Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.828 Operating System Engineering: Fall 2003

Quiz I

All problems are open-ended questions. In order to receive credit you must answer the question as precisely as possible. You have 80 minutes to answer this quiz.

Write your name on this cover sheet AND at the bottom of each page of this booklet.

Some questions may be much harder than others. Read them all through first and attack them in the order that allows you to make the most progress. If you find a question ambiguous, be sure to write down any assumptions you make. Be neat. If we can't understand your answer, we can't give you credit!

THIS IS AN OPEN BOOK, OPEN NOTES QUIZ.

1 (xx/19)	2 (xx/15)	3 (xx/25)	4 (xx/20)	5 (xx/15)	6 (xx/6)	Total (xx/100)

Name:

I Calling conventions

In lab 2 you extended `printf`. Here is the core of `printf` relevant for this question:

```
void printk(const char *fmt, va_list ap)
{
        register char *p, *q;
        register int ch, n;
        u_quad_t uq;
        int base, lflag, qflag, tmp, width;
        char padc;

        for (;;) {
                padc = ' ';
                width = 0;
                while ((ch = *(u_char *) fmt++) != '%') {
                        if (ch == '\0')
                                return;
                        cons_putc(ch);
                }
                lflag = 0;
                qflag = 0;
        reswitch:
                switch (ch = *(u_char *) fmt++) {
                case 'd':
                        uq = getint(&ap, lflag, qflag);
                        if ((quad_t) uq < 0) {
                                cons_putc('-');
                                uq = -(quad_t) uq;
                        }
                        base = 10;
                        goto number;

                [...  other cases omitted ... not relevant to the question]

                number:
                        p = ksprintn(uq, base, &tmp);
                        if (width && (width -= tmp) > 0)
                                while (width--)
                                        cons_putc(padc);
                        while ((ch = *p--) != '\0')
                                cons_putc(ch);
                        break;
                }
       }
}
```

Name:

```
static u_quad_t
getint(va_list *ap, int lflag, int qflag)
{
        if (lflag)
                return va_arg(*ap, u_long);
        else if (qflag)
                return va_arg(*ap, u_quad_t);
        else
                return va_arg(*ap, u_int);
}


int printf(const char *fmt,...)
{
        va_list ap;

        va_start(ap, fmt);
        kprintf(fmt, ap);
        va_end(ap);
        return 0;
}
```

1. [10 points]: What values does gcc on the x86 push on the stack for the call:

```
printf(‘‘the class number is %s and used to be %d \n’’, ‘‘6828’’,
6097)
```

Name:

2. [4 points]: gcc pushes the arguments in a particular order. What is the order and why?

3. [5 points]: Explain what `va_arg` does briefly.

II Concurrency

Sheets 86 through 87 show the code for a simple device: the paper tape reader.

4. [15 points]: If you delete `spl4()` on line 8686, can you give a concrete sequence of events that results in deadlock? (Hint: you don't have to understand the device deeply to answer this question; focus on the interaction of `sleep` and `wakeup`.)

Name:

III Virtual memory

Here is the layout of virtual memory that you set up in lab 2.

```
/*
 * Virtual memory map:                                Permissions
 *                                                    kernel/user
 *
 *    4 Gig -------->  +-------------------------------+
 *                     |                               | RW/--
 *                     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *                     :               .               :
 *                     :               .               :
 *                     :               .               :
 *                     |~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~| RW/--
 *                     |                               | RW/--
 *                     |  Physical Memory              | RW/--
 *                     |                               | RW/--
 *    KERNBASE ----->  +-------------------------------+
 *                     |  Kernel Virtual Page Table    | RW/--    PDMAP
 *    VPT,KSTACKTOP--> +-------------------------------+                --+
 *                     |        Kernel Stack           | RW/--  KSTKSIZE   |
 *                     |  - - - - - - - - - - - - - - -|                PDMAP
 *                     |       Invalid memory          | --/--             |
 *    ULIM     ------> +-------------------------------+                --+
 *                     |      R/O User VPT             | R-/R-    PDMAP
 *    UVPT      ---->  +-------------------------------+
 *                     |        R/O PAGES              | R-/R-    PDMAP
 *    UPAGES    ---->  +-------------------------------+
 *                     |        R/O ENVS               | R-/R-    PDMAP
 * UTOP,UENVS -------> +-------------------------------+
 * UXSTACKTOP -/       |      user exception stack     | RW/RW   BY2PG
 *                     +-------------------------------+
 *                     |       Invalid memory          | --/--   BY2PG
 *    USTACKTOP  ----> +-------------------------------+
 *                     |     normal user stack         | RW/RW   BY2PG
 *                     +-------------------------------+
 *                     |                               |
 *                     |                               |
 *                     ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 *                     .                               .
 *                     .                               .
 *                     .                               .
 *                     |~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~|
 *                     |                               |
 *    UTEXT ------->   +-------------------------------+
 *                     |                               |  2 * PDMAP
 *    0 ------------>  +-------------------------------+
 */
```

Name:

Attached to the quiz is the `i386_vm_init` that was provided to you. Assume you completed `i386_vm_init` correctly.

