2. **No, Alyssa's code does not solve the problem. In the FIND_SPOT() procedure, she has placed the second lock release in the body of the for loop. This will cause the lock to be released at the end of every iteration of the for loop, without ever acquiring it again. This means that the first iteration of the for loop is safe, but the remaining iterations could exhibit race conditions.**
3. **No, this lock is never released, so after this thread runs, no thread can ever acquire this lock again – deadlocking the program.**
4. **Yes, this program meets specifications. The lock is acquired at the start of each iteration of the for loop and released at the commencement of each**

iteration – making the body of the for-loop thread safe. Other threads can safely acquire and release this lock and no deadlock is experienced.

5. **No, he never releases the lock in the FIND_SPOT() method and never acquires the lock in the RELINQUISH_SPOT() method. This means that for each car that leaves – only one can enter and vice versa. This is a result of where he has placed the locks – and if the parking lot is empty to begin with – there will only ever be 1 car in the parking lot at a time.**
6. **A. The client will not get a response until at least one car relinquishes a spot.**
7. **A. The server may be running multiple active threads on behalf of the same client car at any given time.**
8. **B. On any thread switch, the operating system saves the values of the pc, sp, and several registers.**