CS 537:
The Forensics Interview
Spring ’18

There are 26 pages
There are 80 questions
Each question has ONLY ONE answer:
A, B, C, D, or E

Make sure you do them all!
(No penalty for guessing)

POSITION AVAILABLE

SYSTEM FORENSICS EXPERT

System Forensics Expert wanted.

Must be able to read exam questions about systems and figure out what the answers are. Ability to handle time pressure is a plus. Also useful: Skill in writing C code and being nice to professors. Must want to live in bitterly cold and desolate location in upper Midwest.

Computer forensics is the application of investigation and analysis techniques to gather and preserve evidence from a particular computing device in a way that is suitable for presentation in a court of law.

In this exam, **you are interviewing for a job in computer forensics.** Given a series of clues, your task is to learn something definitive about the system under study.

If you do well enough, congratulations, you're hired!

If not … well, at least you tried. Not everyone is cut out for forensics, as it turns out.

Read each question carefully! And good luck.

Details for the answer sheet:

- Fill in your **name** and **student ID** carefully on the answer sheet.
- Fill in **0** for **Special Code A** if you are a **graduate student**, **1** for **Special Code A** if you are an **undergraduate** (this is just for data analysis, it doesn't affect your grade).
- Fill out **one** of **A, B, C, D, or E** for each question.
- **Color each oval completely;** don't use a checkmark, box around the oval, or other oddities that will lead us to question your ability to follow instructions.
- Fill in the oval with **pencil**, not pen.
- If you skip a question, **make sure to fill in the correct bubbles**! Pay careful attention to numbering.

VERY IMPORTANT: Each question has **ONLY ONE ANSWER.** Do not fill out multiple bubbles for any one question!

RAID

You are given information about a specific RAID storage system, and need to determine which approach to RAID is being used.

In this first set of questions, you are given **timing information** about a request (or set of requests). From this information, what can you deduce?

Knowledge: All RAIDs we consider have **8 disks in them**, and that the size of a **chunk is 4 KB**
(i.e., the amount put on one disk before moving on to the next).
All requests, unless otherwise noted, are **chunk aligned**.
The logical **block size** exposed by the array is **4 KB**.

For example, this means that if we had a striping (RAID-0) array with 8 disks, it would spread blocks across the disks something like this (where block 0 is the first 4-KB block, block 1 is the second 4-KB block, etc.):

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15

A read to block 1 would mean that 4KB is read from (disk 1, offset 0), as highlighted above.

IMPORTANT: Each question is about a different RAID; there is no connection between the questions.

Question 1: For a specific RAID array (call it "RAID A"), a read of a block takes about 10ms. A write of a block also takes about 10ms. This RAID is likely:
a) RAID-1 (mirroring)
b) RAID-4 (parity disk)
c) RAID-5 (rotating parity)
d) RAID-4 or RAID-5
e) All of the above

Question 2: for "RAID B", two small random writes usually take about twice as long as one random write. This RAID is likely:
a) RAID-1 (mirroring)
b) RAID-4 (parity disk)
c) RAID-5 (rotating parity)
d) RAID-4 or RAID-5
e) All of the above

Question 3: for "RAID C", a large write (of 7 blocks) usually takes about as much time as a small write (1 block). This RAID is likely:
a) RAID-1 (mirroring)
b) RAID-4 (parity disk)
c) RAID-5 (rotating parity)
d) RAID-4 or RAID-5
e) All of the above

RAID (continued)

Question 4: For “RAID D”, the overall throughput (measured in MB/s) is about 4 MB/s when issuing many 1-block random writes. In comparison, a comparable RAID array configured to use striping (RAID-0) achieved a throughout of about 8 MB/s. This RAID (RAID D) is likely:
a) RAID-1 (mirroring)
b) RAID-4 (parity disk)
c) RAID-5 (rotating parity)
d) RAID-4 or RAID-5
e) All of the above

Question 5: for “RAID E”, the overall throughput when writing large sequential blocks to disk is about 700 MB/s. In comparison, large sequential writes to a RAID 0 achieves about 800 MB/s. This RAID (RAID-E) is likely:
a) RAID-1 (mirroring)
b) RAID-4 (parity disk)
c) RAID-5 (rotating parity)
d) RAID-4 or RAID-5
e) All of the above

For the next part of the RAID question, you discover a new forensic tool: the **ability to measure the number of physical I/Os** that happen in the system (you do this by attaching probes to the internal I/O busses of the RAID array in question; nice work!).