5. [10 points]: What entries (rows) in the page directory have been filled in after `i386_vm_init` has completed? What addresses do they map and where do they point? In other words, fill out this table as much as possible:

Entry	Base Virtual Address	Points to (logically):
1023	?	Page table for top 4MB of phys memory
1022	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
.	?	?
2	0x00800000	?
1	0x00400000	?
0	0x00000000	?

Name:

6. [5 points]:

Here is a section of the course staff solution to Lab 2's `i386_vm_init`. This section sets up the UPAGES mapping.

```
///////////////////////////////////////////////////////////////////
// Make 'pages' point to an array of size 'npage' of 'struct Page'.
// You must allocate this array yourself.
// Map this array read-only by the user at virtual address UPAGES
// (ie. perm = PTE_U | PTE_P)
// Permissions:
//    - pages -- kernel RW, user NONE
//    - the image mapped at UPAGES  -- kernel R, user R
// Your code goes here:
n = npage*sizeof(struct Page);
pages = alloc(n, BY2PG, 1);
boot_map_segment(pgdir, UPAGES, n, PADDR(pages), PTE_U);
```

A common mistake is to add the line:

```
boot_map_segment(pgdir, (u_int)pages, n, PADDR(pages), PTE_W);
```

This line is unnecessary, because the mapping already exists. Why does the mapping already exist? Explain exactly which other code has already provided the mapping. You may find it useful to refer to the `i386_vm_init` attached to this quiz.

Name:

7. [5 points]:

In Lab 3, `env.c` creates the user-level address space for an environment. If the code that created the address space was buggy and did not set up a mapping for the area starting at KERNBASE, when would this bug manifest itself? What specific instruction would cause the bug to "take effect" (triple-fault the processor)? (Note: you can answer this question without having completed lab 3.)

8. [5 points]: On the x86, we make the kernel's `Page` structures accessible to the user environments (in the form of the mapping at UPAGES). What specific mechanism (i.e., what register, memory address, or bit thereof) is used to keep the user environments from changing the `Page` structures?

IV System calls

9. [5 points]: Draw the kernel stack after v6's `icode` called its first instruction and the kernel just entered `trap` in trap.c (sheet 26).

Name:

10. [10 points]: What are the values of the arguments to trap? (Fill out the following table.)

`dev`	
`sp`	
`r1`	
`nps`	
`r0`	
`pc`	
`ps`	

11. [5 points]: Briefly describe the point of the statement on line 3188.

Name:

V Thread switching

A process in UNIX v6 switches to another process using `retu`, which is called on line 2228.

12. [10 points]: Annotate every line of assembly of `_retu` (reproduced from sheet 07). What does the statement do and why?

```
_retu:
   bis     $340, PS

   mov     (sp)+, r1

   mov     (sp), KISA6

   mov     $_u, r0

1:
   mov     (r0)+, sp

   mov     (r0)+, r5

   bic     $340, PS

   jmp     (r1)
```

13. [5 points]: What does the stack pointer point to at line 2229 in the first call to swtch, after the kernel booted?

Name:

VI Feedback

Since 6.828 is a new subject, we would appreciate receiving some feedback on how we are doing so that we can make corrections. (Any answer, except no answer, will receive full credit!)

14. [2 points]: What is the best aspect of 6.828?

15. [2 points]: What is the worst aspect of 6.828?

16. [2 points]: If there is one thing that you would like to see changed in 6.828, what would it be?

End of Quiz I

Name:

