Final - cs143A - Fall2020

Q1 Operating system interface

5 Points

Write a simple xv6 C program that implements tee. The tee command reads from the standard input and writes to both standard output and a file at the same time. The syntax for the tee command is as follows:

```
tee <file_name>
```

Your program does not need to be perfect C, but should use all the system calls correctly.

Q2 Interrupts and exceptions

5 Points

Alice executes the following code in her xv6 program

```
printf(1, "Hello world\n");
```

What is the first assembly instruction that is executed in the kernel (at CPL 0), explain your answer

Q3 Context switch

8 Points

During the context switch, the `switch` function saves only 4 registers, what happens to other registers? Specifically, answer the questions below:

Q3.1

3 Points

Which line of code saves the kernel value of the `esp` and what data structure holds its value when the process is not running? Explain your answer.

Q3.2

3 Points

Which line of code saves the user value of the `esp` and what data structure holds its value when the process is not running? Explain your answer.

Q3.3

1 Point

Which line of code saves the kernel value of the `eax` register and what data structure holds its value when the process is not running? Explain your answer.

Q3.4

1 Point

Which line of code saves the user value of the `eax` register and what data structure holds its value when the process is not running? Explain your answer.

Q4 Locks and synchronization

10 Points

Alice is not entirely sure about the role of the `ptable.lock` so she removes it from xv6 all together.

Q4.1

5 Points

Will she be able to run her modified xv6 on a single CPU machine? Explain your answer.

Q4.2

5 Points

Will she be able to run her modified xv6 on a machine with two CPUs? Explain your answer.

Q5 Page tables

10 Points

Consider the following 32-bit x86 page table setup.

`CR3` holds `0x0`.

The Page Directory Page at physical address `0x0` (the flags are PTE_P (present), PTE_U (user-accessible), and PTE_W (writable):

```
PDE 0: PPN=0x00001, PTE_P, PTE_U, PTE_W
PDE 1: PPN=0x00001, PTE_P, PTE_U, PTE_W
PDE 2: PPN=0x00002, PTE_P, PTE_U, PTE_W
... all other PDEs are zero
```

The Page Table Page at physical address `0x00001000` (which is PPN `0x00001`):

```
PTE 0: PPN=0x00005, PTE_P, PTE_U, PTE_W
PTE 1: PPN=0x00006, PTE_P, PTE_U, PTE_W
... all other PTEs are zero
```

The Page Table Page at physical address `0x00002000` (which is PPN `0x00002`):

```
PTE 0: PPN=0x00006, PTE_P, PTE_U, PTE_W
PTE 1: PPN=0x00005, PTE_P, PTE_U, PTE_W
... all other PTEs are zero
```

Find all virtual addresses that map to physical address `0x00005005`

Q6 Process organization
15 Points

Q6.1
5 Points

For the smallest xv6 process that simply calls the `exit()` system call right away after starting, what will be the `proc->sz` (size of the process memory). Explain your answer.

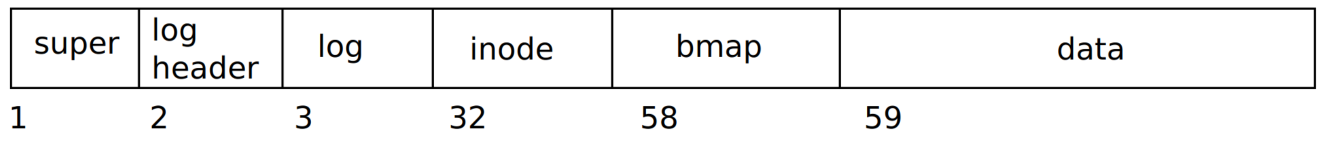
Q6.2
10 Points

You want to maximize the amount of physical memory xv6 can handle. What is the max number you can change `PHYSTOP` to without crashing xv6? Explain your answer.

Q7 File systems

15 Points

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

```
$ ln README foo
```

Which creates another name "foo" for the README file. This command produces the following trace:

```
$ ln README foo

write 3
write 4
write 2
write 32
write 59
write 2
```

Q7.1

5 Points

What is inside block 2 and why is it written twice? Explain your answer.

Q7.2

5 Points

What if the system reboots right between the writes to blocks 2 and 32 (i.e., 2 is written, but 32 is lost), will the foo be visible on disk after reboot? Explain your answer.

Q7.3

5 Points

Alice decides to check how many times she can link the same file, i.e., she runs:

```
$ ln README foo1
$ ln README foo2
$ ln README foo3
```

How many times she can link the same file before ln returns an error? Explain your answer.