Department of Electrical Engineering and Computer Science

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

6.828 Operating System Engineering: Fall 2004

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/40)	2 (xx/25)	3 (xx/25)	4 (xx/10)	Total (xx/100)
40	17 AG	25 AG	10 FL	92 AG

Name: DAN PORTS

I Virtual memory

1. [5 points]: The PDP-11 uses two bits in the Processor Status Word (PSW) to determine whether it is currently running in kernel or user mode. Which bits in which register on the x86 determine whether the processor is running in kernel or user mode, and what x86 instruction does the JOS kernel use to enter user mode? (Explain briefly)

The bottom 2 bits of the CS register are the CPL, which determine whether the kernel is executing in kernel or user mode

5

The IRET instruction loads a new CS and EIP, so it is used to switch from kernel to user mode.

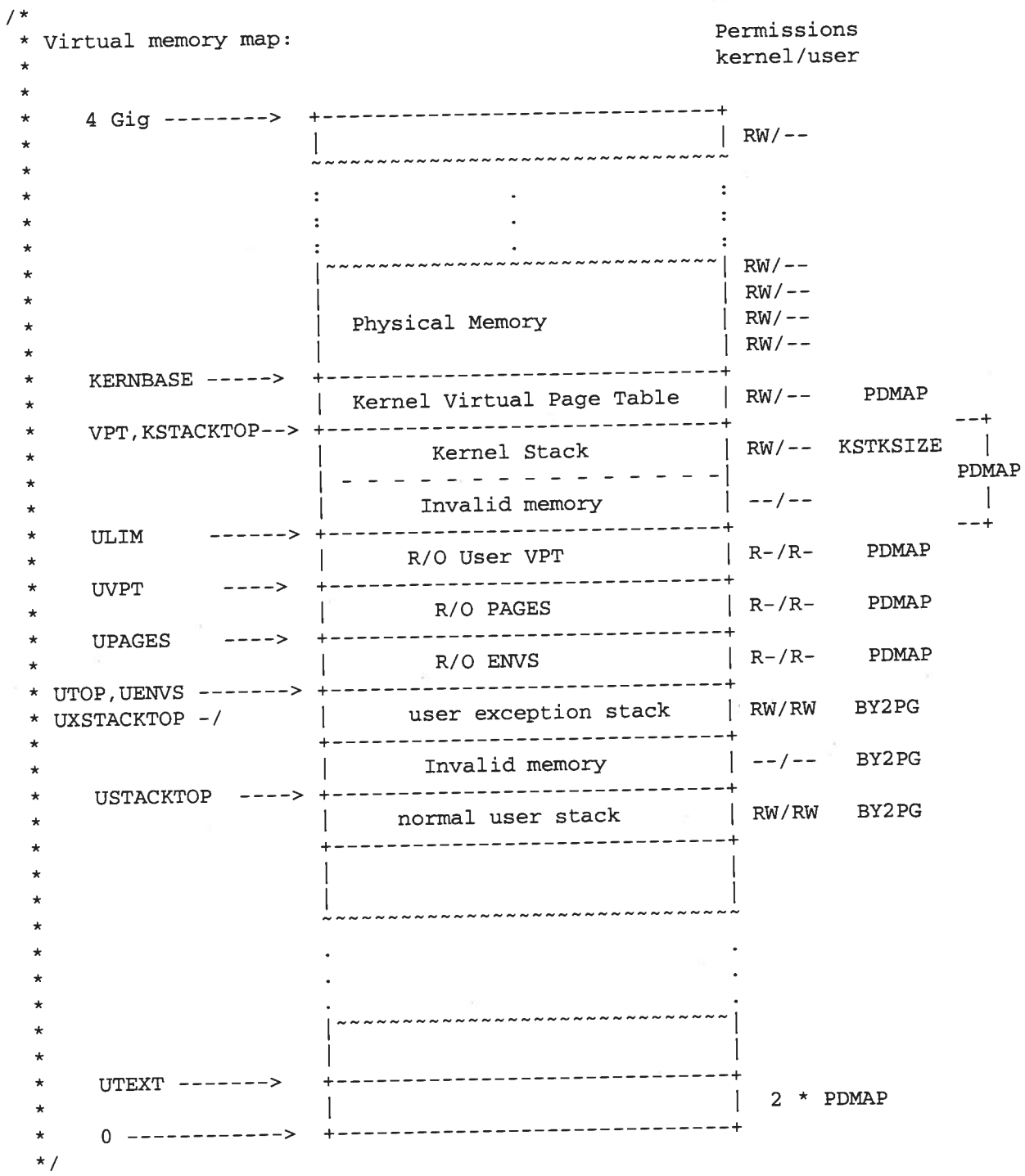
2. [5 points]: The v6 kernel must copy values from user space with the special instruction `mfpi` (push onto the current stack the value of the designated word in the "previous" address space). What instructions does the JOS kernel use to access user-level values? (Explain briefly)

