

Discussion 8

Harishankar Vishwanathan

Agenda

- Midterm review
 - Questions 1 through 5

Question 1

- A program to:
 - Read bytes from the standard input
 - Fork
 - Execute itself with the `exec()` system call
 - Redirect all bytes it reads to its child creating an endless pipeline.

Question 1

```
char buf[1];
pipe(p);
read (0, buf, 1);
pid = fork();

if (pid == 0) {
    close(0);
    dup(p[0]);
    close(p[0]);
    close(p[1]);
    execv(argv[0], argv);
}
```

```
else if (pid > 0) {
    close(1);
    dup(p[1]);
    write(p[1], buf, 1);
    close(p[1]);
}
```

Question 2

```
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);
}
```

Question 2

```
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);
}
```

Fake return PC	
Old EBP	
3	(3rd argument to foo)
2	(2nd argument to foo)
1	(1st argument to foo)
Return address in main	
Old EBP (of main)	
3	(2nd argument to bar)
3 (= 1+2)	(1st argument to bar)
Return address in foo	
Old EBP (of foo)	
3	(4th argument to printf)
3	(3rd argument to printf)
Address of "x:%d, y:%d\n"	(2nd argument to printf)
1	(1st argument to printf)

Question 3

Page Directory Page (at physical address 0x1000)

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

... all other PDEs are zero

The Page Table Page (physical address 0x2000)

