Midterm I
March 13th, 2013
CS194-24: Advanced Operating Systems Structures and Implementation

Your Name: 
SID Number: 
Circle the letters of CS194-24 Login: First: a b c d e f g h i j k l m n o p q r s t u v w x y z
Second: a b c d e f g h i j k l m n o p q r s t u v w x y z
Discussion Section: 

General Information:
This is a closed book exam. You are allowed 2 pages of notes (both sides). You may use a calculator. You have 3 hours to complete as much of the exam as possible. Make sure to read all of the questions first, as some of the questions are substantially more time consuming.

Write all of your answers directly on this paper. Make your answers as concise as possible. On programming questions, we will be looking for performance as well as correctness, so think through your answers carefully. If there is something about the questions that you believe is open to interpretation, please ask us about it!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Possible</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>24</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
[ This page left for π ]

3.14159265358979323846264338327950288419716939937510582097494459230781640628620899
Problem 1: True/False [16 pts]
Please EXPLAIN your answer in TWO SENTENCES OR LESS (Answers longer than this may not get credit!). Also, answers without an explanation GET NO CREDIT.

Problem 1a[2pts]: For Test Driven Development (TDD), the important thing is to start by writing tests that pass and to make sure that your tests never fail.

True / False
Explain:

Problem 1b[2pts]: A Zombie process is one that has come back from the dead, i.e. has be restarted after being killed once.

True / False
Explain:

Problem 1c[2pts]: Threads within the same process can share data with one another by passing pointers to objects on their stacks.

True / False
Explain:

Problem 1d[2pts]: A MicroKernel can improve the resilience of a system against bugs in the OS.

True / False
Explain:

Problem 1e[2pts]: Anything that can be done with a monitor can also be done with semaphores.

True / False
Explain:
Problem 1f[2pts]: “Hyperthreading” is a term used to describe systems with thousands of threads.
   True / False
   Explain:

Problem 1g[2pts]: A Lottery Scheduler can be used to implement any other scheduling algorithm by adjusting the number of tickets that each process holds.
   True / False
   Explain:

Problem 1h[2pts]: A Memory Management Unit (MMU), is a piece of hardware that translates virtual addresses into physical addresses.
   True / False
   Explain:
Problem 2: Short Answer [22pts]

Problem 2a[2pts]: What is priority inversion and why is it an important problem? Present a priority inversion scenario in which a lower priority process can prevent a higher-priority process from running.

Problem 2b[3pts]: Some operating systems do not enforce memory isolation between processes. Describe two advantages and one disadvantage of not enforcing memory isolation.

Problem 2c[2pts]: What happens when an interrupt occurs? What does the interrupt controller do?

Problem 2d[2pts]: Describe one key difference between a monolithic kernel and a modular kernel and one key difference between a modular kernel and a micro kernel.
Problem 2e[2pts]: Why is it necessary to enter Linux via a system call as opposed to a function call?

Problem 2f[2pts]: The Lab 2 handout demonstrates how x264 runs slower when it is configured to run more threads than the number of physical CPUs. What are two sources of this slowdown?

Problem 2g[3pts]: What is an ExoKernel? What functionality is placed in a LibraryOS and how does this functionality differ from the functionality typically placed into the C library?

Problem 2h[2pts]: Unlike other operating systems, Linux can utilize the same system call to generate a new process and to generate a new thread. What is this system call and how does one distinguish between different options?
Problem 2j[4pts]: Cucumber is a domain-specific language (DSL) embedded in Ruby. Explain (with a diagram) how you were able to write BDD tests in Cucumber to test components of your operating system (running in a virtual machine container). Give details about which languages things are in (such as the Step definitions) and what sort of protocol runs between components.
Problem 3: Lock-Free Queue [20 pts]

An object such as a queue is considered “lock-free” if multiple processes can operate on this object simultaneously without requiring the use of locks, busy-waiting, or sleeping. In this problem, we are going to construct a lock-free FIFO queue using an atomic “swap” operation. This queue needs both an Enqueue and Dequeue method.

We are going to do this in a slightly different way than normally. Rather than Head and Tail pointers, we are going to have “PrevHead” and Tail pointers. PrevHead will point at the last object returned from the queue. Thus, we can find the head of the queue (for dequing). Here are the basic class definitions (assuming that only one thread accesses the queue at a time). We will use Java as a slightly higher-level language for our description:

```java
// Holding cell for an entry
class QueueEntry {
    QueueEntry next = null;
    Object stored;
    int taken = 0;

    QueueEntry(Object newobject) {
        stored = newobject;
    }
}

// The actual Queue (not yet lock free!)
class Queue {
    QueueEntry prevHead = new QueueEntry(null);
    QueueEntry tail = prevHead;

    void Enqueue(Object newobject) {
        newEntry = new QueueEntry(newobject);
        tail.next = newEntry;
        tail = newEntry;
    }

    Object Dequeue() {
        QueueEntry curPrevHead = prevHead;
        QueueEntry nextEntry = curPrevHead.next;

        if (nextEntry == null) {
            return null;
        }

        prevHead = nextEntry;
        return nextEntry.stored;
    }
}
```
Problem 3a[5pts]:
Suppose that we have an atomic swap instruction. This instruction takes a local variable (register) and a memory location and swaps the contents. Although Java doesn’t contain pointers, we might describe this as follows:

```java
Object AtomicSwap(variable addr, Object newValue) {
    object result = *addr; // get old object stored in addr
    *addr = newValue; // store new object into addr
    return result; // Return old contents of addr
}
```

