Principles of Operating Systems	**Name (Print):**	
Fall 2019	**Seat:**	SEAT
Final	**Left person:**	
12/13/2019 **Time Limit: 8:00am – 10:00pm**	**Right person:**	

- **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 something is confusing.**
- **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.
- **Don't forget to write your name on this exam.**

Problem	Points	Score
1	10	
2	15	
3	15	
4	15	
5	17	
6	15	
7	4	
Total:	91	

1. Operating system interface

   (a) (10 points) Write code for a simple program that implements the following pipeline:

   `cat main.c | grep "main" | wc`

   I.e., you program should start several new processes. One for the `cat main.c` command, one for `grep main`, and one for `wc`. These processes should be connected with pipes that `cat main.c` redirects its output into the `grep "main"` program, which itself redirects its output to the `wc`.

   ```
   forked pid:811
   forked pid:812
   fork failed, pid:-1
   ```

2. Processes and system calls

Alice is implementing a fork bomb, i.e., she tries to create as many processes in xv6 as possible.

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

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

  for(;;) {
        pid = fork();
        if(pid == -1) {
                printf(1, "fork failed, pid:%d\n", pid);
                exit();
        } else if (pid) {
                printf(1, "forked pid:%d\n", pid);
        } else {
                for (;;) {
                        sleep(1);
                }
        };
  }
  exit();
}
```

(a) (5 points) She boots into xv6 and right away starts her program `forkbomb` in shell. She sees the following output:

```
$ forkbomb
forked pid:4
forked pid:5
forked pid:6
...
forked pid:61
forked pid:62
forked pid:63
forked pid:64
fork failed, pid:-1
```

This means that her program forked 61 times. She realizes that xv6 kernel has an array of proc data structures of size 64, but still she is confused: why did she fork only 61 times? Please explain why.

(b) (10 points) Alice quickly changes the size of the array to 4096, reboots, and runs her program again. How many times she will be able to fork now? Explain your reasoning.

3. Context switch

   The `swtch()` function that implements the core of the context switch saves only 4 registers on the stack

```
.globl swtch
swtch:
  movl 4(%esp), %eax
  movl 8(%esp), %edx

  # Save old callee-saved registers
  pushl %ebp
  pushl %ebx
  pushl %esi
  pushl %edi

  # Switch stacks
  movl %esp, (%eax)
  movl %edx, %esp

  # Load new callee-saved registers
  popl %edi
  popl %esi
  popl %ebx
  popl %ebp
  ret
```

   The context data structure has 5 registers:

```
struct context {
  uint edi;
  uint esi;
  uint ebx;
  uint ebp;
  uint eip;
};
```

   (a) (3 points) How does the EIP register gets saved and restored?

(b) (3 points) How does the kernel EAX register gets saved and restored?

(c) (3 points) How does user-level EAX register gets saved and restored?

(d) (3 points) How does the kernel ESP register gets saved and restored?

(e) (3 points) How does user-level ESP register gets saved and restored?

4. System calls

   (a) (10 points) What does the user stack looks like when the `read()` system call is invoked, i.e., when the execution is already in kernel and it reaches the `sys_read()` function. Draw a diagram, provide a short description for every value on the stack. Remember the `read()` system call has the following signature:

   `int read(int, void*, int);`

   (b) (5 points) If the execution is inside a system call, e.g., inside the `sys_read()` function, and we count from the bottom of the kernel stack (here the top of the stack is pointed by the ESP register, and bottom is the end of the kernel stack page), bytes 0-3 from the bottom contain the `ss` (stack segment of the user program when it entered the kernel with the system call), bytes 4-7 contiain the user ESP value, etc.. Then what do bytes 24-27 contain (explain your answer)?

5. Global Descriptor Table (GDT)

   (a) (5 points) How GDT is used in xv6, i.e., what role does it play in the system?

   (b) (5 points) How many global descriptor tables xv6 creates?

(c) (7 points) Explain lines 1870–1874 in the `switchuvm()` function (be specific).

```
1859 void
1860 switchuvm(struct proc *p)
1861 {
1862   if(p == 0)
1863     panic("switchuvm: no process");
1864   if(p->kstack == 0)
1865     panic("switchuvm: no kstack");
1866   if(p->pgdir == 0)
1867     panic("switchuvm: no pgdir");
1868
1869   pushcli();
1870   mycpu()->gdt[SEG_TSS] = SEG16(STS_T32A, &mycpu()->ts,
1871                                 sizeof(mycpu()->ts)-1, 0);
1872   mycpu()->gdt[SEG_TSS].s = 0;
1873   mycpu()->ts.ss0 = SEG_KDATA << 3;
1874   mycpu()->ts.esp0 = (uint)p->kstack + KSTACKSIZE;
1875   // setting IOPL=0 in eflags *and* iomb beyond the tss segment limit
1876   // forbids I/O instructions (e.g., inb and outb) from user space
1877   mycpu()->ts.iomb = (ushort) 0xFFFF;
1878   ltr(SEG_TSS << 3);
1879   lcr3(V2P(p->pgdir));  // switch to process's address space
1880   popcli();
1881 }
```

6. Interrupts

   (a) (5 points) Can an interrupt preempt execution of a system call, i.e., can the interrupt be delivered and processed why the system executes a system call (explain your answer)?

   (b) (5 points) Can an interrupt preempt execution of another interrupt (explain your answer)?

   (c) (5 points) Xv6 creates a kernel stack for each process. Why can't we simply create one kernel stack per physical CPU?

7. cs143A. I would like to hear your opinions about cs143A, so please answer the following questions. (Any answer, except no answer, will receive full credit.)

   (a) (1 point) What is the best aspect of cs143A?

   (b) (1 point) What is the worst aspect of cs143A?

   (c) (2 points) Any suggestions for how to improve cs143A?