Name: _____   Entry number:_____

Instructions
- Verify that your exam has pages 1 to 10. Please fill the name and entry number on *every page* front and back. Do this first before starting the exam.
- During the exam, you should not have any electronic equipment including laptop, mobile phone, calculator, tablets, etc.
- The exam is open notes, open book, open slides. You can keep the printouts with you during the exam.
- Please answer each question in the provided space. Rough space is available on Page 5.

Please sign below:

I will not give or receive aid in the examination. I accept that any act of mine that can be considered to be an *IITD Honour Code* violation will invite disciplinary action.

Signature: _____

Q1. [4 marks] Assembly program
Let us say that the following recursive function foo is called with arguments 9, 12.

```
foo:
        movl    4(%esp), %eax
        movl    8(%esp), %edx
        cmpl    %edx, %eax
        jne     .L3
        ret
.L11:
        subl    %edx, %eax
        pushl   %edx
        pushl   %eax
        call foo
        ret
.L3:
        cmpl    %eax, %edx
        jl      .L11
        subl    %eax, %edx
        pushl   %edx
        pushl   %eax
        call foo
        ret
```

> *Q1.1 [4 marks]* What will be the return value (value in register %eax) when the function returns?
>
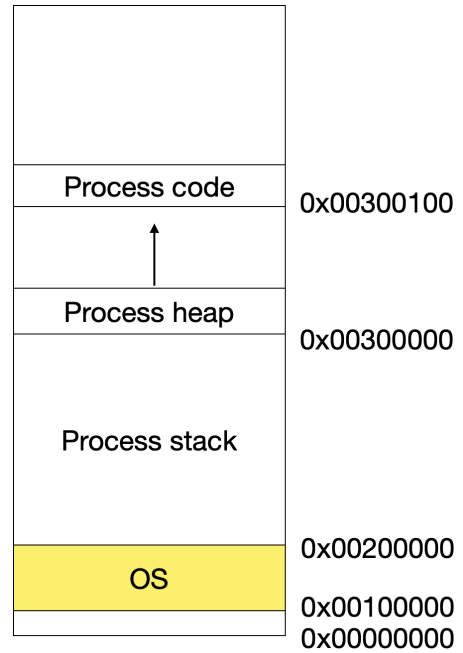> Hint: Draw the stack. Track %eip, %esp, %eax, %edx as the program executes.

Q2. [7 marks] Memory allocator

Let us say that we are running the following program:

```
void* malloc(size_t s) {
  return sbrk(s);
}
void free(void *p){ }

int main(int argc, const char *argv[]) {
  int iter = atoi(argv[1]);
  for(int i = 0; i < iter; i ++) {
    int* ptr = (int*) malloc(sizeof(int));
    *ptr = iter;
    free(ptr);
  }
  return 0;
}
```

Process code	0x00300100
↑ Process heap	0x00300000
Process stack	
	0x00200000
OS	0x00100000
	0x00000000

Notice that the program is using a custom virtual memory allocator. Assume that the OS is using segmentation hardware to do memory isolation. It has set up the segments for this process as shown in the figure. Initially, the base of the process' heap (data segment) is 0x00300000 and the limit is zero.

Q2.1 [2 marks] For the given layout, what is the smallest positive value of `iter` for which the program will crash? Assume that the OS never relocates segments elsewhere. OS will return a null pointer from sbrk when the segment can no longer grow.

Q2.2 [2 marks] The user observes that the program was working fine for `iter=100`. But, when they run the same program again with `iter=100`, the program crashes. What might be the reason for the crash? Assume the same behaviour of the OS as in the above question.

Q2.3 [3 marks] Please modify malloc/free implementations to fix this program such that it works for all iter values. You may assume that the OS can, regardless of other processes in the system, grow the process' heap (data segment) to a size of 0x100 bytes.

Hint: You only need to make sure that the given program does not crash. You need not make arbitrary programs work. You are not allowed to modify the main function.

