238P Midterm - 2020 Spring

Q1 UNIX System Call Interface

10 Points

Write a program that uses the UNIX system call API, as described in Chapter 0 of the xv6 book (i.e., assume that you're writing an xv6 program, so you are limited to xv6 system calls, and you call it "fork"). The program should read bytes from the standard input, fork, execute itself with the exec() system call, and redirect all bytes it reads to its child creating an endless pipeline.

Q2 Calling Conventions

10 Points

Imagine you break right on the printf() line below in the following xv6 program (specifically right on the call instruction that is about to call the printf function inside the bar function):

```
int bar(int x, int y) {
    printf(1, "x:%d, y:%d\n", x, y);
    return x;
}

int foo(int a, int b, int c) {
    return bar(a + b, c);
}

main() {
   foo(1, 2, 3);
   exit(0);
}
```

Please draw the stack and explain every value on the stack. Assume that compiler does not use stack alignment, but maintains stack frames. You can make up any values for the return addresses.

Q3 Address translation

5 Points

Your system has the following page table (assume that the Page Directory Page is at the physical address 0x1000, and the Page Table Page is at the physical address 0x2000).

Page Directory Page:

```
PDE 0: PPN=0x2, PTE_P, PTE_U, PTE_W

... all other PDEs are zero
```

The Page Table Page:

```
PTE 0: PPN=0x3, PTE_P, PTE_U, PTE_W
PTE 1: PPN=0x4, PTE_P, PTE_U, PTE_W

... all other PTEs are zero
```

What virtual addresses (and to what physical addresses) are mapped by this page table?

Q4 More Page Tables

10 Points

Using the same format for describing the page table as in the question above construct the page table that maps the following virtual addresses [0; 4MB] and [2GB; 2GB+4MB] to physical addresses [0; 4MB] (here we map up to 4MB boundary, i.e., the page 4MB and up doesn't have to be mapped). Note: you should use 4KB page tables.

Q5 Process Organization

15 Points

After working on the first question of this exam (the endless pipe) you want to understand how many times it might execute without crashing. You decided to implement a fork bomb, i.e., the program that forks endlessly like this

```
main() {
   while (1) {
     fork();
   }
}
```

How many times fork() in the program above executes successfully running on the xv6 kernel (imagine you changed NPROC (the size of the process array to 64000). Justify your answer, be specific.

Q6 Process Organization

15 Points

Bob says that he can take xv6 and disable the "exec()" system call in the kernel, but he still will be able to execute new programs by implementing "exec()" entirely in user-space. Alice doesn't believe him. Help Bob to design the user-level implementation of exec(). Be specific and justify your answer (provide a code sketch with comments and explanations for what you're doing).

You can assume that you can write and call simple assembly functions that follow normal calling convention if you need. For example, the assembly function below grabs two arguments from the stack and calls the function "func" on the new stack "stack_address" by first saving the function into the eax register, then setting the new stack (moving the "stack_address" into esp) and then calling the function "func" saved in eax.

```
# Call a function on the new stack like
#
# void call_on_new_stack(unsigned long stack_address, unsigned long
func)
#
call_on_new_stack
    mov eax, [esp + 8]
    mov esp, [esp + 4]
    call eax
    ret
```