Computer Science 161

Final Exam

May 11, 2009 - May 12, 2009 5:00 PM

220 minutes/220 points

The points allocated to each problem correspond to how much time we think the problem should take. Although you have 24 hours in which to take the exam, we do not recommend that you pull an all-nighter and spend 24 hours completing the exam. Here is Margo's advice on how to complete this exam:

Download the exam and read through the entire exam.
Go to dinner and think about the exam.
Spend 3-4 hours Monday evening writing up answers the questions that you know.
Get a good night's sleep.
Spend another 4-5 hours on Tuesday working on the remaining problems.
Read over your answers.
Email them to me.

Please place the answers to the questions in a PLAIN TEXT file. Mark the answers clearly so it is obvious what answers correspond to what questions. Please place your answers in order in the file. Send your completed exam to margo@eecs.harvard.edu. Please enclose your answers directly in your mail message - do not make it a separate attachment.

This exam is open text book, open source code, and open notes. **You may not use computers**[1] **during the exam** (other than to record your answers, look up the manual page for mmap, and to view OS/161 code). You must complete the exam on your own.

I have neither given nor received help on this exam.

Please sign your name here indicating that you have neither given nor received help on this exam.

1. Computers include cell phones, pdas, smart phones or any device with which you can talk to a network.

Short Answer (60 points)

For these questions, try to formulate your answer as clearly and succinctly as possible. For the first few, you ought to be able to answer in a few sentences or a paragraph. After that, if you are writing more than half a page, you haven't thought through the problem. Do not write every idea that you have; think carefully about your ideas and pick the most important ones, explain those clearly and justify them.

1. What deadly food item was responsible for the midterms not being graded as quickly as originally intended? (0 points)

2. `thread_exit()` **disables interrupts, but it never turns them back on. How do interrupts ever get turned back on? (5 points)**

3. Given the description below about the characteristics of NAND flash devices, explain why most people immediately think to use a log-structured file system on them. (15 points)

- Bits are organized into pages (approximately 2 KB).
- Pages are grouped into blocks (approximately 256 KB).
- You can read/write to the device in units of pages.
- Write operations can clear bits (make them 0), not set them.
- Erase operations set all the bits in a block (not a page) to 1.
- An erase is about 6-7x the cost of a write and about 60-70x the cost of a read.
- You can write to a page about 4 times before the page requires erasing.
- Blocks may only be erased a limited number of times (somewhere between 10,000 and 1,000,000 times).

4. You have designed and developed a new file system that is now in the hands of your first few customers. Most of your customers are having no difficulty. However, one customer reports that approximately once a week, they find corruption in their file system. The corruption is seemingly random in that they cannot discern any pattern to it. How would you debug this problem? What information would you ask the customer to obtain? What experiments would you ask them to do? What other information might you want to obtain? Your company's future hinges on your finding and fixing this problem. (20 points)

5. Next year, once CS161 is a pleasant memory, a friend approaches you and says that they've not slept for two days because they have a deadlock in their VM system. They ask you for advice on how to debug it -- what do you suggest? ("Ask a teaching fellow." is not a suitable answer.) (15 points)

Critical Evaluation (40 points)

6. You have been hired as a consultant for the UBuildEmWeFixEm (UBEWFE) computer company. They are designing a database management system. Their customer has strong security requirements and must make sure that no one can access data they aren't supposed to. They also want their system to perform well. They have two competing designs. You've been called in to evaluate the designs in terms of correctness, performance, and security.

In both designs, data are stored in files. The data are allocated to the files according to their security requirements. Data accessible only by user J is stored in a file named J; data accessible to users K and P are stored in a file named K_P, etc. The operating system upon which they are building this system supports ACLs, so when the system creates a file K_P, it sets the ACLs such that only users K and P have permission to read and write the data.

Design A: UBEWFE proposes to build a data caching layer at user-level. The data caching layer is implemented as a library against which applications link. The library maintains a single region of memory shared among all the applications. When an application needs data, it first looks in the cache for the data. If the data are in the cache, they are copied into a buffer provided by the application. If the data are not in the cache, the cache library code reads the data from the appropriate file into the cache and then copies it into the application's buffer.

