Full Name: _____

CS 450 Spring 2010

Final Exam

May 5, 2010
¡Happy Cinco de Mayo!

Instructions:

- This exam is closed-book, closed-notes.
- Write your full name on the front, and make sure that your exam is not missing any sheets.
- Good luck!

Problem 1 (/15) :	
Problem 2 (/9) :	
Problem 3 (/9) :	
Problem 4 (/9) :	
Problem 5 (/9) :	
Problem 6 (/4) :	
Problem 7 (/4) :	
TOTAL (/50) :	

Problem 1. (15 points):

Multiple choice. For each of the following multiple choice problems, choose the *single best* answer by circling its corresponding letter.

1. Which of the following bits of code can be used to update the PC and PS registers so as to yield control back to the "user half" of a process?

   (a) `rtt`

   (b) `rts pc`

   (c) `jsr pc,pword`

   (d) `mov UISD0,-(sp)`

2. Which of the following data/structures associated with a given process *must remain in core* while the process is swapped out?

   (a) the process priority

   (b) the prototype segmentation registers

   (c) the process code/text segment

   (d) the kernel stack

3. It is frequently the case that v6 code needs to be run *atomically*. Which of the following can be used to *begin* an atomic chunk of code?

   (a) `bis  $340, PS`

   (b) `spl0()`

   (c) `rp->p_stat = SWAIT`

   (d) `mov  $1, SSR0`

4. Which of the following will almost certainly be executed by the kernel in response to every system call invocation?

   (a) `jsr r0,call1; _trap`

   (b) `mov nofault,(sp)`

   (c) `wakeup(&proc[1]);`

   (d) `newproc()`

5. One "gotcha" in `newproc` occurs when there is insufficient core space for the new process. Which of the following captures the context of the process before the necessary swap-out ?

   (a) `rp->p_flag =& ~SSWAP;`

   (b) `aretu(u.u_ssav)`

   (c) `savu(u.u_rsav)`

   (d) `savu(u.u_ssav)`

Problem 2. (9 points):

The following bit of code is taken from the `newproc` function. Recall that `rip` points to the current process, and `rpp` points to the process being created.

```
1859   rip = u.u_procp;
1860   up = rip;
1861   rpp->p_stat = SRUN;
1862   rpp->p_flag = SLOAD;
1863   rpp->p_uid = rip->p_uid;
1864   rpp->p_ttyp = rip->p_ttyp;
1865   rpp->p_nice = rip->p_nice;
1866   rpp->p_textp = rip->p_textp;
1867   rpp->p_pid = mpid;
1868   rpp->p_ppid = rip->p_pid;
1869   rpp->p_time = 0;
```

- What is the purpose of lines 1868-1869?

- After executing lines 1861-1862, which of the two processes is the currently executing process? Explain.

- Later in `newproc` we find the following line just before allocating space for the new process:

  ```
  u.u_procp = rpp;
  ```

  What is its purpose? Be specific.

Problem 3. (9 points):

This problem is based on the following section of code:

```
p = NULL;
n = 128;
i = NPROC;
do {
    rp++;
    if(rp >= &proc[NPROC])
        rp = &proc[0];
    if(rp->p_stat==SRUN && (rp->p_flag&SLOAD)!=0) {
        if(rp->p_pri < n ) {
            p = rp;
            n = rp->p_pri;
        }
    }
} while(--i);
```

- Where and when would you expect to find this code executed in the kernel?

- Justify the initial value of the variable `n` – what does it refer to in the body of the loop?

- Upon exiting the loop, what does the variable `p` refer to?

Problem 4. (9 points):

The following code appears in the body of the clock interrupt handler – note that some lines have been omitted for the sake of brevity. `HZ` is 60, and `SCHMAG` is 10.

```
3794    pp = u.u_procp;
3795    if (++pp->p_cpu == 0)
3796        pp->p_cpu--;
3797    if (++lbolt >= HZ) {
3798        if ((ps&0340) != 0)
3799            return;
3800        lbolt =- HZ;
....
3810        for (pp = &proc[0]; pp < &proc[NPROC]; pp++)
3811        if (pp->p_stat) {
3812            if (pp->p_time != 127)
3813                pp->p_time++;
3814            if ((pp->p_cpu & 0377) > SCHMAG)
3815                pp->p_cpu =- SCHMAG; else
3816                pp->p_cpu = 0;
3817            if (pp->p_pri > PUSER)
3818                setpri(pp);
3819        }
....        /* rest of code omitted */
3830    }
```

- What is the purpose of lines 3795-3796? What is the significance of the `pp->p_cpu` variable?

- How often is the loop at line 3810 executed?

- What is the purpose of lines 3815-3816?

Problem 5. (9 points):

You're just starting work at Filesystems-R-Us – a startup with high hopes for its fledgling file system. On your first day you notice that their current filesystem is FAT-based, non-journaled, and lacks support for a filesystem buffer. You make for the CTO's office and prepare yourself for a battle.

Your boss tries to justify the design decisions with the following statements. For each statement, supply a rejoinder consisting of (1) a critique of the current approach and (2) a proposal and justification for a different, better approach.

1. "FAT is compact, efficient, and easy to implement!"

2. "Journaling would force us to duplicate every write to the disk – it's just too inefficient!"

3. "Writing to a filesystem buffer (instead of writing directly to the disk) results in a loss of durability to the client!"

Problem 6. (4 points):

One of the alternatives we considered to the journaling mechanism implemented by most modern filesystems is the notion of a *log-structured filesystem*. Briefly explain and discuss the pros and cons of a log-structured filesystem.

Problem 7. (4 points):

In addition to improving robustness via software journaling, we also considered increasing failure resistance through the use of various RAID levels. Given the right combination of RAID techniques, is it possible to circumvent the use of journaling/logging altogether? Explain.