ICS 143A
Fall 2017
Midterm
11/15/2017
Time Limit: 9:00am - 9:50am

Name (Print): ______________________

- **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 us if you something is confusing 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	5	
2	10	
3	20	
4	20	
Total:	55	

1. Basic page tables.

    (a) (5 points) Illustrate organization of the x86, 4K, 32bit 2-level page tables through an simple example. Assume that the hardware translates the virtual address '0xc04005' (binary 0b1100 0000 0100 0000 0000 0101) into the physical address '0x55005'. The physical addresses of the page table directory and the page table (Level 2) involved in the translation of this virtual address are 0x8000 and 0x2000. Draw a diagram, provide a short explanation.

2. Xv6 shell implements a pipe command (e.g., ls — wc) as follows:

```
8650 case PIPE:
8651 pcmd = (struct pipecmd*)cmd;
8652 if(pipe(p) < 0)
8653   panic("pipe");
8654 if(fork1() == 0){
8655   close(1);
8656   dup(p[1]);
8657   close(p[0]);
8658   close(p[1]);
8659   runcmd(pcmd>left);
8660 }
8661 if(fork1() == 0){
8662   close(0);
8663   dup(p[0]);
8664   close(p[0]);
8665   close(p[1]);
8666   runcmd(pcmd>right);
8667 }
8668 close(p[0]);
8669 close(p[1]);
8670 wait();
8671 wait();
8672 break;
```

(a) (5 points) Why does the child process that runs the left-side of the pipe close file descriptor 1 and why does the child process that runs the right-side of the pipe close file descriptor 0?

(b) (5 points) It looks that in the sh.c code above after the first fork() (at line 8654) both parent and child will reach the second fork() (line 8661) creating two child processes. Both child processes will start reading from the pipe and will try to execute the right side of the pipe. This seems wrong. Can you explain what is happening?

3. OS isolation and protection

   (a) (5 points) In xv6 user processes cannot access kernel memory. Explain why.

   (b) (5 points) Imagine you have hardware that is identical to x86, but does not have a user bit in the page tables. What changes need to be made to xv6 to ensure isolation of the kernel from user-processes?

   (c) (10 points) Bob is very new to xv6, he writes hist first program:

```
#include "types.h"
#include "user.h"

int
main(int argc, char *argv[])
{

  char buf[512], *p = 0;

  memmove(p, buf, sizeof(buf));
  exit();
}
```

   When he runs it however, he gets the following error

```
$ copy
pid 3 copy: trap 14 err 5 on cpu 1 eip 0x0 addr 0x11f0--kill proc
$
```

   Explain this error and what goes wrong.

4. OS organization. Imagine you want to optimize xv6 to run a large number of very small processes. A realistic example can be a web server that implements a Facebook's login page—you have to isolate each user in its own process, otherwise a single exploit from any user reveals accounts of all users going through the login page, but at the same time each process is very small (it just sends out a simple HTML page). Entire logic of such web server program can fit in 2-3K of memory.

   (a) (10 points) You start by analizing the overheads involved into creating a process. How many pages of memory are allocated when xv6 creates a smallest process? Count both user-level and kernel resources.

   (b) (10 points) Suggest a set of changes to xv6 aimed at minimizing the number of pages that are required for creating very small processes, e.g., the once that are 1K in size.