# CS 537: Introduction to Operating Systems
# Fall 2023: Midterm Exam #3 ALTERNATE

## Part 1. Designate if the statement is True (A) or False (B).

1. Device driver code is software that is typically part of the OS and runs on the main CPU.
**True, device driver code is typically part of the OS and runs on the main CPU, though its protocols may be specific to each peripheral device.**
2. Using Direct Memory Access (DMA) lowers CPU overhead.
**True, Direct Memory Access (DMA) lowers CPU overhead**
3. Typically device drivers comprise only a small portion of kernel code.
**False, in Linux over 70% of the OS is found in device drivers.**
4. With a 15,000 RPM disk, the expected rotation time for a random access is 4ms.
**False, this is the full rotation time. The expected rotation time is half of that (2ms).**
5. Transfer time is the same for random accesses as compared to sequential accesses.
**True, transfer time is the same for random or sequential access. The higher time for random access is due to the more frequent seek time between blocks of data.**
6. One difference between the SCAN and C-SCAN algorithms is that C-SCAN reads/writes data when the head is traveling in either direction.
**False, SCAN reads/writes data when the head is traveling in both directions while Circular Scan (C-SCAN) only reads/writes data when the head is traveling from outer to inner tracks and then resets.**
7. A non-work conserving scheduler may schedule available requests even when the disk is idle.
**True, IO requests are buffered and performed in batch by non-work conserving schedulers and these batches may occur when the disk is idle.**
8. A disk cylinder is composed of all tracks at the same distance on all surfaces of every platter.
**True.**
9. The Shortest Seek Time First algorithm doesn’t suffer from starvation.
**False, imagine a stream of requests that are close to where the head is currently positioned.**
10. The SPTF algorithm is usually performed inside the OS.
**False, since the location of the head and rotation position must be known to implement it, which is not easy within the OS, it is usually implemented in a drive.**
11. A RAID-0 system has a capacity that is the same as using the disks without being in a RAID.
**True, a RAID-0 system rotates which disk to write blocks to, but there is no redundancy of the data.**
12. Striping in a RAID is a way to design the system to have more redundancy in the data, by spreading the data over multiple disks.
**False, striping helps to extract the most parallelism when requests are made to contiguous chunks. All disks can be actively handling a portion of a sequential request. Striping alone does not add any redundancy to a system.**
13. RAID-4 has better capacity than RAID-1 but worse reliability than RAID-0.
**False,, It is true that RAID-4 has just one parity disk for N data disks, whereas RAID-1 has an extra mirror disk for every data disk, but RAID-0 cannot withstand any disk failures and RAID-4 can handle any single disk failure.**
14. One advantage of RAID-4 over RAID-5 is that it can reconstruct the parity data if the parity disk fails while RAID-5 cannot.

**False, it is true that RAID-4 can reconstruct the parity data but so can RAID-5.**
15. The capacity of RAID-4 is greater than RAID-5, since only one disk is used for the parity.
**False, the only difference is where the parity information is stored.**
16. The throughput rate for random writes in a RAID-5 system is double that of a RAID-4 system.
**False, the rate for RAID-5 is N/4*R and for a RAID-4 system it is R/2.**
17. The latency of a write in a RAID-5 system is four times that of a single disk because it requires doing both a read and a write for each of 2 logical writes.
**False, the data and parity blocks must be read in order to calculate the new parity value before it can be written. The latency is two times that of a single disk not four because the reads and writes can be done in parallel.**
18. In an FFS-like file system, multiple directory entries may point to the same inode.
**True, this is often done by calling link**
19. An inode structure typically contains a field indicating the time at which this file was last accessed.
**True, they do ...**
20. An inode structure could contain pointers to directory data or file data.
**True, inodes contain pointers to either file or directory data. (I did change the wording of the question)**
21. FFS attempts to put inodes of files that are in the same directory in the same cylinder group.
**True, to maximize namespace locality.**
22. After a crash, FSCK reverts the on-disk state of the file system to match its state before the last update to disk began.
**False, FSCK just tries to make the on-disk state consistent, it doesn't know the correct state of the file system before a particular update. ( I just changed the wording on the question)**
23. A data journaling file system checkpoints data blocks to their final in-place positions and then a commit block is written.
**False, after committing the next step is checkpointing.**
24. The difference between data journaling and metadata (order) journaling is the type of blocks that are written to the journal and the order that blocks are copied from the journal to their final in-place positions during checkpointing.
**False. It is the and part that is false.  The difference is the type of blocks that are written to the journal. In metadata journaling only the metadata blocks are written to the journal.**
25. When LFS writes a new copy of an inode to a segment, it does not write a new copy of all data that the inode points to.
**True, the inode will contain pointers to wherever the data blocks for the inode are located, which may have some data blocks in other segments that aren't rewritten in the new segment.**
26. LFS periodically checkpoints pointers to imap pieces to a known location on disk (alternating between two locations to withstand crashes).
**True, LFS checkpoints pointers to portions of the imap. The imaps themselves are written in various segments.**
27. When performing garbage collection, LFS determines that a data block is valid by scanning all valid inodes from the imap and verifying that one of the valid inodes points to this location.
**False, this would be way too slow. Instead, LFS writes segment summary information to each segment that describes each updated data block (the inode that points to it and its offset in the file).**
28. SSDs erase at the block level but read and program at the page level.
**True**
29. Block corruption in cheap disks is more common than latent sector errors.
**False, the study of 1.5 million disks over 3 years found that LSEs were more common than block corruption.**