The V6 kernel needs a separate instruction for this because there is a different set of PAR/PDR registers for the user and kernel spaces. This isn't true on the x86 so the kernel can directly access the memory of the current user process (mapped below UTOP) with the standard mov* instructions, etc.

5

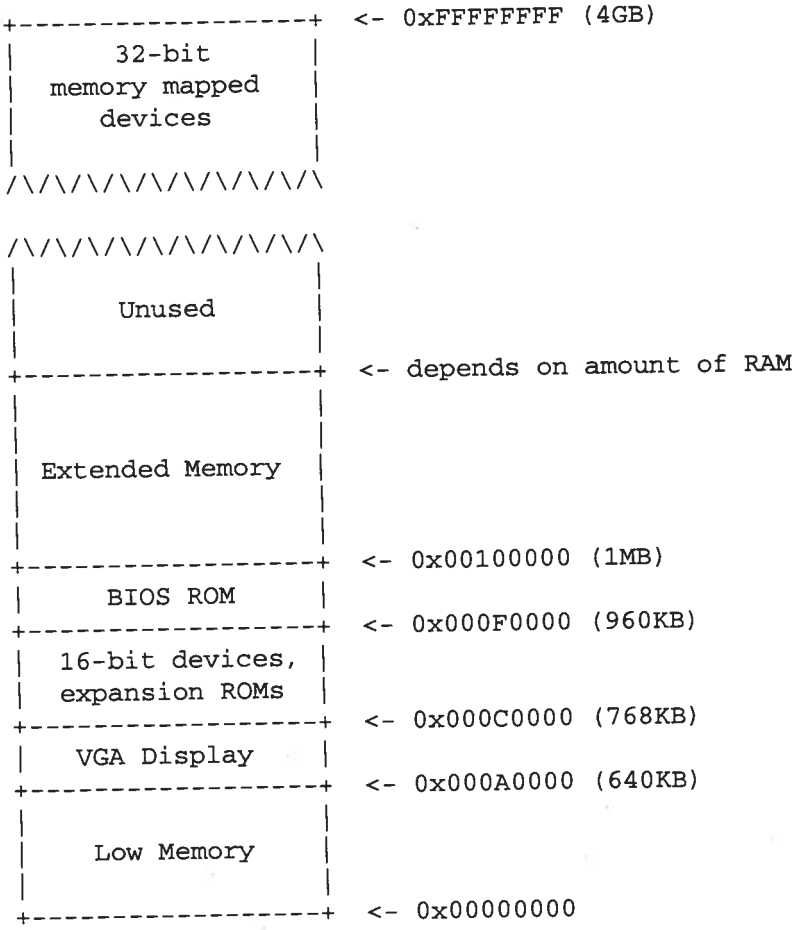
Name: DAN PORTS

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



Name: DAN PORTS

Also, here is the layout of physical memory in a PC:



3. [5 points]: Why does the JOS kernel arrange that the virtual addresses from KERNBASE and higher have read-write access in kernel mode and no access in user mode? (Explain briefly.)

The addresses above KERNBASE map the first 256 MB of physical memory. The kernel needs access to this in order to access physical memory locations (e.g. in page_walk). If the user processes had access to this, they could access the physical memory used by other processes and make a mockery of fault isolation.