Design B: UBEWFE is going to leverage the operating system's address space. They build a library that interfaces between an application and the OS using mmap. (You may consult manual pages to understand what mmap does.) When an application wants to read a file, the library checks to see if the application has the data mmapped into its address space. If it does, then it simply returns a pointer to the appropriate data to the application. If it does not, then it tries to mmap the file into the application's address space and then return a pointer to the appropriate data. The library limits the number of files it can have mapped into the address space at any one time. If a new file request exceeds this limit, then the library unmaps the least recently used file and maps the new one in.

Write a report (less than 1 page) that summarizes your thoughts on the correctness, security and performance implications of the two designs and then make a final recommendation to the UBEWFE folks.

Synchronization (60 points)

It is rumored that some people have replaced the OS/161 low-level synchronization primitives (spinlocks and wait channels) with a new thing called a "turnstile", defined as follows.

Note: With this new primitive `thread_switch` writes a 0 into a `spinlock_data_t` using `spinlock_data_set()` instead of releasing a spinlock as is currently done. Assume that `threadlist_peekhead` does the obvious thing.

```
1 struct turnstile {
2    spinlock_data_t ts_spin;
3    struct threadlist ts_threads;
4    unsigned ts_code;
5    unsigned ts_unready_count;
6    bool ts_stalled;
7 };
8
9 struct turnstile *
10 turnstile_create(unsigned startcode)
11 {
12   struct turnstile *ts;
13
14   ts = kmalloc(sizeof(*ts));
15   if (ts == NULL) {
16        return NULL;
17   }
18   spinlock_data_set(&ts->ts_spin, 0);
19   threadlist_init(&ts->ts_threads);
20   ts->ts_code = startcode;
21   ts->ts_unready_count = 0;
22   ts->ts_stalled = false;
23   return ts;
24 }
25
26 void
27 turnstile_destroy(struct turnstile *ts)
28 {
29   KASSERT(spinlock_data_get(&ts->ts_spin) == 0);
30   KASSERT(threadlist_isempty(&ts->ts_threads));
31   threadlist_cleanup(&ts->ts_threads);
32   kfree(ts);
33 }
```

```
34 static
35 void
36 turnstile_internal_spin(struct turnstile *ts)
37 {
38   while (spinlock_data_testandset(&ts->ts_spin) != 0) {
39        /* spin */
40   }
41 }
42
43 static
44 void
45 turnstile_internal_unqueue(struct turnstile *ts, unsigned code)
46 {
47   struct thread *t;
48
49   t = threadlist_remhead(&ts->ts_threads);
50   KASSERT(t == curthread);
51   ts->ts_code = code;
52   t = threadlist_peekhead(&ts->ts_threads);
53   thread_make_runnable(t);
54 }
55
56 static
57 unsigned
58 turnstile_internal_queue(struct turnstile *ts, bool ready)
59 {
60   struct thread *t;
61   unsigned ret;
62
63   threadlist_addtail(&ts->ts_threads, curthread);
64   if (ready) {
65        ts->ts_unready_count = 0;
66   }
67   t = threadlist_peekhead(&ts->ts_threads);
68   if (t != curthread) {
69        if (ready && ts->ts_stalled) {
70             ts->ts_stalled = false;
71             thread_make_runnable(t);
72        }
73        thread_switch(S_SLEEP, &ts->ts_spin);
74        turnstile_internal_spin(ts);
75   }
76   ret = ts->ts_code;
77   spinlock_data_set(&ts->ts_spin, 0);
78   return ret;
79 }
```

```
80 unsigned
81 turnstile_register(struct turnstile *ts, bool ready)
82 {
83   turnstile_internal_spin(ts);
84   return turnstile_internal_queue(ts, ready);
85 }

86 void
87 turnstile_unregister(struct turnstile *ts, unsigned code)
88 {
89   turnstile_internal_spin(ts);
90   turnstile_internal_unqueue(ts, code);
91   spinlock_data_set(&ts->ts_spin, 0);
92 }
93
94 unsigned
95 turnstile_shift(struct turnstile *ts, unsigned code, bool ready)
96 {
97   turnstile_internal_spin(ts);
98   if (!ready) {
99        ts->ts_unready_count++;
100       if (ts->ts_unready_count == threadlist_num(&ts-
>ts_threads)) {
101            ts->ts_stalled = true;
102            thread_switch(S_SLEEP, &ts->ts_spin);
103            turnstile_internal_spin(ts);
104            KASSERT(ts->ts_stalled == false);
105       }
106  }
107  turnstile_internal_unqueue(ts, code);
108  return turnstile_internal_queue(ts, ready);
109 }
```