30. To detect a lost write you must place the checksum with the data block.
**False, the ZFS file system places the checksum in the inode. Placing it with the data block may make lost writes undetectable.**

## Part 2. Disks -- select the one best answer, A - E.

Consider a disk with the following characteristics:

• Number of surfaces: 4 (=2^2)
• Number of tracks / surface: 2 M (=2^21)
• Number of bytes / track: 4 MB (=2^22 bytes)
• Number of sectors / track: 4 K (=2^12)

31. How many read/write heads does this disk have?
A. 1
B. 2
C. 4
D. 8
E. Not enough information or none of the above
**4 heads: 1 head per surface**
32. What is the size of each sector?
A. 1 KB
B. 2 KB
C. 3 KB
D. 4 KB
E. Not enough information or none of the above
**1KB, 2^22 bytes/track * track/2^12 sectors = 2^10 bytes/sector**
33. How many bytes per cylinder?
A. 8 MB
B. 16 MB
C. 32 MB
D. 64 MB
E. Not enough information or none of the above
**16 MB, 2^22 bytes / track * 4 tracks / cylinder = 2^24 bytes / cylinder**
34. What is the total capacity of this disk?
A. 2^42 bytes
B. 2^43 bytes
C. 2^45 bytes
D. 2^46 bytes
E. Not enough information or none of the above
**2^45, 2^22 bytes/track * 2^21 tracks/surface * 4 surfaces**
Assume that the disk head is at cylinder 12 and moving towards higher cylinders. Assume a stream of requests arrives for the following cylinders: 2, 20, 10, 6, 60, 45, 25, 65, 17, 47, 24, 44

35. With a SSTF scheduling policy, what is the total seek distance?
A. 63
B. 73
C. 124
D. 284
E. Not enough information or none of the above
**73, start at 12 and go Schedule: 10, 6, 2, 17, 20, 24, 25, 44, 45, 47, 60, 65**
36. With a FCFS scheduling policy, what is the total seek distance to service all the requests?
A. 63
B. 302
C. 292
D. 284
E. Not enough information or none of the above
**292, start at 12 to 2, then 20, then 10, … = 10 + 18 + 10 + 4 + 54 + 15 + 20 + 40 + 48 + 30 + 23 + 20 = 292**
37. With a SCAN scheduling policy, what is the total seek distance?
A. 112
B. 116
C. 157
D. 119
E. Not enough information or none of the above
**116, start at 12 and go Schedule: 17, 20, 24, 25, 44, 45, 47, 60, 65, 10, 6, 2**
38. With a C-SCAN scheduling policy (servicing only when head is moving towards higher cylinders), what is the total seek distance?
A. 157
B. 114
C. 124
D. 173
E. Not enough information or none of the above
**124, start at 12 and go Schedule: 17, 20, 24, 25, 44, 45, 47, 60, 65, 2, 6, 10**

## Part 3. Basic File System Operations and Data Structures -- select the one best answer, A - E.

These questions ask you to understand how different file system operations lead to different file system data structures being modified on disk. You do not need to consider journaling or crash consistency in these questions.