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

Question 3

0x1000

0x2000

0x2	P U W

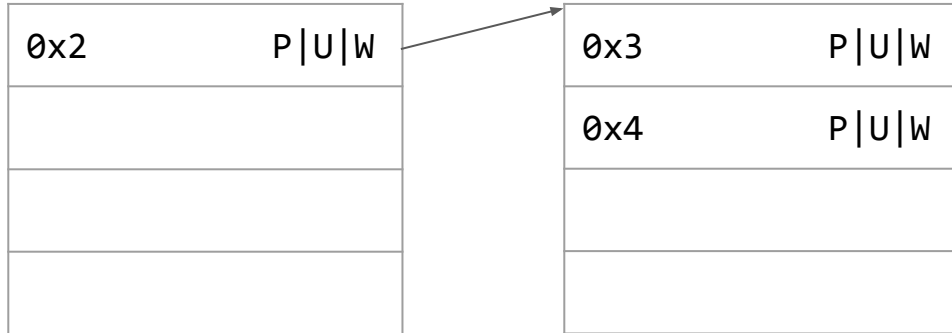


0x3	P U W
0x4	P U W

Question 3

0x1000

0x2000



Phy: 0x0 to 0xFFF

Vir: 0x3000 to 0x3FFF

Phy: 0x1000 to 0x1FFF

Vir: 0x4000 to 0x4FFF

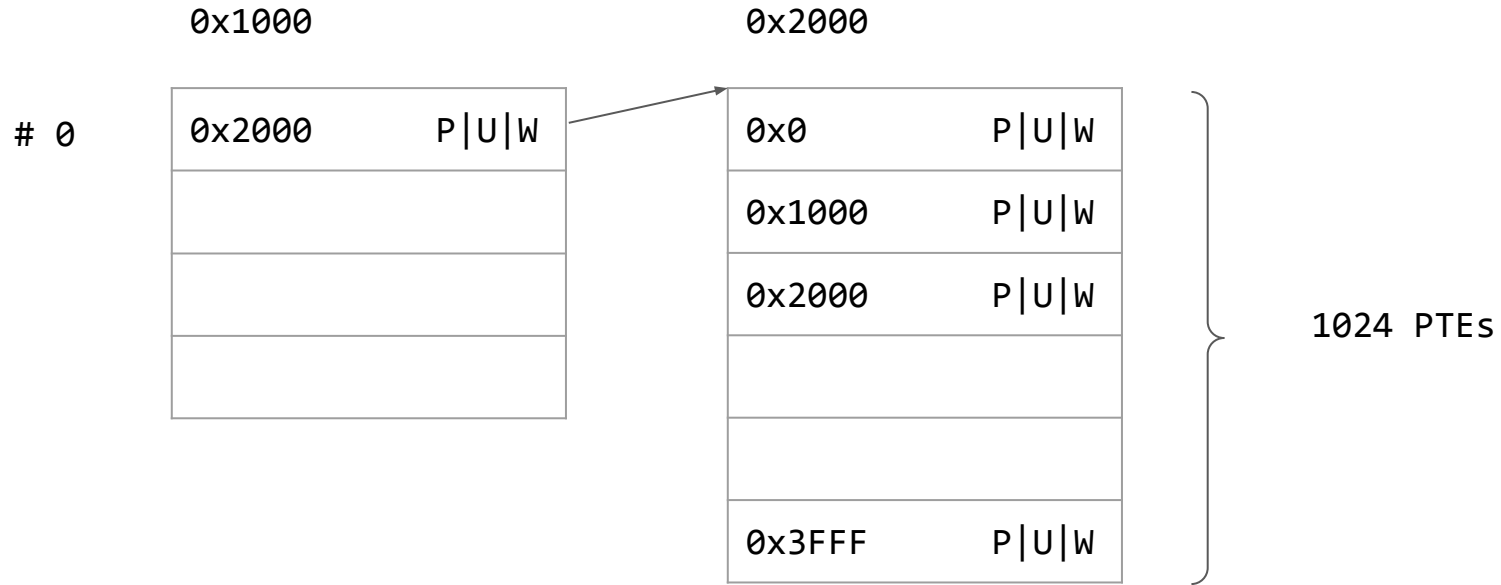
Question 4

Construct the page table that maps the following virtual addresses

- 0 to 4MB to physical addresses 0 to 4MB
- 2GB to 2GB+4MB to physical addresses 0 to 4MB