Question 6: For “RAID F”, you know that the RAID is likely RAID-4 or RAID-5. You issue a single perfectly aligned block **write**. The number of physical I/Os you measure on RAID F during this write is:
a) 0
b) 1
c) 2
d) 3
e) 4

Question 7: For “RAID G”, you have already figured out that it is likely a RAID-1 (mirroring). You then issue a single **write** to the RAID. The write is small (1 block) is aligned. The number of physical I/Os you measure on the disks of RAID G is always:
a) 0
b) 1
c) 2
d) Sometimes =2, sometimes >2
e) Always > 2

RAID (continued)

Question 8: For “RAID H”, you have already figured out that it is likely a RAID-1 (mirroring). You then issue a single **write** to the RAID. The write is small (1 block); however, it is **not necessarily aligned**. The number of physical I/Os you measure on the disks of RAID H is always:
a) 0
b) 1
c) 2
d) Sometimes =2, sometimes >2
e) Always > 2

Question 9: For “RAID I”, you already know that it is RAID-5 (rotating parity). You then issue a single small **read** (1 block), **which is never aligned.** The number of physical I/Os you measure on the disks of RAID I is always:
a) 0
b) 1
c) 2
d) Sometimes =2, sometimes >2
e) Always > 2

Question 10: For “RAID J”, you measure the total number of physical I/Os under a **read-only** workload. You find that it is equal to the number of logical reads issued to the RAID. You also see that one disk is never accessed. From this, you conclude that the RAID is:
a) RAID-0 (striping)
b) RAID-1 (mirroring)
c) RAID-4 (parity disk)
d) RAID-0 or RAID-1
e) RAID-0 or RAID-1 or RAID-4

Shingled Disk Drives

A new type of disk has come to market. It is called a “shingled disk”, and has the following properties. First, tracks are grouped into groups called “shingles”. Within each shingle, tracks are packed very tightly, such that when you write to a block on track K, you will likely overwrite (oops) the corresponding block on track K+1. Thus, tracks within a shingle should be written in order, from 0 to T-1 (assuming T tracks in each shingle); if you stick to writing tracks in order within a shingle, you won’t lose any data.

The reason for shingles: it allows disk manufacturers to pack even more data on disk, making disks cheaper.

The problem: you can’t easily overwrite a block in place; rather, the device needs to keep a mapping table of some kind (the **Shingle Translation Layer**, or **STL**), and write blocks out (within each shingle) in **log-structured fashion**.

Question 11: Assume the STL maps each **4KB** block to a location on the disk. How big must the STL be to hold translations for an entire **512-GB** disk? Assume a **4-byte disk address** for each entry in an array-like structure in the STL.
a) 1 MB
b) 128 MB
c) 512 MB
d) 1 GB
e) None of the above

Question 12: The STL size can be changed by mapping chunks in sizes other than the usual 4KB block. With each **doubling** of the block size, the STL:
a) Increases in size by 2x
b) Decreases in size by 2x
c) Increases in size by 4x
d) Decreases in size by 4x
e) None of the above

Question 13: In a shingled disk, all writes are log-structured. As a result, which of the following is **NOT** true:
a) Write performance is similar to a regular hard drive for sequential workloads
b) Write performance is similar to a regular hard drive for random workloads
c) Read performance is similar to a regular hard drive for sequential workloads
d) Read performance is similar to a regular hard drive for random workloads
e) All of the above are true

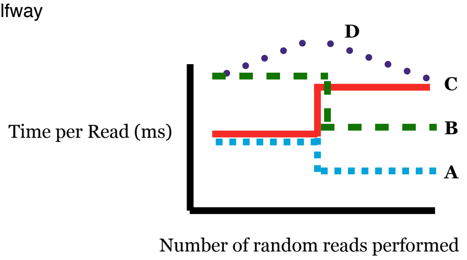
Question 14: The STL size can be changed by mapping chunks in sizes other than the usual 4KB block. With each **doubling** of the block size, the STL:
a) Increases in size by 2x
b) Decreases in size by 2x
c) Increases in size by 4x
d) Decreases in size by 4x
e) None of the above

Shingled Disk Drives (2)

Question 15: In a shingled disk, all writes are log-structured. As a result, which one of the these is **NOT** true:
a) Performance is similar to a regular hard drive for sequential write workloads
b) Performance is similar to a regular hard drive for random write workloads
c) Performance is similar to a regular hard drive for sequential read workloads
d) Performance is similar to a regular hard drive for random read workloads
e) Performance is similar to a regular hard drive for workloads with a mix of sequential reads/writes

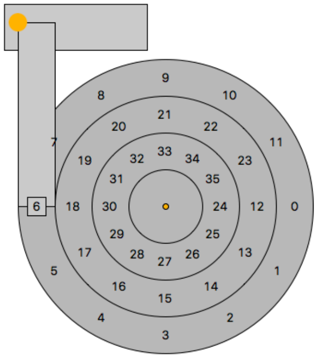
With this basic knowledge in place, let us now explore some forensics questions about a shingled drive you have.