This file system supports 7 operations:

| | |
|---|---|
| - mkdir() | - creates a new directory |
| - creat() | - creates a new (empty) file |
| - open(),write(), close() | - appends a block to a file |
| - link() | - creates a hard link to a file |
| - unlink() | - unlinks a file (removing it if linkcnt==0) |

The state of the file system is indicated by the contents of four different data structures:

| | |
|---|---|
| inode bitmap | - indicates which inodes are allocated (not shown because not needed for questions) |
| inodes | - table of inodes and their contents |
| data bitmap | - indicates which data blocks are allocated (not shown) |
| data | - indicates contents of data blocks |

The inodes each have three fields: the first field indicates the type of file (f for a regular file, d for a directory); the second indicates which data block belongs to a file (here, files can only be empty, which have the address of the data block set to -1, or one block in size, which would have a non-negative address); the third shows the reference count for the file or directory. For example, the following inode is a regular file, which is empty (address field set to -1), and has just one link in the file system: [f a:-1 r:1]. If the same file had a block allocated to it (say block 10), it would be shown as follows: [f a:10 r:1]. If someone then created a hard link to this inode, it would then become [f a:10 r:2].

Data blocks can either retain user data or directory data. If filled with directory data, each entry within the block is of the form (name, inumber), where "name" is the name of the file or directory, and "inumber" is the inode number of the file. Thus, an empty root directory looks like this, assuming the root inode is 0: [(.,0) (..,0)]. If we add a single file "f" to the root directory, which has been allocated inode number 1, the root directory contents would then become: [(.,0) (..,0) (f,1)].

If a data block contains user data, it is shown as just a single character within the block, e.g., "h". If it is empty and unallocated, just a pair of empty brackets ([]) are shown.

Empty inodes and empty data blocks may not all be shown.

An entire file system is thus depicted as follows:

```
inode bitmap 11110000
inodes [d a:0 r:6] [f a:1 r:1] [f a:-1 r:1] [d a:2 r:2] [] ...
data bitmap 11100000
data [(.,0) (..,0) (y,1) (z,2) (f,3)] [u] [(.,3) (..,0)] [] ...
```

This file system has eight inodes and eight data blocks. The root directory contains three entries (other than "." and ".."), to "y", "z", and "f". By looking up inode 1, we can see that "y" is a regular file (type f), with a single data block allocated to it (address 1). In that data block are the contents of the file "y": namely, "u". We can also see that "z" is an empty regular file (address field set to -1), and that "f" (inode number 3) is a directory, also empty. You can also see from the bitmaps that the first four inode bitmap entries are marked as allocated, as well as the first three data bitmap entries.

Assume the initial state of the file system is as follows:

```
inodes [d a:0 r:2] [] [] [] [] []
data [(.,0) (..,0)] [] [] []
```

If the file system transitions into each of the following states, what operation or operations must have been performed? The state of the file system is cumulative across questions.

39. File System State:
```
inodes [d a:0 r:3] [d a:1 r:2] [] [] [] []
data [(.,0) (..,0) (c,1)] [(.,1) (..,0)] [] []
```
A. mkdir("/c");
B. creat("/c");
C. link("/","/c");
D. mkdir("/c/d");
E. None of the above
**mkdir("/c");, a new dir inode exists,  a new dirent is located inside of the root directory linking "c" to the inode, and a new data block is present in block 1 with the default "." and ".." links.**
40. File System State:
```
inodes [d a:0 r:3] [d a:1 r:2] [f a:-1, r:1] [] [] []
data [(.,0) (..,0) (c,1)] [(.,1) (..,0) (w,2)] [] []
```

A. mkdir("/w");
B. creat("/c/w");
C. creat("/w");
D. link("/c","/w");
E. None of the above
**creat("/c/w");, a new file inode exists and a new dirent in the "/c" directory data points to it.**
41. File System State:
```
inodes [d a:0 r:3] [d a:1 r:2] [f a:-1, r:1] [f a:-1, r:1] [] []
data [(.,0) (..,0) (c,1) (w,3)] [(.,1) (..,0) (w,2)] [] []
```

A.  mkdir("/w");
B. creat("/c/w");
C. creat("/w");
D. link("/c","/w");
E. None of the above
**creat("/w");, a new file inode exists and it is referred to from the root directory data block.**
42. File System State after **more than 1** operations:
```
inodes [d a:0 r:3] [d a:1 r:2] [f a:2, r:2] [f a:-1, r:1] [] []
data [(.,0) (..,0) (c,1) (w,3) (x, 2)] [(.,1) (..,0) (w,2)] [v] []
```

