CS 140 Final Examination
Winter Quarter, 2012

You have 3 hours (180 minutes) for this examination; the number of points for each question indicates roughly how many minutes you should spend on that question. Make sure you print your name and sign the Honor Code below. During this exam you are allowed to consult two double-sided pages of notes that you have prepared ahead of time; other than that, you may not consult books, notes, or your laptop. If there is a trivial detail that you need for one of your answers but cannot recall, you may ask the course staff for help.

I acknowledge and accept the Stanford University Honor Code. I have neither given nor received aid in answering the questions on this examination, and I have not consulted any external information other than two double-sided pages of prepared notes.

____
(Signature)

____
(Print your name, legibly!)

____
(email id, for sending score)

Problem	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	Total
Score												
Max	12	10	10	40	14	8	6	16	14	30	5	165

Problem 1 (12 points)

Indicate whether each of the statements below is true or false, and explain your answer *briefly*.

(a) If a memory allocation mechanism uses a compacting garbage collector for reclamation then it does not experience fragmentation

(b) The dispatcher is responsible for setting thread priorities.

(c) If a thread is CPU-bound, it makes sense to give it higher priority for disk I/O than an I/O-bound thread.

(d) Virtual addresses must be same size as physical addresses.

(e) Page offsets in virtual addresses must be the same size as page offsets in physical addresses.

(f) Implementing processor affinity in a multiprocessor scheduler is likely to reduce the number of misses in caches such as translation lookaside buffers.

Problem 2 (10 points)

Suppose the `grep` program is invoked under Pintos with the following command line:

```
grep inode file.c inode.c
```

Using a diagram for illustration, explain what the call stack will look like when the program's `main` function is invoked. How much total memory will be needed to set up this stack?

Problem 3 (10 points)

(a) If a normal access to memory takes 100 ns (100×10^{-9} sec) and reading a page from disk takes 10 ms (10×10^{-3} sec), what is the average memory access time (including page fault overhead) if page faults occur in 0.1% of the memory references? You may assume that there is no additional overhead due to TLB misses.

(b) What is the maximum allowable page fault rate if performance degradation is to be no more than 10% (i.e. average memory access time $\leq$ 110 ns)?

Problem 4 (40 points)

Suppose that you are given the task of adding **hard** links to Pintos. You must implement a new system call that programs can use to create hard links:

```
bool ln(const char *target, const char *link)
```

The `target` argument contains the path name of an existing file or directory, and `link` contains the path name where a new hard link should be created. After the `ln` system call returns, `link` should refer to the same file or directory as `target`. The system call returns true for success and false for failure. Once created, hard links can be deleted using the `remove` system call that you implemented in Pintos Project 2.

(a) (15 points) Describe the changes you would need to make to Pintos to implement hard links, assuming that you are starting from your solution to Pintos Project 4. You do not need to write actual code, but your answer must be precise enough for us to understand exactly what changes are required. Be sure to describe any changes to data structures, as well as code changes.

Problem 4, cont'd

(b) (10 points) If a system crash occurs during or shortly after a call to `ln` or `remove`, it is possible that some of the on-disk structures may be left in an inconsistent state, which could impact the operation of the system after it restarts. Describe all of the possible inconsistencies that relate to hard links (i.e., information that is manipulated by the `ln` system call). You do not need to consider inconsistencies unrelated to hard links, such as the allocation of disk blocks to files or directories.

(c) (15 points) Suppose you were asked to implement journaling in Pintos in order to repair inconsistencies caused by system crashes. When `ln` or `remove` is invoked, one or more log records will be written before each operation is carried out. During the next system restart, these log records will be processed to repair inconsistencies. Describe the log record(s) that must be written by `ln` and/or `remove` to repair the inconsistencies described in (b) above, and how the information in these records should be used during crash recovery. Assume that hard links are implemented as described in your answer to part (a) above. You do not need to write code, but make sure your answer is precise and complete (use pseudo-code if that's helpful).

Problem 5 (14 points)

Suppose you are asked to design a new file system that will be used exclusively for storing and playing videos on the YouTube Web site.
(a) (4 points) Describe the access patterns that you expect to be most common in this file system.

(b) (4 points) You are given the choice of using flash memory instead of hard disks to store and serve the video data. How would you choose between these two alternatives?

(c) (6 points) How would you expect the design of this file system to differ from the organizations described in class and/or implemented in Pintos, and why?

Problem 6 (8 points)

Suppose a friend told you: "If a system has lots of runnable threads, it can use them to hide the cost of page faults." Explain whether your friend is right, wrong, or both.

Problem 7 (6 points)

Below is one of the attempts presented in lecture for the “Too Much Milk” problem (note: the `buyMilk` function will increment the `milk` variable):

Thread A

```
noteA = 1;
if (noteB == 0) {
    if (milk == 0) {
        buyMilk();
    }
}
noteA = 0;
```

Thread B

```
noteB = 1;
if (noteA == 0) {
    if (milk == 0) {
        buyMilk();
    }
}
noteB = 0;
```

Unfortunately this approach suffers from starvation. A friend suggests the following code in order to fix the starvation problem:

Thread A

```
noteA = 1;
if (noteB == 0) {
    if (milk == 0) {
        buyMilk();
    }
}
noteA = 0;
```

Thread B

```
if (noteA == 0) {
    noteB = 1;
    if (milk == 0) {
        buyMilk();
    }
    noteB = 0;
}
```

Does your friend’s approach solve the “Too Much Milk” problem? Explain your answer.