Q3. *[5 marks]* Hard disk drives

Let us say we are using a RAID-5 setup with 8 identical HDDs. The HDD mentions the following specifications in its datasheet:

Capacity: 1TB; Max. Sustained Transfer Rate (MB/s): 200MB/s; Bytes per sector: 4096; Rotational speed (RPM): 6,000; Average seek delay: 5ms

Q3.1 [1 mark] How many disk failures can the setup tolerate before we lose data?

Q3.2 [1 mark] What is the capacity of the setup?

Q3.3 [1 mark] What sequential read throughput can we expect?

Q3.4 [1 mark] What random read throughput can we expect?

Q3.5 [1 mark] What random write throughput can we expect?

Q4. [7 marks] Metadata journaling

Let us say our disk has 512 byte blocks. There is a file which is spread over three data blocks (file size = 1536 bytes). The file contains 1536 'A's. A program opens the file, seeks to the start of the file, makes one write operation to write 1536 'B's, and closes the file. All the working of the program is in a single file system transaction. After the program ends, the computer abruptly restarts. When the file system becomes available, the user sees that the file has 512 'A's, then 512 'B's, and then 512 'A's.

Q4.1 [4 mark] Precisely explain the sequence of events which led to this outcome. The file system was using metadata journaling.

Q4.2 [3 mark] Can the same outcome be observed if the file system was instead using full journaling, i.e, logging both the data and the metadata blocks? Justify your answer.

Name: _____   Entry number:_____

Q5 [3 marks]: Protection

Let us say that the value of the code segment (cs) register is 0x001B and the data segment (ds) register is 0x0012.

Q5.1 [1 mark] What is the current privilege level?
Q5.2 [1 mark] Which index of the GDT is going to be used to do address translation for eip?
Q5.3 [1 mark] What value should the descriptor privilege level be set to, in the segment descriptor, such that it can be referred by the program?

This area is intentionally left blank. You can use it for rough work.

Q6 [24 marks]. Snapshotting

In this problem, we are interested in extending the indexed file system studied in class to support *hourly snapshotting*. A snapshot is a consistent state of the file system. Snapshots can be used to recover old versions of files and folders. For example, let us say a user accidentally deleted a file named `/exam.md` and wants to recover it. The following example shows how to list all the versions of `/exam.md` saved in snapshots.
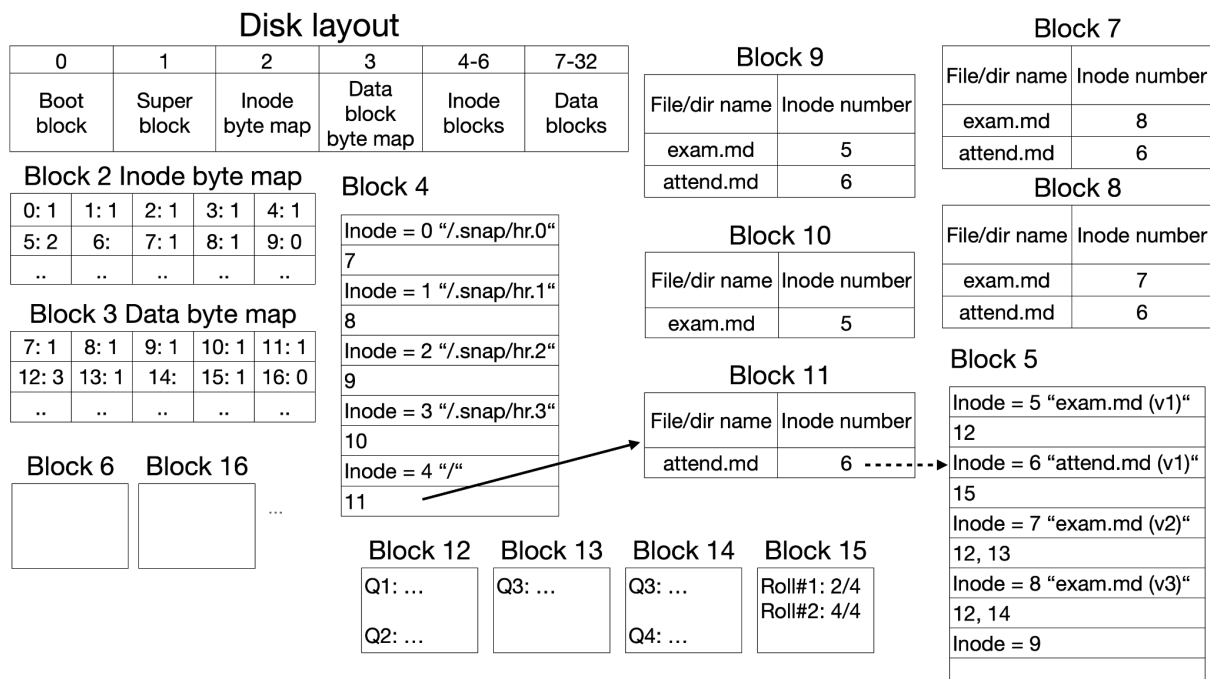
```
$ ls -lh /.snap/*/exam.md
-rw-r--r--@ 1 user  group  1002 Feb 24 11:00 /.snap/hr.0/exam.md
-rw-r--r--@ 1 user  group   958 Feb 24 10:00 /.snap/hr.1/exam.md
-rw-r--r--@ 1 user  group   512 Feb 24 09:00 /.snap/hr.2/exam.md
-rw-r--r--@ 1 user  group   512 Feb 24 08:00 /.snap/hr.3/exam.md
```