5

Name: DAN PORTS

You completed the following function in lab 2:

```
//
// Initialize page structure and memory free list.
//
page_init(void)
{
        // The exaple code here marks all pages as free.
        // However this is not truly the case.  What memory is free?
        //  1) Mark page 0 as in use(for good luck)
        //  2) Mark the rest of base memory as free.
        //  3) Then comes the IO hole [IOPHYSMEM, EXTPHYSMEM) => mark it as in u
se
        //     So that it can never be allocated.
        //  4) Then extended memory(ie. >= EXTPHYSMEM):
        //     ==> some of it's in use some is free. Where is the kernel?
        //     Which pages are used for page tables and other data structures?

        //
        // Change the code to reflect this.
        int i;
        LIST_INIT (&page_free_list);
        for (i = 0; i < npage; i++) {
                pages[i].pp_ref = 0;
                LIST_INSERT_HEAD(&page_free_list, &pages[i], pp_link);
        }
}
```

Name: DAN PORTS

4. [10 points]: Which pages in extended memory should `page_init` mark as in use? (Explain briefly and feel free to refer to the virtual memory map.)

The kernel ELF image, page dir, some page tables, and pages array are allocated and mapped before page_init is called. So they must be marked as in use. This is all extended memory below the freemem pointer used in alloc.

10

5. [5 points]: Why should `page_init` mark page 0 as in use for good luck? (Explain briefly.)

Because luck and black magic are critical elements of kernel development (look at V6!). ☺

5

Also, because various BIOS procedures will access and modify physical addresses in low memory, and hence kernel or user data should not be stored there

Name: DAN PORTS

In lab 3 you wrote the following code to setup the IDT for system calls in `idt_init`:

```
SETGATE(idt[T_SYSCALL], 1, GD_KT, handler_SYSCALL, 3);
```

6. [5 points]: Why must the descriptor protection level be set to 3 instead of 0 (as with most of the other entries in the IDT)? (Explain briefly.)

User processes should be able to make syscalls with the int 30 instruction, but not trigger other interrupts (e.g. Page faults) this way.

5

7. [5 points]: If the descriptor protection level is set to 0, what would happen if the processor executes an "int 30" instruction? (Explain briefly.)

A general protection fault will occur, if the CPL is not zero. If it is zero, the processor will execute handler_SYSCALL.

5

Name: DAN PORTS

II Proccesses

8. [5 points]: In which structure does v6 allocate the kernel stack? (Explain briefly.)

The "struct user" named "U" (the u-area)

✓ +5

9. [5 points]: How many of these structures does v6 allocate? (Explain briefly.)

One per process.

✓ +5

10. [5 points]: The structure (and thus the kernel stack) is stored at the *virtual* address 140000 and up. What must happen when the kernel switches to a different process? (Explain briefly.)

The U struct is at a fixed location in memory. To switch to a different U struct, when switching processes, the kernel must remap the page starting at 0140000 to a new physical memory segment by changing KIPAR6 and KIPDR6

✓ +5

Name: DAN POATS

11. [10 points]: In v6, the `init` process, pid 1, inherits processes whose parent have exited. Sketch out the required changes to the kernel so that children that have no parent anymore clean themselves up on `exit` (sheet 32).

The code inside the if on lines 3252-3253 assigns orphaned processes to init. To have orphaned children die, we could add the following line:

```
Psignal(p, SIGKILL)
```

This sends signal 9 to the orphaned process, causing it to die.