Question 16: You have a suspicion that the STL cannot hold all mapping entries in device memory; as a result, **sometimes when reading from disk, an extra read is incurred** (to look up the mapping information). The workload used here is to repeatedly read some number of blocks N (many times), and report the **average time per read**; N is increased along the x-axis. Which line on the graph best reflects this result?
a) Lowest line with drop halfway
b) Next lowest dashed line line with drop halfway
c) Solid line with rise halfway
d) Topmost line which goes up then down
e) None of the above



The next questions will use the picture on the right, which reveals the internals of this shingled disk. As you can see, it only has three tracks, and looks remarkably like a disk simulator used in some OS classes.

Question 17: Assuming a **FIFO** scheduling policy, and a request stream which reads blocks **21, 17, 11, 7, 13,** what is the last block to be read?
a) 21
b) 17
c) 11
d) 7
e) 13



Question 18: Same question, but assume a **SSTF** scheduling policy, and a VERY FAST seek. What is the last block read?
a) 21
b) 17
c) 11
d) 7
e) 13

Shingled Disk Drives (3)

Question 19: Same question, but assume SATF scheduling policy, and a VERY FAST seek. What is the last block read?
a) 21
b) 17
c) 11
d) 7
e) 13

Question 20: In a shingled disk, all writes are log-structured. Specifically, writes are directed to the currently-being-written shingle, and the STL is updated accordingly. As a result, does disk scheduling of write requests (assuming a workload only contains write requests) help performance?
a) Yes (always)
b) Yes (sometimes)
c) No (never)
d) Can't answer without more details
e) None of the above

File System API

In this question, we explore the file system API. Although not a forensics task per se, what kind of forensic scientist doesn't know the file system API? Answer: an unemployed one.

Question 21: A file descriptor is:
a) a system-wide object used to access files
b) a per-process integer used to access files
c) readily forged
d) returned to a process via the `close()` system call
e) hard to understand

Question 22: Adding the O_TRUNC flag to the `open()` call will
a) puts the data in a locked "trunk" file
b) causes the open call to fail (usually)
c) implies you must add O_TRUNC to close() as well
d) creates the file (if it doesn't exist)
e) truncates the file to size=0

Question 23: The "unlink()" system call is to the program "rm" as THIS is to the "rmdir" program
a) unlink()
b) delete()
c) rmdir()
d) link()
e) fork()

Question 24: The "lseek()" call is used to
a) reposition the disk head
b) do a long disk seek, immediately
c) change the current file offset
d) force changes to disk
e) close files after a layoff period

Question 25: The following information is **NOT** available within a typical inode:
a) owner
b) size (bytes)
c) blocks allocated
d) file name
e) last access time

File System API (2)

Question 26: The “read()” call is to the “cat” program as the BLANK call is to the “ls” program:
a) read()
b) readdir()
c) stat()
d) fstat()
e) umount()

Question 27: Which type of links do most UNIX file systems support?
a) hyperlinks
b) soft links
c) forked links
d) sausage links
e) unlinks

Question 28: A file is **NOT**
a) a container for data
b) a byte array that can be read or written
c) something with a low-level name
d) easily deleted
e) something that can be referred to via a high-level name, thanks to directories

Question 29: Let’s say we wish to write data to a file and then force the contents of a file to disk. We should thus call:
a) write()
b) write() then fsync()
c) write() then fopen()
d) write() then falloc()
e) write() then fit()

Question 30: To atomically replace the contents of a file **`foo`**, we should use the following sequence of system calls: open() [to open a new temporary file], write() to write data to the disk, force the contents to disk (with a certain file system call, as in question 29), and finally:
a) close() to close the file
b) rename() to change the name of the temporary file to the desired file name
c) link() to link the file to another name
d) unlink() to remove the original file **`foo`**
e) None of the above

File System Implementation

In this bit of forensics, only bits and pieces remain on disk of a file system that looks very similar to the **Very Simple File System** (**VSFS**). VSFS, as you may recall, has an inode bitmap, some inodes, a data bitmap, and some data blocks. Here is a sample VSFS empty file system disk image:

```
inode bitmap  10000000
inodes        [d a:0 r:2] [] [] [] [] [] [] []
data bitmap   10000000
data          [(.,0) (..,0)] [] [] [] [] [] [] []
```

As you can see, only the (empty) root directory is allocated in this tiny file system. In the inode of the root directory, you see the following contents: **`[d a:0 r:2]`**. This shows that the inode is of type “directory” (**`d`**, or **`f`** it is a regular file), it points to one data block (**`a:0`**, where a is for address; 0 means data block 0), and its reference count is 2 (**`r:2`**).