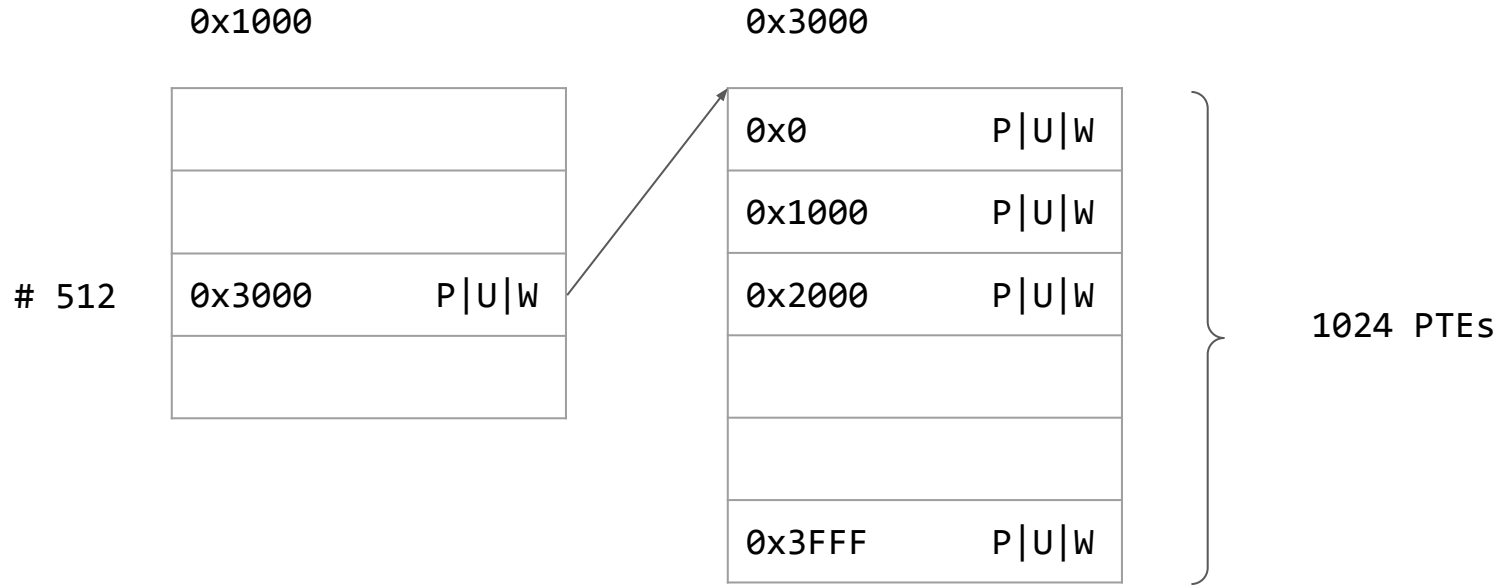
Question 4

- 0 to 4MB to physical addresses 0 to 4MB



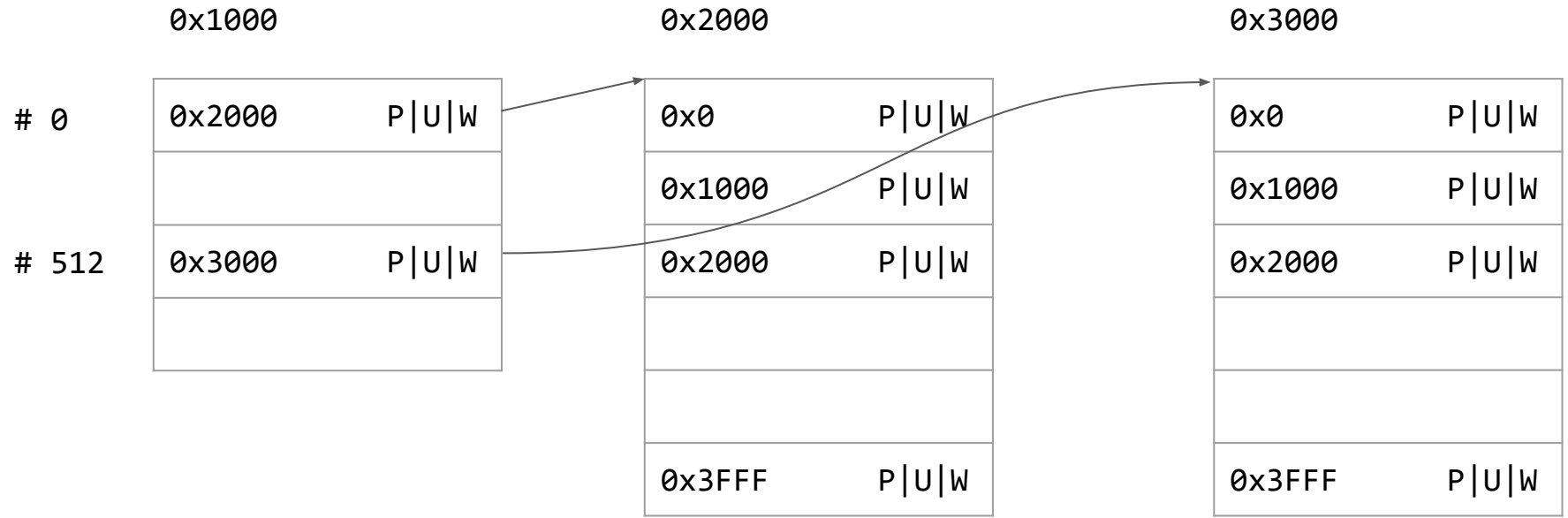
Question 4

- 2GB to 2GB+4MB to physical addresses 0 to 4MB



Question 4

- 2GB to 2GB+4MB to physical addresses 0 to 4MB



Question 5

How many times `fork()` in the program above executes successfully running on the xv6 kernel?

```
# NPROC is 64000
main() {
    while (1) {
        fork();
    }
}
```

- Asking you to estimate, trying to be as specific as possible.
- Answer include SEVERAL details (both implementation and conceptual)
- Marks will be deducted based on the main ideas missed in the estimate.

Question 5

- Estimate available memory (or no. of pages available) before kernel starts init and shell
- Estimate no. of pages per process, typically
- Calculate number of forks in the forkbomb

Question 5

- **Estimate available memory (or no. of pages available) before kernel starts init and shell**
 - Assume a kernel end virtual address
 - Calculate first virtual address of first page donated to the kernel memory allocator (hw)
 - Calculate the size of the kernel page tables
 - It maps 4 regions (kmap), 65536 pages
 - X page directories, Y page tables.
 - Until now: size of kernel image + size of page tables

Question 5

- **Estimate no. of pages per process, typically**
 - 1 page per region: text, data, (guard), stack
 - 1 page directory page
 - 1 page table page for mapping the different regions
 - Every process maps the kernel
 - Calculate the number of pages for mapping KERNBASE:KERNBASE+PHYSTOP (to 0:PHYSTOP)
 - Add page table pages and page directory entries if needed
 - 1 page for the kernel stack

Question 5

- Calculate number of forks in the forkbomb
 - Init, shell, fork