A. creat("/x");
fd=open("/c/w", O_WRONLY|O_APPEND); write(fd,buf,BLOCKSIZE);

B. link("/w","/x");
fd=open("/c/w", O_WRONLY|O_APPEND); write(fd,buf,BLOCKSIZE); close(fd);

C. creat("/x");
creat("/c/w");

D. link("/c/w","/x");
fd=open("/c/w", O_WRONLY|O_APPEND); write(fd,buf,BLOCKSIZE); close(fd);

E. None of the above
**link("/c/w","/x");fd=open("/c/w", O_WRONLY|O_APPEND); write(fd,buf,BLOCKSIZE); close(fd);, the**

**reference count for inode 2 went up by one and there is a dirent in the root referring to it. Also, the address in the same inode now points to data block 2.**

43. File System State after **TWO** operations:

```
inodes [d a:0 r:4] [d a:1 r:2] [f a:2, r:2] [d a:3, r:2] [] []
data [(.,0) (..,0) (c,1) (x,2) (z, 3)] [(.,1) (..,0) (w,2)] [v] [(.,3) (.., 0)]
```

A.unlink("/c/w");
mkdir("/z");

B. unlink("/w");
mkdir("/z");

C. unlink"/w");
create("/z");

D. unlink("/c/w");
creat("/z");

E. None of the above

**unlink("/w");mkdir("/z");, inode 3 is now a directory instead of a file and the dirent in the root directory for w is gone. Also, a new dirent is found in the root directory referring to z and a new data block with the "." and ".." entries of the new directory is written.**

## Part 4. Crash Consistency and Journaling -- select the one best answer, A - E.

Imagine you have an FFS-like file system that is creating a new empty file in an existing directory and must update 4 blocks: the directory inode, the directory data block, the file inode, and the inode bitmap. Assume the directory inode and the file inode are in different on-disk blocks. Assume this initial system does not perform any journaling and FSCK is not run. What happens if a crash occurs after only updating the following block(s)?

44. Bitmap
A. Data/inode leak
B. Multiple file paths may point to same inode
C. Point to garbage
D. Multiple problems listed above
E. No inconsistency (it simply appears that the operation was not performed)
**Data/inode leak, The bitmap shows that this inode has been used, but no structure is actually pointing to this inode.**
45. File inode
A. No inconsistency (it simply appears that the operation was not performed)
B. Data/inode leak
C. Multiple file paths may point to same inode
D. Point to garbage
E. Multiple problems listed above
**No inconsistencies, if only the file inode is written to then it is not connected into the file system (i.e. no directory entries point to it and the bitmap still says the inode is free).**
46. Directory inode and Directory data

A. No inconsistency (it simply appears that the operation was not performed)
B. Data/inode leak
C. Multiple file paths may point to same inode
D. Point to garbage
E. Multiple problems listed above
**Multiple problems exist, the directory will point to an inode but the inode will not contain the proper content (problem D), since bitmap was not updated, the inode may be used again (problem C), and if the user tries to create the file again, it will say file already exists.**
47. Bitmap and File inode
A. No inconsistency (it simply appears that the operation was not performed)
B. Data/inode leak
C. Multiple file paths may point to same inode
D. Point to garbage
E. Multiple problems listed above
**Data/inode leak, Since no directory entry points to the file inode, the file is not connected into the file system, but the bitmap indicates the inode is used so will not be allocated again.**
48. Bitmap and Directory inode and Directory data
A. No inconsistency (it simply appears that the operation was not performed)
B. Data/inode leak
C. Multiple file paths may point to same inode
D. Point to garbage
E. Multiple problems listed above
**Point to garbage, a directory entry exists to an inode but the inode was not properly setup so the directory entry points to garbage. Because the bitmap shows this inode is already allocated, it will not be allocated to another file.**
49. File inode and Directory inode and Directory data
A. No inconsistency (it simply appears that the operation was not performed)
B. Data/inode leak
C. Multiple file paths may point to same inode
D. Point to garbage
E. Multiple problems listed above
**Multiple problems listed above - Multiple file paths may point to same inode, since the bitmap was not updated the inode may be allocated to another file, causing multiple files to point to the same inode. Data/inode leak may occur when another file is allocated, any data allocated to the old inode will become unreachable (leaked).**
Now assume a basic implementation of metadata journaling has been added to the file system and the same file create operation is performed. Assume a transaction header block and a transaction commit block are used. Assume each block is written in its entirety (no crashes in the middle of writing a block) and only one block is written at a time and completes before the next block is written. If the system crashes after the following number of blocks have been written to disk, what will happen after the system reboots? (If the number of disk writes exceeds those needed, assume they are unrelated.)