After this, you perform certain operations on the empty root file system, but the contents are sometimes lost. Your job is to reconstruct the contents of the VSFS image.

Question 31: Assume you call **`mkdir(“/n”)`** on the empty root file system. The inode bitmap is missing; what should it look like:

```
inode bitmap  ????????
inodes        [d a:0 r:3] [d a:1 r:2] [] [] [] [] [] []
data bitmap   11000000
data          [(.,0) (..,0) (n,1)] [(.,1) (..,0)] [] [] [] [] [] []
```

a) `10000000`
b) `11000000`
c) `11100000`
d) `10100000`
e) `None of the above`

Question 32: Assume you had instead called creat(“/z”) on an empty file system. Unfortunately, in this case, the data block for the root directory has gone bad. What should be in there?

```
inode bitmap  11000000
inodes        [d a:0 r:2] [f a:- r:1] [] [] [] [] [] []
data bitmap   10000000
data          [CORRUPT!] [] [] [] [] [] [] []
```

a) `[(.,0) (..,0)]`
b) `[(.,0) (..,0) (z,1)]`
c) `[(.,0) (..,0) (z,2)]`
d) `[(.,0) (..,0) (/z,1)]`
e) `[(.,0) (..,0) (/z,2)]`

File System Implementation (2)

This file system represents the on-disk state after two operations:

```
inode bitmap 11100000
inodes       [d a:0 r:4] [d a:1 r:2] [d a:2 r:2] [] [] [] [] []
data bitmap  11100000
data         [(.,0) (..,0) (d,1) (w,2)] [????] [(.,2) (..,0)] [] [] [] [] []
```

Question 33: What are the contents of the missing data block (Block 1)?

a) `(.,0) (..,0)`
b) `(.,1) (..,0)`
c) `(.,2) (..,0)`
d) `foofoofoo`
e) None of the above

Question 34: Which two operations were run upon the empty file system to result in this state?

a) creat(“/d”); creat(“/w”);
b) creat(“/d”); link(“/d”, “/w”);
c) creat(“/w”); unlink(“/d”);
d) mkdir(“/d”); mkdir(“/w”);
e) None of the above

Let’s examine one particular corrupt file system image from VSFS for the next two questions:

```
inode bitmap  11000000
inodes        [d a:0 r:2] [f a:1 r:2] [] [] [] [] [] []
data bitmap   ????????
data          [(.,0) (..,0) (c,1) (m,1)] [foofoofoo] [] [] [] [] [] []
```

Question 35: Which ONE of the following is **not true** about the above file system state?

a) There is a proper root directory
b) The file “/c” exists
c) The file “/m” exists
d) If you read the first block of “/c”, you get “foofoofoo”
e) If you unlink “/c”, you can no longer read “foofoofoo”

Question 36: Which ONE of the following is **true** about the above file system state?

a) File “/m” is a hard link to “/c”
b) File “/c” is a hard link to “/m”
c) Both “/c” and “/m” are links to the same file
d) The root directory has many data blocks in it
e) The file “/m” has many data blocks in it

File System Implementation (3)

Here is another on-disk state to examine:

```
inode bitmap 11111100
inodes       [d a:0 r:3] [f a:1 r:1] [f a:2 r:1] [d a:3 r:2] [f a:- r:1]
             [f a:4 r:1] [] []
data bitmap  11111000
data         [(.,0) (..,0) (d,1) (g,2) (y,3) (j,5)] [d] [v] [(.,3) (..,0) (n,4)]
             [/u] [] [] []
```

Question 37: In this final file system state, which regular files exist?

a) Only `/d, /g, /j`
b) Only `/d, /g, /j, /n`
c) Only `/d, /g, /j, /y/n`
d) Only `/d, /g, /j, /y`
e) None of the above

Question 38: In the above file system format, what is the largest number of regular-file inodes that can be allocated?

a) 1
b) 6
c) 7
d) 8
e) As many as needed

Now let’s do a few general questions to wrap up.

Question 39: Bitmaps are useful as allocation structures because of all the reasons below **EXCEPT**:

a) They are compact
b) They are human readable
c) They allow for quick lookup of free space
d) They allow for lookup to readily find consecutive free blocks
e) Updates to them do not add any disk traffic

Question 40: VSFS has all the following features **EXCEPT:**

a) Regular files
b) Directories
c) Hard links
d) Simple allocation structures
e) Fast crash consistency

Journaling

You are now given some tasks about a journaling file system. The first task is to figure out what blocks would end up in a journal transaction, given some base knowledge of the system. We assume for these first questions that the system under inspection uses **data journaling mode**, in which all blocks (metadata and data) are first journaled before being updated in place. Assume the standard structures of a file system here: an **inode bitmap**, a **data bitmap**, a table of **inodes**, and **data blocks**.