The user can recover the most recent version by copying it:
```
$ cp /.snap/hr.0/exam.md /
```

One naive approach to implement snapshotting would be to copy the entire file system every hour. But this approach will be very slow. It also wastes disk space: if most files are not changed across snapshots, we are unnecessarily creating copies of it.
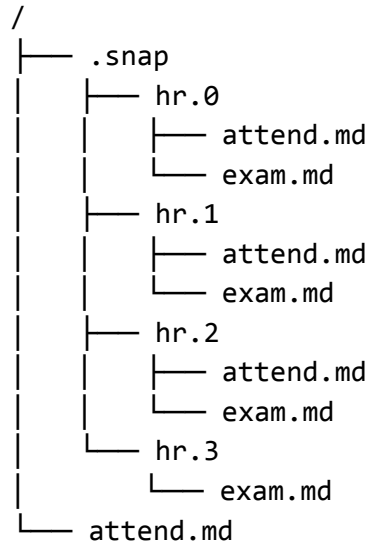
Therefore, we try to use a *copy-on-write* approach to share blocks across the current file system and the snapshots. The following shows the state of our file system with four hourly snapshots. We assume that each inode block contains 5 inodes each. Inodes 0, 1, 2, 3, and 4 are in block 4. Inodes 0, 1, 2, and 3 contain hourly snapshots and inode 4 contains the current file system.

Disk layout

0	1	2	3	4-6	7-32
Boot block	Super block	Inode byte map	Data block byte map	Inode blocks	Data blocks

Block 2 Inode byte map

0: 1	1: 1	2: 1	3: 1	4: 1
5: 2	6:	7: 1	8: 1	9: 0
..	..	..	..	..

Block 3 Data byte map

7: 1	8: 1	9: 1	10: 1	11: 1
12: 3	13: 1	14:	15: 1	16: 0
..	..	..	..	..

Block 6   **Block 16**   ...

Block 4

Inode = 0 "/.snap/hr.0"
7
Inode = 1 "/.snap/hr.1"
8
Inode = 2 "/.snap/hr.2"
9
Inode = 3 "/.snap/hr.3"
10
Inode = 4 "/"
11

Block 12

Q1: …

Q2: …

Block 13

Q3: …

Block 14

Q3: …

Q4: …

Block 15

Roll#1: 2/4
Roll#2: 4/4

Block 9

File/dir name	Inode number
exam.md	5
attend.md	6

Block 10

File/dir name	Inode number
exam.md	5

Block 11

File/dir name	Inode number
attend.md	6 ------

Block 7

File/dir name	Inode number
exam.md	8
attend.md	6

Block 8

File/dir name	Inode number
exam.md	7
attend.md	6

Block 5

Inode = 5 "exam.md (v1)"
12
Inode = 6 "attend.md (v1)"
15
Inode = 7 "exam.md (v2)"
12, 13
Inode = 8 "exam.md (v3)"
12, 14
Inode = 9

Name: _____ Entry number:_____

> *Q6.1 [2 marks]*: Draw solid arrows from inodes to data blocks and dotted arrows from directory entries to inodes. Two examples are already drawn in the figure.