Problem 8 (16 points) Short answers

(a) (3 points) If the page size is doubled, does the size of the working set for a process increase, decrease, or stay the same? Explain your answer.

(b) (3 points) What causes a thread's state to change from "running" to "ready"?

(c) (3 points) Briefly describe one advantage and one disadvantage of the UNIX `fork`-`exec` style of process creation, in comparison to the Windows `CreateProcess` style.

(d) (4 points) Describe two situations in which busy-waiting is appropriate (and indicate why busy-waiting is appropriate in each situation).

(e) (3 points) Of the four conditions for deadlock, which is the one most commonly eliminated in order to prevent deadlock? Why?

Problem 9 (14 points)

(a) (6 points) Alice, Bob, and Carol go to a Chinese restaurant at a busy time of the day. The waiter apologetically explains that the restaurant can provide only two pairs of chopsticks (for a total of four chopsticks) to be shared among the three people.

Alice proposes that all four chopsticks be placed in an empty glass at the center of the table and that each diner should obey the following protocol:

```
while (!had_enough_to_eat()) {
    acquire_one_chopstick();       /* May block. */
    acquire_one_chopstick();       /* May block. */
    eat();
    release_one_chopstick();       /* Does not block. */
    release_one_chopstick();       /* Does not block. */
}
```

Can this dining plan lead to deadlock? Explain your answer.

(b) (8 points) Suppose now that instead of three diners there will be an arbitrary number, D. Furthermore, each diner may require a different number of chopsticks to eat. For example, it is possible that one of the diners is an octopus, who for some reason refuses to begin eating before acquiring eight chopsticks. The second parameter of this scenario is C, the number of chopsticks that would simultaneously satisfy the needs of all diners at the table. For example, Alice, Bob, Carol, and one octopus would result in C = 14. Each diner's eating protocol will be as displayed below:

```
int s;
int num_sticks = my_chopstick_requirement();
while (!had_enough_to_eat()) {
    for (s = 0; s < num_sticks; ++s) {
        acquire_one_chopstick();     /* May block. */
    }
    eat();
    for (s = 0; s < num_sticks; ++s) {
        release_one_chopstick();     /* Does not block. */

    }
}
```

What is the smallest number of chopsticks (in terms of D and C) needed to ensure that deadlock cannot occur? Explain your answer.

Problem 10 (30 points)

You have been hired to help implement RoboDeli.com, a new automated delicatessen where both the customers and the servers are robots. Like all delis, RoboDeli.com uses a ticketing system and a "Now Serving" display. You must implement the ticketing mechanism in C by defining a structure `struct deli`, plus four functions described below.

The following function is invoked once to initialize the deli object:

```
void deli_init(struct deli *deli)
```

At the time this function is invoked there are no customers and no servers in the deli.

When a new customer arrives in a deli, it invokes the function

```
int deli_get_ticket(struct deli *deli)
```

This function will return the customer's ticket number, which is the smallest positive integer that has not been returned to any previous customer (1 for the first customer, 2 for the second, and so on). No two customers must ever receive the same ticket number. At this point the customer is free to look around the deli. Eventually, each customer will invoke the function

```
deli_wait_turn(struct deli *deli, int number)
```

where `number` is the value returned to the customer when it called `deli_get_ticket`. This function must not return until there is an available server and the "Now serving" display shows a number at least as high as `number`. Once `deli_wait_turn` returns, the customer will go to the counter for service (you do not need to implement that mechanism).

When a new server arrives, or when an existing server finishes with its current customer and is ready to serve a new customer, it will invoke the function

```
deli_wait_customer(struct deli *deli)
```

This function must increment the value displayed in the "Now Serving" display. It must not return until there is a customer ready for service (i.e. invocations of `deli_wait_customer` and `deli_wait_turn` should return in pairs at about the same time). Once this function returns the server will meet the customer and fill their order (you do not need to implement that mechanism).

- You must write your solution in C using the Pintos functions for locks and condition variables:
  ```
  lock_init (struct lock *lock)
  lock_acquire(struct lock *lock)
  lock_release(struct lock *lock)
  cond_init(struct condition *cond)
  cond_wait(struct condition *cond, struct lock *lock)
  cond_signal(struct condition *cond, struct lock *lock)
  cond_broadcast(struct condition *cond, struct lock *lock)
  ```
  Use only these functions (e.g., no semaphores or other synchronization primitives).
- You may not use more than one lock in each `struct deli`.
- You may assume that someone else has written a function `set_now_serving(int number)`, which will cause `number` to be displayed in the "Now Serving" display. Invoke this function as needed in your code.
- Your solution must not use busy-waiting.
- You do not need to worry about ticket numbers overflowing.
- You can assume that customers are always available when their number comes up (you do not need to implement a mechanism to skip over nonresponsive customers).

Use the space below to write a declaration for `struct deli` and the four functions described on the previous page. You do not need to write any other code for the customers or servers.

Additional working space for Problem 10...

Problem 11 (5 points)

Is your solution to Problem 10 fair? Explain why or why not.