Important: For the questions below, ignore any additional journal metadata that would be written (i.e., a transaction start and end block), and assume no reads to disk need to take place to complete the given action (i.e., relevant structures are cached in memory).

Question 41: Assume that a process **appends a data block** to an existing (small) file. How many blocks are written to the journal as part of this update?
a) 1
b) 2
c) 3
d) 4
e) None of the above

Question 42: Now assume that a process **reads** a block from a file. Reading, in this file system, updates the “last accessed time” field in the inode. How many blocks are written to the journal as part of this read?
a) 1
b) 2
c) 3
d) 4
e) None of the above

Question 43: Now a process **creates a 0-byte file** in the root directory (which does not have many entries in it, so there is room for another entry in an existing directory data block). How many blocks are written to the journal as part of this file creation?
a) 1
b) 2
c) 3
d) 4
e) None of the above

Question 44: Finally, a process **deletes a 1-byte file** from the root directory (leaving the root directory empty). Assuming the root directory only uses a single data block for its data, how many blocks are written to the journal as part of this file creation?
a) 1
b) 2
c) 3
d) 4
e) None of the above

Journaling (2)

For the next questions, we will now assume ordered mode journaling, which only writes metadata to the journal (user file data is written only once as a result). We otherwise make the same assumptions as above, and answer the same questions about ordered mode journaling.

Question 45: Assume that a process has **appends a data block** to an existing (small) file. How many blocks are written to the journal as part of this update?
a) 1
b) 2
c) 3
d) 4
e) None of the above

Question 46: Now assume that a process **reads** a block from a file. Reading, in this file system, updates the “last accessed time” field in the inode. How many blocks are written to the journal as part of this read?
a) 1
b) 2
c) 3
d) 4
e) None of the above

Question 47: Now a process **creates a 0-byte file** in the root directory (which does not have many entries in it, so there is room for another entry in an existing directory data block). How many blocks are written to the journal as part of this file creation?
a) 1
b) 2
c) 3
d) 4
e) None of the above

Question 48: Finally, a process **deletes a 1-byte file** from the root directory (leaving the root directory empty). Assuming the root directory only uses a single data block for its data, how many blocks are written to the journal as part of this file creation?
a) 1
b) 2
c) 3
d) 4
e) None of the above

Journaling (3)

Finally, you are asked a few questions about journaling, to see if your above answers can be trusted.

Question 49: Which of the following statements is **NOT** true about journaling file systems?
a) Journaling adds a new on-disk structure to the file system
b) Journaling is the same as write-ahead logging (the terms are used interchangeably)
c) Journaling generally increases the amount of write traffic to the disk
d) Journaling always makes performance worse (than the same file system without journaling)
e) Whether in data or ordered journaling modes, file system metadata is always first written to the journal before being updated in place.

Question 50: Which of the following best represents a final, complete, and most optimized version of the **ordered** (metadata only) journaling protocol?
a) Data write, then journal metadata write, then journal commit.
b) Data write, then journal metadata write, then journal commit, then checkpoint of metadata.
c) Data write, then journal metadata write, then journal commit, then checkpoint of metadata, then (later) mark the transaction free in the journal superblock.
d) Data write and journal metadata write (concurrently), then journal commit, then checkpoint of metadata, then (later) mark the transaction free in the journal superblock.
e) None of the above

Log-Structured File System

You discover a disk filled with what seems to be a log-structured file system. In its initial state on disk, the first few blocks seem to be filled with an empty file system:

```
[0] checkpoint: 3 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[1] [.,0] [..,0] -- -- -- -- -- --
[2] type:dir size:1 refs:2 ptrs: 1 -- -- -- -- -- -- --
[3] chunk(imap): 2 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
```

As you examine these first few blocks, you realize their structure. The first block (Block 0) is the “checkpoint region” - it points to pieces of the inode map. Block 3 holds that first chunk of the inode map; in this case, you figure out that only inode 0 is live, and it lives in Block 2. Block 2 holds the inode of the root directory, which contains 1 block, which is located in Block 1 (the first ptr in the list of addresses held in the inode). Finally, the contents of the root directory are in Block 1: a . and .. entry, each referring to the root inode 0. Good job!