The file system described above is as follows. The (v1), (v2), etc. annotations in block 5 are only shown for clarity.

```
/
├── .snap
│   ├── hr.0
│   │   ├── attend.md
│   │   └── exam.md
│   ├── hr.1
│   │   ├── attend.md
│   │   └── exam.md
│   ├── hr.2
│   │   ├── attend.md
│   │   └── exam.md
│   └── hr.3
│       └── exam.md
└── attend.md
```

Byte maps to track reference counts: In the indexed file system, we kept inode (data block) bitmaps where we stored 0 if the inode (data block) is free. We modify this slightly for our snapshotting file system. The inode byte map (block 2) and the data block byte map (block 3) now store 1 byte for each inode and data block respectively. For each inode (data block), inode byte map (data block byte map) stores the number of incoming arrows to the inode (data block).

> *Q6.2 [1 mark]*: What will be the byte map values for inode 6 and for data block 14 in the file system shown above.
>
>
> *Q6.3 [1 mark]: How many inode byte map blocks are required if this file system had 1000 inode blocks.*
>
>

Deduplication: Notice that this file system design is de-duplicating inodes and data blocks across snapshots. For example, `/.snap/hr.2/exam.md` and `/.snap/hr.3/exam.md` were identical. Therefore, their directory entries for `exam.md` point to the same inode number 5. Similarly, the first data block of `exam.md` was the same for all versions. Therefore, all `exam.md` inodes have the same first data block as 12.

> *Q6.4 [1 mark]:* `/.snap/hr.1/exam.md` has more contents than `/.snap/hr.2/exam.md`.
>
> *[True/False]* _____

Q6.5 [1 mark]: `attend.md` has not changed in the last 2 hours.

[True/False] _____

Q6.6 [4 marks]: With deduplication, we are able to store 4 snapshots and the current file system in 2 inode blocks (4, 5) and 9 data blocks (7-15). Calculate how many inode blocks and how many data blocks would be required for storing the same information if we were simply copying the file system for creating snapshots i.e, without any deduplication.

Hint: Redraw the disk blocks when there is no deduplication.

Copy on write.
Now, whenever we are modifying an inode (or a data block), we first check if the corresponding byte map value is greater than 1. If it is, that means the inode (data block) is also being referenced from earlier snapshots. In such cases, we instead copy the inode (data block). For example, at hr.0, the user appended "Q4" to the exam.md file. At that point, instead of directly changing inode 7 and data block 13, the file system created a new inode 8 and data block 14.

Q6.7 [3 marks]: Let us say the user wants to update the attendance of Roll#1 to 4/4 from 2/4 in attend.md. Redraw only the changed blocks below.

Snapshotting. At each hour, we want to snapshot the current file system state. In particular,

- /.snap/hr.0 should contain what / contained before the snapshot,
- /.snap/hr.1 should contain what /.snap/hr.0 contained before the snapshot,
- /.snap/hr.2 should contain what /.snap/hr.1 contained before the snapshot,
- /.snap/hr.3 should contain what /.snap/hr.2 contained before the snapshot.

Q6.8 [3 marks]: Modify and redraw the state of the disk blocks after a new snapshot has been created. Try to change the least number of blocks.

Crash consistency. Let us say that we "recover" the file system after a crash by simply reverting to the last snapshot. Therefore, we are not doing any kind of write ahead logging.

Q6.9 [3 marks]: Describe an ordering based approach to create a new snapshot which also cleans up the oldest snapshot. For simplicity, assume we are blocking all other file system operations while we are creating the snapshot. Argue why this ordering cannot lead to dangling pointers.

Q6.10 [3 marks]: Is creating a snapshot a fast or a slow operation? Justify your answer.

Q6.11 [2 marks]: Contrast this file system with the ext3 file system studied in the class. What are the pros and cons?