7. What is the external interface for turnstile (i.e., which of the routines provided here are the ones you would expect users of a Turnstile to use)? Describe what each interface function does (such that a programmer would know how to use that routine to synchronize his/her code). (20 points)

8. For each of the following synchronization primitives, write < if the given primitive is weaker than a turnstile, = if the primitive is comparable to a turnstile, and > if the primitive is stronger than a turnstile. By weaker, we mean that you could implement the primitive using a single turnstile, but you could not implement a turnstile using a single instance of the primitive. By stronger, we mean that you could implement a turnstile using the given primitive, but could not implement the primitive using a single instance of a turnstile. By comparable, we mean that you can both write a turnstile in terms of the primitive and write the primitive in terms of the turnstile (note that comparable is a superset of both weaker and stronger; indicate the broadest classification). Write one sentence per primitive justifying your answer. (15 points)

A. semaphore
B. lock
C. condition variable
D. lock and condition variable

9. Pick one of the primitives listed in question 8 and: (35 points)

- If turnstile is comparable to the primitive you selected, show the mapping from one to the other.
- If Turnstile is more powerful than the primitive you selected, provide an implementation for your primitive (all interfaces) using Turnstile.
- If Turnstile is less powerful than the primitive you selected, provide an implementation for Turnstile using your primitive.

Memory Management (55 points)

In a physically mapped processor cache, a page's physical address determines where it can be cached on the processor. For example, if the processor has 1000 pages in its direct-mapped L2 cache, then the pages with physical addresses 1, 1001, 2001, etc will all map to the same page in the cache. Since only one page may be in the cache at any instant, if a program required interleaved access to pages 1, 1001, and 2001, performance would suffer as the pages would be moved back and forth to the cache.

10. Since most programs access both instructions and data sequentially, it seems that the potential for cache conflicts is not worth worrying about. Unfortunately, this is untrue. Explain why this is untrue. (5 points)

Page coloring or cache coloring is a technique that operating system designers use to avoid the problem to which we're alluding in question 10. The basic idea is to assign each virtual memory page a color and require that pages of a particular color be placed in particular parts of the processor cache. Two pages that are likely to be used at the same time by a program are assigned different colors and therefore will be located in different places in the cache, and therefore will not lead to cache thrashing.

11. What information do you need to be able to obtain from the hardware to implement cache coloring? (5 points)

12. Assuming that you have the information that you requested in question 11, write a design document for how you would incorporate cache coloring into OS/161. Be precise about what parts of your system need to be modified, what data structures you would want to augment or develop. You must assume that the reader of this document is not intimately familiar with your OS/161 implementation, so if you need to modify your own structures, please be sure to describe them fully. (45 points)

Parting Thoughts (10 points)

Inside Risks 187*, CACM 49,* 1 January 2006

Software and Higher Education

John C. Knight and Nancy G. Leveson

- * There is too little emphasis in these degree programs on the principles of software development; the emphasis at present is on popular detail and not principles. Important topics such as specification and testing techniques tend to be taught rarely and superficially if at all. Graduates know the syntax of the language du jour and details of the associated libraries, but not how to work with other engineers to specify, design, or test a system. Coverage of all the major elements of software development (including requirements, specification, design, and verification) must be included in degree programs, in such a way that graduates understand what choices are available and how to use them.

13. List 5 principles that you learned in this course.