Now, for some questions. You perform a single file system operation, with the resulting on-disk state:

```
[0] checkpoint: 7 -- -- -- -- -- -- 8 -- -- -- -- -- -- -- --
[1] [.,0] [..,0] -- -- -- -- -- --
[2] type:dir size:1 refs:2 ptrs: 1 -- -- -- -- -- -- --
[3] chunk(imap): 2 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[4] [.,0] [..,0] [ku3,122] -- -- -- -- --
[5] type:dir size:1 refs:2 ptrs: 4 -- -- -- -- -- -- --
[6] type:reg size:0 refs:1 ptrs: -- -- -- -- -- -- -- --
[7] chunk(imap): 5 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[8] chunk(imap): -- -- -- -- -- -- -- -- -- -- 6 -- -- -- -- --
```

Question 51: What file operation was performed?

a) mkdir(“ku3”);
b) creat(“ku3”);
c) rmdir(“/“);
d) link(“.”, “ku3”);
e) None of the above

Question 52: In the above file system state, which of the blocks are live?
a) 0 through 8
b) 4 through 8
c) 0 through 3
d) 0 and 4 through 8
e) None

Question 53: In the above file system state, which inodes are allocated?
a) 7 and 8
b) 1, 2, and 4
c) 0 and 122
d) 5 and 6
e) None of the above

Log-Structured File System (2)

You reset the file system, and then run **TWO** operations. The contents of the entire file system (as a result) are as follows:

```
[0]  checkpoint: 7 -- -- -- 11 -- -- -- -- -- -- -- -- -- -- --
[1]  [.,0] [..,0] -- -- -- -- -- --
[2]  type:dir size:1 refs:2 ptrs: 1 -- -- -- -- -- -- --
[3]  chunk(imap): 2 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[4]  [.,0] [..,0] [ab3,72] -- -- -- -- --
[5]  type:dir size:1 refs:2 ptrs: 4 -- -- -- -- -- -- --
[6]  type:reg size:0 refs:1 ptrs: -- -- -- -- -- -- -- --
[7]  chunk(imap): 5 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[8]  chunk(imap): -- -- -- -- -- -- -- -- 6 -- -- -- -- -- -- --
[9]  t0t0t0t0t0t0t0t0t0t0t0t0t0t0t0t0
[10] type:reg size:8 refs:1 ptrs: -- -- -- -- -- -- -- 9
[11] chunk(imap): -- -- -- -- -- -- -- -- 10 -- -- -- -- -- -- --
```

Question 54: What file operations were performed to get to this new state?

a) A file read
b) A file link
c) A file create
d) A file create and then write
e) A file unlink

Question 55: In the above file system state, which of the blocks are live?
a) 0 through 11
b) 4 through 11
c) 4 through 11 (except 6 and 8)
d) 4 through 11 (except 6, 8, and 10)
e) 7 through 11

Question 56: In the above file system state, how many versions of the root inode exist (live or dead)?
a) 0
b) 1
c) 2
d) 3
e) 4 or more

Question 57: In the above file system state, how many live chunks of the imap are there?
a) 0
b) 1
c) 2
d) 3
e) 4 or more

Log-Structured File System (3)

You once again delete the entire file system, and recreate it from scratch. After running a few operations, you see the following on-disk state:

```
[ 0] checkpoint: 20 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[ 1] [.,0] [..,0] -- -- -- -- -- --
[ 2] type:dir size:1 refs:2 ptrs: 1 -- -- -- -- -- -- --
[ 3] chunk(imap): 2 -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
[ 4] [.,0] [..,0] [ym6,1] -- -- -- -- --
[ 5] type:dir size:1 refs:2 ptrs: 4 -- -- -- -- -- -- --
[ 6] type:reg size:0 refs:1 ptrs: -- -- -- -- -- -- -- --
[ 7] chunk(imap): 5 6 -- -- -- -- -- -- -- -- -- -- -- -- -- --
[ 8] [.,0] [..,0] [ym6,1] [hs2,2] -- -- -- --
[ 9] type:dir size:1 refs:2 ptrs: 8 -- -- -- -- -- -- --
[10] live type:reg size:0 refs:1 ptrs: -- -- -- -- -- -- -- --
[11] chunk(imap): 9 6 10 -- -- -- -- -- -- -- -- -- -- -- -- --
[12] [.,0] [..,0] [ym6,1] [hs2,2] [cu3,1] -- -- --
[13] type:dir size:1 refs:2 ptrs: 12 -- -- -- -- -- -- --
[14] type:reg size:0 refs:2 ptrs: -- -- -- -- -- -- -- --
[15] chunk(imap): 13 14 10 -- -- -- -- -- -- -- -- -- -- -- -- --
[16] f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0
[17] l1l1l1l1l1l1l1l1l1l1l1l1l1l1l1l1
[18] h2h2h2h2h2h2h2h2h2h2h2h2h2h2h2h2
[19] type:reg size:8 refs:2 ptrs: -- -- -- -- -- 16 17 18
[20] chunk(imap): 13 19 10 -- -- -- -- -- -- -- -- -- -- -- -- --
```