Here is `i386_vm_init` from Lab 2.

```
// Set up a two-level page table:
//    boot_pgdir is its virtual address of the root
//    boot_cr3 is the physical adresss of the root
// Then turn on paging.  Then effectively turn off segmentation.
// (i.e., the segment base addrs are set to zero).
//
// This function only sets up the kernel part of the address space
// (ie. addresses >= UTOP).  The user part of the address space
// will be setup later.
//
// From UTOP to ULIM, the user is allowed to read but not write.
// Above ULIM the user cannot read (or write).
void
i386_vm_init(void)
{
  Pde *pgdir;
  u_int cr0, n;

  panic("i386_vm_init: This function is not finished\n");

  //////////////////////////////////////////////////////////////////////
  // create initial page directory.
  pgdir = alloc(BY2PG, BY2PG, 1);
  boot_pgdir = pgdir;
  boot_cr3 = PADDR(pgdir);

  //////////////////////////////////////////////////////////////////////
  // Recursively insert PD in itself as a page table, to form
  // a virtual page table at virtual address VPT.
  // (For now, you don't have understand the greater purpose of the
  // following two lines.)

  // Permissions: kernel RW, user NONE
  pgdir[PDX(VPT)] = PADDR(pgdir)|PTE_W|PTE_P;

  // same for UVPT
  // Permissions: kernel R, user R
  pgdir[PDX(UVPT)] = PADDR(pgdir)|PTE_U|PTE_P;

  //////////////////////////////////////////////////////////////////////
  // Map the kernel stack (symbol name "bootstack"):
  //   [KSTACKTOP-PDMAP, KSTACKTOP)  -- the complete VA range of the stack
  //     * [KSTACKTOP-KSTKSIZE, KSTACKTOP) -- backed by physical memory
  //     * [KSTACKTOP-PDMAP, KSTACKTOP-KSTKSIZE) -- not backed => faults
  //   Permissions: kernel RW, user NONE
  // Your code goes here:

  //////////////////////////////////////////////////////////////////////
  // Map all of physical memory at KERNBASE.
  // Ie.  the VA range [KERNBASE, 2^32 - 1] should map to
  //      the PA range [0, 2^32 - 1 - KERNBASE]
  // We might not have that many(ie. 2^32 - 1 - KERNBASE)
  // bytes of physical memory.  But we just set up the mapping anyway.
  // Permissions: kernel RW, user NONE
  // Your code goes here:

  //////////////////////////////////////////////////////////////////////
  // Make 'pages' point to an array of size 'npage' of 'struct Page'.
  // You must allocate this array yourself.
  // Map this array read-only by the user at virtual address UPAGES
  // (ie. perm = PTE_U | PTE_P)
  // Permissions:
  //    - pages -- kernel RW, user NONE
  //    - the image mapped at UPAGES  -- kernel R, user R
  // Your code goes here:

  //////////////////////////////////////////////////////////////////////
  // Make 'envs' point to an array of size 'NENV' of 'struct Env'.
  // You must allocate this array yourself.
  // Map this array read-only by the user at virtual address UENVS
  // (ie. perm = PTE_U | PTE_P)
  // Permissions:
  //    - envs itself -- kernel RW, user NONE
  //    - the image of envs mapped at UENVS  -- kernel R, user R
  // Your code goes here:

  check_boot_pgdir();

  //////////////////////////////////////////////////////////////////////
  // On x86, segmentation maps a VA to a LA (linear addr) and
  // paging maps the LA to a PA.  I.e. VA => LA => PA.  If paging is
  // turned off the LA is used as the PA.  Note: there is no way to
  // turn off segmentation.  The closest thing is to set the base
  // address to 0, so the VA => LA mapping is the identity.

  // Current mapping: VA KERNBASE+x => PA x.
  //     (segmentation base=-KERNBASE and paging is off)

  // From here on down we must maintain this VA KERNBASE + x => PA x
  // mapping, even though we are turning on paging and reconfiguring
  // segmentation.

  // Map VA 0:4MB same as VA KERNBASE, i.e. to PA 0:4MB.
  // (Limits our kernel to <4MB)
  pgdir[0] = pgdir[PDX(KERNBASE)];

  // Install page table.
  lcr3(boot_cr3);

  // Turn on paging.
  cr0 = rcr0();
  cr0 |= CR0_PE|CR0_PG|CR0_AM|CR0_WP|CR0_NE|CR0_TS|CR0_EM|CR0_MP;
  cr0 &= ~(CR0_TS|CR0_EM);
  lcr0(cr0);
```

```
  // Current mapping: KERNBASE+x => x => x.
  // (x < 4MB so uses paging pgdir[0])

  // Reload all segment registers.
  asm volatile("lgdt _gdt_pd+2");
  asm volatile("movw %%ax,%%gs" :: "a" (GD_UD|3));
  asm volatile("movw %%ax,%%fs" :: "a" (GD_UD|3));
  asm volatile("movw %%ax,%%es" :: "a" (GD_KD));
  asm volatile("movw %%ax,%%ds" :: "a" (GD_KD));
  asm volatile("movw %%ax,%%ss" :: "a" (GD_KD));
  asm volatile("ljmp %0,$1f\n 1:\n" :: "i" (GD_KT));  // reload cs
  asm volatile("lldt %0" :: "m" (0));

  // Final mapping: KERNBASE+x => KERNBASE+x => x.

  // This mapping was only used after paging was turned on but
  // before the segment registers were reloaded.
  pgdir[0] = 0;

  // Flush the TLB for good measure, to kill the pgdir[0] mapping.
  lcr3(boot_cr3);
}
```