50. 1 disk write (hint: just the transaction header block is written to disk)
A. Transaction replayed during recover; file system in old state
B. Transaction replayed during recover; file system in new state
C. Transaction replayed during recover; file system in unknown state
D. No transactions replayed during recover; file system in old state
E. No transactions replayed during recover; file system in new state
**No replay, file system remains in old state. Since the transaction wasn't committed, it will not be**

**replayed. Note: ALL blocks of this I/O operation are metadata - there are no data blocks, and so the first block to be written is the header block of the transaction.  All blocks are: (TxB,**

51. 4 disk writes (hint: transaction header, plus 3 additional blocks to journal)
A. No transactions replayed during recover; file system in old state
B. No transactions replayed during recover; file system in new state
C. Transaction replayed during recover; file system in old state
D. Transaction replayed during recover; file system in new state
E. Transaction replayed during recover; file system in unknown state
**No replay, file system remains in old state. Since the transaction wasn't committed, it will not be replayed**

52. 6 disk writes
A. No transactions replayed during recover; file system in old state
B. No transactions replayed during recover; file system in new state
C. Transaction replayed during recover; file system in old state
D. Transaction replayed during recover; file system in new state
E. Transaction replayed during recover; file system in unknown state
**Transaction replayed, since it was committed, it will be checkpointed.  File system will be in a new state.**

53. 10 disk writes
A. No transactions replayed during recover; file system in old state
B. No transactions replayed during recover; file system in new state
C. Transaction replayed during recover; file system in old state
D. Transaction replayed during recover; file system in new state
E. Transaction replayed during recover; file system in unknown state
**Transaction replayed, since it was committed, it will be checkpointed even though the metadata was copied to the file system proper, it will be redone since the transaction hasn't been marked as completed.  File system will be in a new state.**

54. 11 disk writes
A. No transactions replayed during recover; file system in old state
B. No transactions replayed during recover; file system in new state
C. Transaction replayed during recover; file system in old state
D. Transaction replayed during recover; file system in new state
E. Transaction replayed during recover; file system in unknown state
**No transaction replayed but FS in new state, the transaction completed and was marked as such before the crash so it is not replayed.**

## Part 5.  Data integrity, LFS, and SSDs -- select the one best answer, A - E.

A 4 byte block is written to disk with the data 0xD191100B (binary 1101 0001 1001 0001 0001 0000 0000 1011) and its checksum value.

55. What is the 1 byte XOR checksum value?
A. 0x5B (binary 0101 1011)
B. 0xA4 (binary 1010 0100)
C. 0xC1, 0x9A (binary 1100 0001,  1001 1010)
D. 0x3E, 0x65 (binary 0011 1110,   0110 0101)
E. None of the above
**0x5B, line up the 4 bytes and perform bit-wise xor of each column ( even number of 1s in each**

**column)**
CORRECT
1101 0001
1001 0001
0001 0000
0000 1011
--------------
0101 1011
0x5B

56. What is the 1 byte addition (ignoring overflow) checksum value?
A. 0x8B (binary 1000 1011)
B. 0x7D (binary 0111 1101)
C. 0xE9, 0x9C (binary 1110 1001, 1001 1100)
D. 0x16, 0x6C (binary 0001 0110, 0110 1100)
E. None of the above
**0x7D, line up the 4 bytes and perform addition, throwing away the overflow.**

CORRECT
1101 0001
1001 0001
0001 0000
0000 1011
**========**
**0111 1101 (throw out last extra 1)**
**0x7D**

57. Storing a physical identifier along with the checksum can be used to detect what type of errors?
A. Misdirected Write
B. Latent Sector Error
C. Lost write
D. Block corruption
E. None of the above
**Misdirected writes can be detected by storing the identifier of where the data was meant to be written (i.e. a physical identifier).**
An LFS system contains the following initial state:

INITIAL file system contents:
```
[   0 ] live checkpoint: 3 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[   1 ] live [.,0] [..,0] -- -- -- -- -- --
[   2 ] live type:dir size:1 refs:2 ptrs: 1 -- -- -- -- -- -- --
[   3 ] live chunk(imap): 2 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```
Notice that the checkpoint region lives in block 0 and initially contains the 0th entry pointing to the 3rd data block.  This data block holds the first 16 entries (index 0 through 15) of the imap, with inode 0 referring to block 2.  Block 2 is the inode of the root directory of the file system and this inode's data pointers refer to block 1.  Block 1 is the data for the directory, which initially has directory entries for "." and ".." which both refer back to inode 0.  also notice that all data blocks are marked live because they are reachable (i.e. connected into the file system).

An I/O operation is performed on the LFS which modifies the system contents to the final system state:

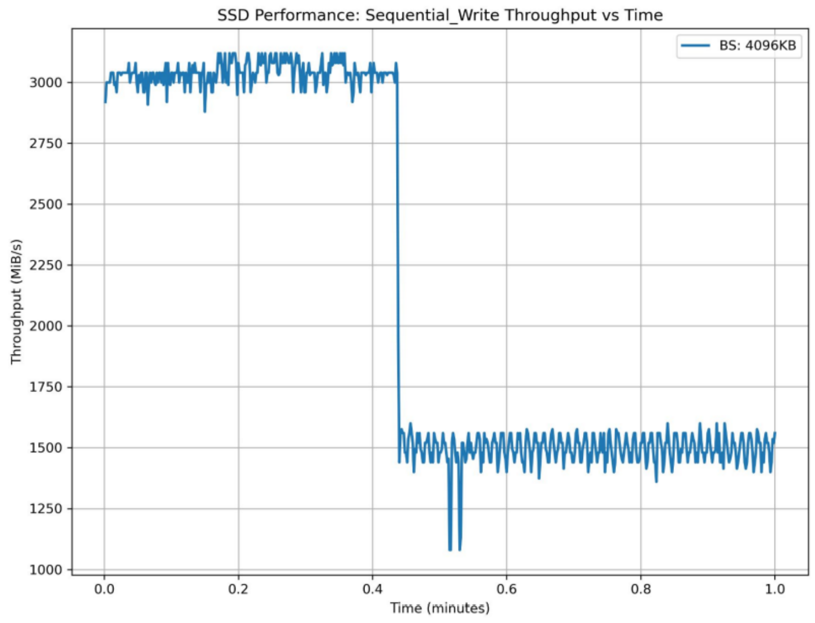
FINAL file system contents:

```
[   0 ]  ?   checkpoint: 8 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[   1 ]  ?   [.,0] [..,0] -- -- -- -- -- --
[   2 ]  ?   type:dir size:1 refs:2 ptrs: 1 -- -- -- -- -- -- --
[   3 ]  ?   chunk(imap): 2 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[   4 ]  ?   [.,0] [..,0] [ku3,1] -- -- -- -- --
[   5 ]  ?   type:dir size:1 refs:3 ptrs: 4 -- -- -- -- -- -- -
[   6 ]  ?   [.,1] [..,0]
[   7 ]  ?   type:dir size:1 refs:2 ptrs: 6 -- -- -- -- -- -- -- --
[   8 ]  ?   chunk(imap): 5 7 -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

58. What I/O operation was performed?
A. creat("/ku3");
B. mkdir("/ku3");
C. link("/","/ku3");
D. unlink("/ku3");
E. None of the above
**mkdir("/ku3");, a new inode has been created that is a directory file type with an associated data blocks and a directory entry within the root inode's data block for "ku3" exists.**
59. Which data blocks are live?
A. 4, 5,  6, 7, 8
B. 0, 1, 2, 3, 4, 5, 6, 7, 8
C.  0, 4, 5, 6, 7
D. 0, 4, 5, 6, 7, 8
E. None of the above
**0, 4, 5, 6, 7, 8 -- 0 is the checkpoint region and from it you reach 7 (the imap). From the imap you can reach 5 and 6 (the two inodes). The inodes contain references to 4 and 6.**

60. A 2TB Samsung 970 EVO Plus SSD contains ~1.9TB of TLC with 78 GB of SLC Cache.  It is benchmarked by doing sequential writes and observing the throughput rate.  The plot below is the result

of this benchmarking.



What is the most likely reason for the dramatic drop in throughput rate just after 0.4 minutes?

A. A block is fully programmed and a new block must be erased before it can be programmed

B. The FTL's mapping table is full and must resort to a backup mapping table
C. The disk is full and garbage collection starts running
D. The SLC is filled and now data must be transferred at the slower TLC rate
E. A page is fully programmed and a new page must be erased before it can be programmed

**The SLC is full and data transfer must occur at the slower TLC rate. The disk is not nearly filled so the GC doesn't need to run yet, and many blocks will have been erased and pages programmed in this time so those cannot be the reason.**

**That’s it. Have a great break!**