Question 58: The last five blocks of the log (16-20) represent the results of which operation?
a) A directory creation
b) A file creation
c) A multi-block write
d) A file link
e) None of the above

Question 59: The blocks 12-15 represent the results of which operation?
a) A directory creation
b) A file creation
c) A multi-block write
d) A file link
e) None of the above

Question 60: If you read the contents of the first block (offset=0) of file "/ym6", you would get:
a) `f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0f0`
b) `l1l1l1l1l1l1l1l1l1l1l1l1l1l1l1l1`
c) `h2h2h2h2h2h2h2h2h2h2h2h2h2h2h2h2`
d) All of the above
e) None of the above

Flash-based SSDs

Assume you have a tool that can examine the contents of a raw flash chip within an **FTL-based SSD**.
You see the following flash page contents, on physical pages 0 … 7.

[0] `aaaa`
[1] `bbbb`
[2] `cccc`
[3] `dddd`
[4] `eeee`
[5] `ffff`
[6] `gggg`
[7] `hhhh`

The first four page (0…3) are on **Block 0** of the flash, and pages 4…7 are on **Block 1**. Your job is to use your forensics skills to answer the following questions.

Question 61: Assume the following **page-mapped** FTL: 1000:0, 1001:1, 1002:2 (X:Y means logical address X maps to physical page Y). What is data is returned if the user of the SSD issues a read to address=1002?
a) `aaaa`
b) `bbbb`
c) `cccc`
d) `dddd`
e) None of the above

Question 62: Assume the following **page-mapped** FTL: 100:7, 101:6, 102:5, 103:4. What is data is returned if the user of the SSD issues a read to address=102?
a) `aaaa`
b) `bbbb`
c) `cccc`
d) `dddd`
e) None of the above

Question 63: Assume a **page-mapped** FTL. The user issues a read which returns "`ffff`". Which of the following could **NOT** accurately represent the contents of the **entire FTL**?
a) 1000:1, 1001:3, 1002:5, 1003:7
b) 1000:1, 1001:3, 1002:5, 1003:7, 1004:0
c) 1000:1, 1001:3, 1002:5, 1003:7, 1004:0, 1005:2
d) 1000:1, 1001:3, 1002:5, 1003:7, 1004:0, 1005:2, 1006:4
e) None of the above

Question 64: Assume a **page-mapped** FTL. The user issues a read which returns "`aaaa`". Which of the following could **NOT** accurately represent the contents of the **entire FTL**?
a) 1000:1, 1001:3, 1002:5, 1003:7
b) 1000:1, 1001:3, 1002:5, 1003:7, 1004:0
c) 1000:1, 1001:3, 1002:5, 1003:7, 1004:0, 1005:2
d) 1000:1, 1001:3, 1002:5, 1003:7, 1004:0, 1005:2, 1006:4
e) None of the above

Flash-based SSDs (2)

Assume the same flash contents, now showing Blocks 0, 1, and part of Block 2:
[0] `aaaa`
[1] `bbbb`
[2] `cccc`
[3] `dddd`
[4] `eeee`
[5] `ffff`
[6] `gggg`
[7] `hhhh`
[8] `erased (not yet programmed)`
[9] `erased (not yet programmed)`
[10] `etc.`

For the next questions, assume the following **page-mapped FTL** is the **initial state** of the system**:**
10:0, 11:1, 12:2, 13:3, 14:4, 15:5, 16:6, 17:7

Question 65: Assume the user then issues a write to the SSD(address=14, data=`iiii`). What will the contents of flash physical page 4 be just **after** this write takes place?
a) `dddd`
b) `eeee`
c) `ffff`
d) `iiii`
e) None of the above

Question 66: Assuming Block 2 (pages 8…11) of the flash is used for the write above to address=14, which of the following could represent the contents of the FTL after the write completes?
a) 10:0, 11:1, 12:2, 13:3, 14:4, 15:5, 16:6, 17:7
b) 10:0, 11:1, 12:2, 13:3, 14:2, 15:5, 16:6, 17:7
c) 10:0, 11:1, 12:2, 13:3, 14:8, 15:5, 16:6, 17:7
d) 10:0, 11:1, 12:2, 13:3, 15:5, 16:6, 17:7
e) None of the above

Question 67: After the write (to address=14, data=`iiii`) is complete, the SSD decides to perform **garbage collection** (**GC**). Which of the following will **NOT** happen as part of the GC?
a) Block 0 is erased
b) Block 1 is erased
c) Data from pages 5, 6, 7 are copied elsewhere
d) The contents of the FTL (the mappings) will change
e) Block 2 is erased

Flash-based SSDs (2)

One problem with page-mapped FTLs is that they make the mapping tables too large. The last few questions around SSDs deal with this issue.

