CS444/544 Operating Systems II Instructor: Dr. Yeongjin Jang

Name: OSU ID:

CS 444/544 Operating Systems II
Sample Quiz #1

You have 30 minutes to answer the questions in this quiz. In order to receive credit you must answer the question as precisely as possible.

If you find any ambiguity in the questions, be sure to write down any assumptions you make. You do not have to list all of the assumptions.

In case if we cannot read your answer nor interpret your answer, we can't give you credit.

Write your name and OSU ID on this cover sheet, and make sure you have all pages with the quiz sheet package: Sample Quiz #1 should have total 3 pages.

NO Internet access, NO communication with other students, and NO consulting to textbook, slides, laptop, etc.

Section	I	II	Total
Score			
Max Score	25	75	100

I. Multiple choices (25 pts, 5 pts each)

I-1. Which register you need to control (i.e., set a value to the register) to enable the protected mode and paging on Intel x86 processor?

a) eax	**b) CR0**	c) CR1	d) CR2	e) CR3

I-2. Which register is the one that the Intel x86 CPU reserves to point the page directory of the currently running application?

a) eax	b) CR0	c) CR1	d) CR2	**e) CR3**

I-3. From the followings, choose two traps that must be available for Ring 3

a) NMI	**b) Breakpoint**	c) Divide Error	d) Page Fault	**e) System Call**

FYI, NMI stands for Non-maskable Interrupt.

I-4. To access the physical address 0x212233 in JOS kernel, you may refer to the virtual address at:

a) 0x212233	b) 0x2122330	c) 0xe0212233	**d) 0xf0212233**	e) 0xf2122330

I-5. To let CPU know about the address of the current page directory via the register that you will put the answer for I-2, you need to put the ["*choose one from below*"] of the page directory to that register.

a) virtual address	**b) physical address**	c) physical address + KERNBASE	d) virtual address + KERNBASE

II. Segmentation, Paging, and Virtual Memory (75 pts)

II-1. (5pts) In the x86 real mode, accessing an address 0xb800:0001 ([segment:offset]) will access the physical address at

Your answer here: **0xb800 * 16 + 0x0001 = 0xb8001**

II-2. (30 pts, 10 pts each) Suppose we have the following Global Descriptor Table (GDT), and our CPU is in the 80386 Protected Mode (the mode that we use in JOS). Note that all values are in hexadecimal.

GDT OFFSET	BASE	LIMIT	FLAGS
0x8	0x41410101	0x1000	Assume the flags set as accessible to current execution
0x10	0x42424343	0x2000	Assume the flags set as accessible to current execution

a) Which address will be accessed if you let the CPU read 0x8:0102 ?

Answer: **0x41410101 + 0x0102 = 0x41410203**

b) Which address will be accessed if you let the CPU read 0x10:1111 ?

Answer: **0x42424343 + 0x1111 = 0x42425454**

c) What will happen if you let the CPU read 0x10:8888 or 0x8:4444 ?

Answer: **Offsets are over the limits (0x2000 and 0x1000, respectively) of GDT entries, so CPU will generate segmentation fault...**

II-3. (20pts) In JOS, we manage each of physical page by having an array,

`struct PageInfo * pages`.

Each of element in that array, e.g., pages[i], refers to the i-th physical page and maintains the number of virtual references to the page. In using this structure, we sometimes need to get the physical address of the elements of the array, pages. For this purpose, JOS prepares a function, page2pa(), which you can get the corresponding physical address of an element of pages.

Now, here is the question for you. The following is a skeleton of the page2pa() function. Can you fill the blank to implement page2pa()?

```
Struct PageInfo *pages;  // a global variable

physaddr_t page2pa(struct PageInfo *pp) {

        return      (pp - pages) << PGSHIFT;

}
```

Hint: (`pp - pages`) returns the index of `pp` in the array `pages`.

II-4. (20 pts, 5 pts each) Suppose you have the following page directory and table.

Page Directory at 0x300000	Top 20 bits	Lower 12 bits
...	...	...
0x18	0x33220	PTE_P | PTE_U | PTE_W
0x19	0x44221	PTE_P | PTE_U
...	...	...

Page Table at 0x33220000	Top 20 bits	Lower 12 bits
...	...	...
0x20	0x12345	PTE_P | PTE_U | PTE_W
0x21	0x23456	PTE_P | PTE_W
...	...	...

Page Table at 0x44221000	Top 20 bits	Lower 12 bits
...	...	...
0x20	0x34567	PTE_P | PTE_U
0x21	0x45678	PTE_P | PTE_U | PTE_W
...	...	...

Please answer the following questions for the address translations in x86 paging.
FYI, (0x18 << 10) == 0x60, (0x19 << 10) == 0x64.
In reverse, (0x6020 >> 10) == 0x18, (0x6420 >> 10) == 0x19.

a) Accessing the virtual address 0x6020333 will access the physical address at: **0x12345333**

b) From a user mode, you can perform write on 0x6021345: **False, No PTE_U in PT entry (0x21)**

True **False**

c) Accessing the virtual address 0x6421333 will access the physical address at: **0x45678333**

d) From a user mode, you can perform write on 0x6421333: **False, No PTE_W in PD entry (0x19)**

True **False**