+2

misinterpretation

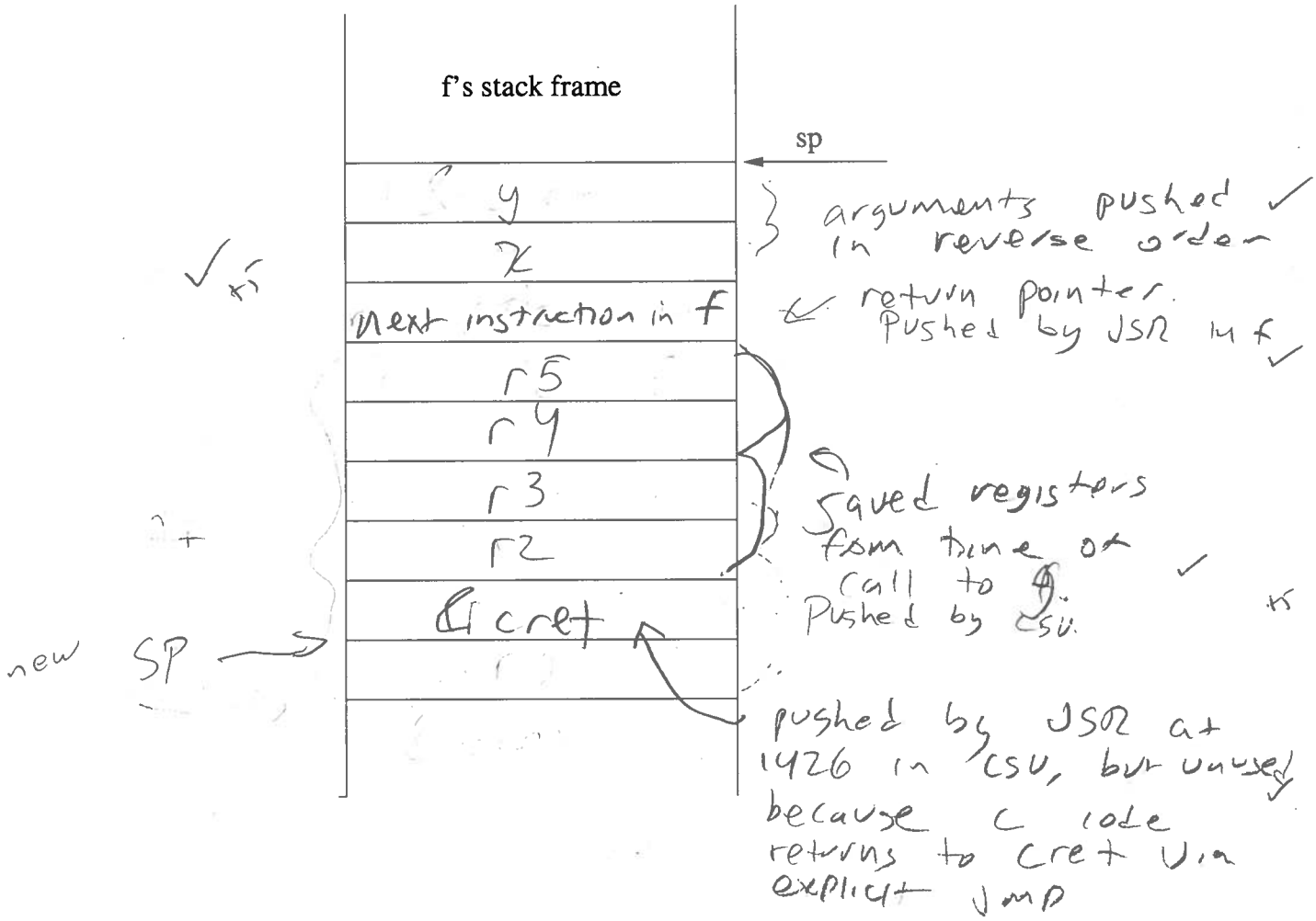
Name: Dan Ports

III Calling conventions

12. [5 points]: Assume a function `f` calls a function `g` using the C calling conventions of v6 on the PDP-11. The callee function `g` takes two arguments, as follows:

```
void g(int x, int y) {
...
}
```

Below is the stack at the point that `f` is about to call `g`. So far `f`'s own local variables and other private state are already on the stack, but nothing related to its call to `g`. Draw the stack at the point just before the processor executes the first instruction of `g` that does "real" work, after running `g`'s function prologue. (The information on page 10-3 of the commentary might be helpful.)



13. [5 points]: Annotate your picture by providing for each value a brief explanation why the value is pushed on the stack.

Name: DAN PORTS

The following PDP-11 assembly function appears (with a different name) in the v6 C library:

```
_mystery:
        mov     r5,-(sp)
        mov     2(r5),r1                / pc of caller of caller
        mov     sp,r5
        clr     r0
        cmp     -4(r1),jsrsd
        bne     8f
        mov     $2,r0
8:
        cmp     (r1),tsti
        bne     1f
        add     $2,r0
        br      2f
1:
        cmp     (r1),cmpi
        bne     1f
        add     $4,r0
        br      2f
1:
        cmp     (r1),addi
        bne     1f
        add     2(r1),r0
        br      2f
1:
        cmp     (r1),jmpi
        bne     1f
        add     2(r1),r1
        add     $4,r1
        br      8b
1:
        cmpb    1(r1),bri+1
        bne     2f
        mov     r0,-(sp)
        mov     (r1),r0
        swab    r0
        ash     $-7,r0
        add     r0,r1
        add     $2,r1
        mov     (sp)+,r0
        br      8b
2:
        asr     r0
        mov     (sp)+,r5
        rts     pc

.data
jsrsd:  jsr     pc,*$0
tsti:   tst     (sp)+
cmpi:   cmp     (sp)+,(sp)+
addi:   add     $0,sp
jmpi:   jmp     0
bri:    br      .
```