Question 68: Assume you have a page-mapped FTL. If each entry in the FTL takes **4 bytes** (assuming it is an array), how large is the FTL? Assume the SSD is only **1 MB** in size, and uses **1 KB** pages.
a) 1 KB
b) 4 KB
c) 1 MB
d) 4 MB
e) None of the above

Question 69: Assume the same system as in Question 68, but now with block mappings (not page). Assume each block fits 4 pages. Assuming each entry is still 4 bytes, how large is this block-mapped FTL?
a) 1 KB
b) 4 KB
c) 1 MB
d) 4 MB
e) None of the above

And now, a final, general question for you about SSDs.

Question 70: Which of the following is **NOT** true about flash-based SSDs?
a) SSDs need less memory to perform logical to physical translation than hard drives
b) SSDs are generally faster than hard drives for write-oriented workloads
c) SSDs are generally more expensive than hard drives (cost per byte)
d) SSDs are generally faster than hard drives for read-oriented workloads
e) SSDs use fewer moving parts than hard drives

Virtual Memory

You are given a system with 64 bytes of physical memory, 4 byte pages, and 16-byte virtual address spaces. Your forensics tools dig up the following page table structure (high bit: Valid/NOT, rest is the PFN):

```
[0] 0x00000000
[1] 0x800000?? <- missing!
[2] 0x00000000
[3] 0x00000000
```

Question 71: A trace you have accesses virtual address 0x7, which translates to 0x33. What two hex digits are missing from page table entry 1 above?
a) 0x0a
b) 0x0b
c) 0x0c
d) 0x0d
e) None of the above

Assuming the same system (4 byte pages, 16 byte virtual address space), and the following page table:

```
[0] 0x00000000
[1] 0x80000005
[2] 0x8000000a
[3] 0x00000000
```

Question 72: Given the virtual address 4, which decimal physical address does it translate to?
a) 5
b) 10
c) 20
d) 54
e) 45

Assuming the following page table (on the same system, again):

```
[0] 0x00000000
[1] 0x8000000e
[2] 0x00000000
[3] 0x80000009
```

Question 73: Which ranges of virtual address are valid?
a) 0 … 64
b) 0 … 16
c) 4 … 7 and 12 … 15
d) 0 … 3 and 8 … 11
e) 80 … 89

Virtual Memory (2)

You now are given some statistics about various systems, and need to do some basic calculations before continuing.

Question 74: You are told a given system has a **30-bit virtual address**, with a **4KB page size**. Assuming a **4-byte** page table entry size, how big is a linear page table for a given process?
a) 1 MB
b) 2 MB
c) 4 MB
d) 8 MB
e) None of the above

Question 75: You are next given system has a **10-bit virtual address**. with a **256 byte page size**. Assuming a **4-byte** page table entry size, how big is a linear page table for a given process?
a) 16 bytes
b) 16 KB
c) 16 MB
d) 16 GB
e) None of the above

Question 76: You are now given some new information about a particular system. Specifically, this system has **1 MB linear page table size** (per process), and has a **1KB page size**. Assuming page table entry size is **4 bytes**, how many bits are in the virtual page number (**VPN**) on this system?
a) 28
b) 18
c) 8
d) 32
e) None of the above

Lastly, you are given some memory dumps, which tell you something about the contents of memory. In this dump, you find that the system only has **3 pages**, that pages **3, 4, and 5 are in memory**. You also check a history log and find that the last 10 pages accessed were 8, 7, 4, 2, 5, 4, 7, 3, 4, 5 (in that order, with 5 being most recently accessed).

Question 77: You are then asked to determine which replacement policy was used. Is it:
a) FIFO
b) LRU
c) MRU
d) LFU
e) Cannot tell from the given information

Virtual Memory (3)

With the same situation as above (3-page system, with 3, 4, 5 in memory, and the last 10 pages accessed are 8, 7, 4, 2, 5, 4, 7, 3, 4, 5), you are asked the following few questions.

Question 78: Assuming the replacement policy was **FIFO**, how many **misses** were encountered while those last 10 pages were accessed? (assume the memory was empty to begin)
a) 7
b) 8
c) 9
d) 10
e) None of the above

Question 79: Assuming the replacement policy was **LRU**, how many **misses** were encountered while those last 10 pages were accessed? (assume the memory was empty to begin)
a) 7
b) 8
c) 9
d) 10
e) None of the above

Question 80: Assuming the replacement policy was **OPT** (the optimal replacement policy), how many **misses** were encountered while those last 10 pages were accessed? (assume the memory was empty to begin)
a) 7
b) 8
c) 9
d) 10
e) None of the above