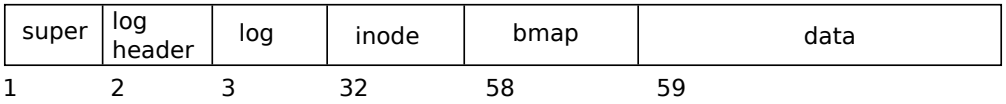
Principles of Operating Systems **Name (Print):** _____
Fall 2017
Final
12/13/2017
Time Limit: 8:00am - 10:00am

- **Don't forget to write your name on this exam.**
- **This is an open book, open notes exam. But no online or in-class chatting.**
- **Ask me if something is not clear in the questions.**
- **Organize your work**, in a reasonably neat and coherent way, in the space provided. Work scattered all over the page without a clear ordering will receive very little credit.
- **Mysterious or unsupported answers will not receive full credit**. A correct answer, unsupported by explanation will receive no credit; an incorrect answer supported by substantially correct explanations might still receive partial credit.
- If you need more space, use the back of the pages; clearly indicate when you have done this.

Problem	Points	Score
1	20	
2	5	
3	10	
4	15	
5	20	
6	15	
7	5	
Total:	90	

1. File system

   Xv6 lays out the file system on disk as follows:



   Block 1 contains the super block. Blocks 2 through 31 contain the log header and the log. Blocks 32 through 57 contain inodes. Block 58 contains the bitmap of free blocks. Blocks 59 through the end of the disk contain data blocks.

   Ben modifies the function bwrite in bio.c to print the block number of each block written.

   Ben boots xv6 with a fresh fs.img and types in the command rm README, which deletes the README file. This command produces the following trace:

```
$ rm README
write 3
write 4
write 5
write 2
write 59
write 32
write 58
write 2
$
```

   (a) (5 points) Briefly explain what block 59 contains in the above trace. Why is it written?

(b) (5 points) What does block 5 contain? Why is it written?

(c) (10 points) How many non-zero bytes are written to block 2 when it's written the first time and what are the bytes? (To get the full credit you have to explain what block 2 contains, and why each non-zero byte is written).

2. Synchronization

   (a) (5 points) Ben runs xv6 on a single CPU machine, he decides it's a good idea to get rid of the acquire() and release() functions, since after all they take some time but seem unnecessary in a single-CPU scenario. Explain if removal of these functions is fine.

3. Process memory layout

   Bob decides to mess up his friend Jack's xv6 environment by typing this in hello.c

```
#define PGSIZE  4096
int main(int argc, char *argv[]) {
  char buf[PGSIZE] = {0};
  printf(1, "Hello World!, %p\n", buf);
  exit();
}
```

   (a) (10 points) Jack types hello and encounters a trap message such as this.

```
pid 3 hello: trap 14 err 7 on cpu 1 eip 0x22 addr 0x1fd0--kill proc
```

   Can you help Jack understand the problem? (To receive full points, you should explain why this error happened, what does the values signify and how to solve this)

4. Virtual memory

Ben wants to know the address of physical pages that back up virtual memory of his process. He digs into the kernel source and comes across the V2P() macro all over the kernel.

```
#define KERNBASE  0x80000000
#define V2P(a) (((uint) (a)) - KERNBASE)
```

He decides to try the V2P macro in his program like in the example below but encounters a program crash.

```
int main(int argc, char *argv[]) {
  int a;
  *(uint*)V2P(&a) = 0xaddb;
  printf(1, "I changed physical memory %x\n", a);
  exit();
}
```

(a) (5 points) Explain what is going on and why Ben's program crashes.

(b) (10 points) If Ben puts this code inside a new system call that would allow him to run it in the kernel, will it work?

5. Demand paging

   Ben wants to extend xv6 with demand paging. Ben observes that some pages of that are allocated for user processes (heap, text, and stack) are not accessed that frequently, yet anyway they consume valuable physical memory. So Ben comes up with a plan to save content of an idle page to disk (swap a page out), unmap it from the user address space, and free the physical page back to the memory allocator, making it accessible for other processes or the kernel. Obviously, Ben wants paging to be transparent. I.e., when a process accesses one of the swapped pages, Ben wants to catch an exception, allocate a new physical page, read old content of the page from disk and fix the process page table in such a way that process can access it like nothing happened.

   (a) (5 points) What changes to page tables are required to catch an exception when the process tries to access one of the swapped pages (hint: look at how guard page is implemented)?

(b) (15 points) Ben plans to catch the exception caused by an unmapped page access from the trap() function. Provide a sketch of the exception code below.

6. Process creation

While editing the xv6 code, Jimmy accidentally erases the below section of code under fork() function on proc.c

```
2584 for(i = 0; i < NOFILE; i++)
2585    if(proc->ofile[i])
2586        np->ofile[i] = filedup(proc->ofile[i]);
```

(a) (5 points) Explain what the above section of code does?

(b) (10 points) Explain where things can go wrong without this code. Quote a concrete example.

7. ics143A. I would like to hear your opinions about 6.828, so please answer the following questions. (Any answer, except no answer, will receive full credit.)

   (a) (1 point) Grade ics143A on a scale of 0 (worst) to 10 (best)?

   (b) (2 points) Any suggestions for how to improve ics143A?

   (c) (1 point) What is the best aspect of ics143A?

   (d) (1 point) What is the worst aspect of ics143A?