Name: DAN PORTS

For your convenience, Russ Cox has translated it into some equivalent pseudocode:

```
mystery:
        cpc = *(r5+2)    /* cpc = pc of caller of caller */
        n = 0
        if inst[cpc-4] == "jsr pc, *$x"
                n = 2
again:
        if inst[cpc] == "tst (sp)+"
                n += 2
        else if inst[cpc] == "cmp (sp)+,(sp)+"
                n += 4
        else if inst[cpc] == "add $x, sp"
                n += x
        else if inst[cpc] == "jmp x"
                cpc += x
                goto again
        else if inst[cpc] == "br .+x"
                cpc += x
                goto again
        return n/2
```

14. [15 points]: What the does the `mystery` function compute? (Please, give a brief explanation.)

+15 The number of words removed from the stack by a sequence of tst/cmp/add instructions possibly separated by jmp and brs after the caller returns. This is the number of arguments to the caller, since the caller's caller pushes the arguments onto the stack before calling the caller, then deallocates them after the caller returns.

Name: DAN PORTS

IV 6.828

15. [6 points]: Describe the most memorable error you have made so far in one of the labs. (Provide enough detail that we can understand your answer.)

Not incrementing the refcount on a struct Page correctly every time a page is allocated or mapped. This caused pages to be prematurely released into the free list and reallocated during a copy-on-write page fault handler. Much confusion ensued.

We love to know what your experience is in 6.828. Please, answers the following two questions. (Any answer, except no answer, will receive full credit!)

16. [2 points]: What is the best aspect of 6.828?

I'm writing a real OS / kernel. What more needs to be said?

I told myself a few years ago that my life wouldn't be complete until I'd done this.

17. [2 points]: What is the worst aspect of 6.828?

Hours of debugging, especially with x86 grunge.

Midterms scheduled on the same day as the IPTPS submission deadline. ☺

End of Quiz I

Name: DAN PORTS

```
---------------------------
6.828 MID-TERM GRADE REPORT
---------------------------

STUDENT: Dan Ports [drkp]


QUIZ 1 SCORE: 92/100

HOMEWORKS: 5 6 7 9 10 11 12 13 TOTAL:8/8

LAB SCORES:
------------
LAB 1 50    SUBMISSION: drkp.1 Thu-Sep-16-22:37:51 -- LATE: 0 days
LAB 2 50    SUBMISSION: drkp.2 Fri-Oct-1-23:59:45 -- LATE: 0 days
LAB 3 60    SUBMISSION: drkp.4 Sun-Oct-10-17:40:48 -- LATE: 0 days

LAB SUBMISSION LOG:
drkp.1.handin.tar.gz From drkp@MIT.EDU Thu Sep 16 22:37:51 2004
dan@midnight-anchovy.mit.edu.1.handin.tar.gz From dan@midnight-anchovy.mit.edu Wed Sep 15 11:15:20 2004
dan@midnight-anchovy.mit.edu.2.lab2.tar.gz From dan@midnight-anchovy.mit.edu Fri Oct  1 23:59:45 2004
dan@midnight-anchovy.mit.edu.3.lab3.tar.gz From dan@midnight-anchovy.mit.edu Thu Oct  7 23:35:12 2004
dan@midnight-anchovy.mit.edu.4.lab3.tar.gz From dan@midnight-anchovy.mit.edu Sun Oct 10 17:40:48 2004
dan@midnight-anchovy.mit.edu.5.lab4.tar.gz From dan@midnight-anchovy.mit.edu Mon Oct 18 19:01:05 2004
dan@midnight-anchovy.mit.edu.6.lab4.tar.gz From dan@midnight-anchovy.mit.edu Sat Oct 23 04:22:19 2004
```