Rewrite code for `Enqueue()`, using the `AtomicSwap()` operation, such that it will work for any number of simultaneous Enqueue and Dequeue operations. You should never need to busy wait. **Do not use locking (i.e. don’t use a test-and-set lock).** The solution is tricky but can be done in a few lines. **Hint:** during simultaneous insertions, objects may be temporarily disconnected from the queue (i.e. the set of entries reachable by `nextEntry = nextEntry.next` starting from `prevEntry` may not include every entry); but eventually the queue needs to be connected.

Please complete the following sketch. We will be grading on conciseness; Solutions with more than 5 lines will not be accepted for full credit (you need far less):

```java
void Enqueue(Object newobject) {
    newEntry = new QueueEntry(newobject);
    // Missing code here:
}
```

Problem 3b[2pts]: Provide an informal explanation of why your solution works for simultaneous enqueue operations:
Problem 3c[5pts]:
Suppose that we have a compare and swap instruction. This instruction takes two local variables (registers) and a memory location and stores the second register into memory as long as the first register matches the current contents of memory. Although Java doesn’t contain pointers, we might describe this as follows:

```java
Boolean CAS(variable addr, Object oldValue, Object newValue) {
    object curValue = *addr; // get old object stored in addr
    if (curValue == oldValue) {
        *addr = newValue; // store new object into addr
        return true;  // Success!
    } else {
        return false;  // Failure!
    }
}
```

Rewrite code for Dequeue() using the CAS instruction such that it will work for any number of simultaneous threads working at once. Your solution should dequeue items in the same order that they were originally enqueued. **Again, do not use locking.** There may be short-lived busy waiting during simultaneous dequeue operations. The solution is tricky but can be done by modifying a small number of lines. *Hint: why did we save prevHead into curPrevHead?*

Please complete the following code sketch. We will be grading on conciseness: Solutions with more than 10 lines additional lines will be rejected (need many fewer!)

```java
Object Dequeue() {
    // Missing code here:
    QueueEntry curPrevHead = prevHead;
    QueueEntry nextEntry = curPrevHead.next;

    if (nextEntry == null) {
        return null;
    }

    // Missing code here:

    return nextEntry.stored;
}
```
Problem 3d [2pts]: Provide an informal explanation of why your solution for dequeue() works in the presence of simultaneous enqueue() and dequeue() operations:

Problem 3f [4pts]: What is an MCS lock? Name two reasons why an MCS lock is better than a simple spin lock (say a Test&Test&Set lock) under high contention. When might an MCS lock be less desirable than a simple spin lock?

Problem 3g [2pts]: What are reactive synchronization algorithms? *Hint: one example could have something to do with your answer to (3f).*
Problem 4: Scheduling [24pts]

**Problem 4a[2pts]:** Vanilla Linux provides 100 static priority levels for realtime processes and Kernel threads. Explain how you would use this functionality to perform RMS scheduling of realtime tasks. Would there be a limit to how many tasks could be simultaneously scheduled?

**Problem 4b[2pts]:** Explain the mechanism by which the O(1) scheduler attempted to enhance the user experience with interactive applications. Give details.

**Problem 4c[2pts]:** Explain how the CFS scheduler would schedule 4 CPU-bound processes with nice value +10, Targeted Latency = 20ms, and Minimum Granularity = 1ms. What is an important difference about how the O(1) scheduler would schedule the same 4 processes (give qualitative answer, do not need exact numbers for this latter question).

**Problem 4d[3pts]:** Explain one similarity and one difference between the behavior that users would see for processes within a scheduling group in CFS and for jobs within a CBS. How would other processes or jobs in the system affect this behavior?
**Problem 4e[2pts]**: What is two-level scheduling? (i.e. as in the Tessellation Operating system)?

**Problem 4f[3pts]**: The CBS paper provides the following prescription of how to handle the arrival of a new job when the CBS queue is empty:

*When a job $J_{i,j}$ arrives and the server is idle, if $c_s \geq (d_{s,k} - r_{i,j})$ Us the server generates a new deadline $d_{s,k+1}=r_{i,j} + T_s$ and $c_s$ is recharged to the maximum value $Q_s$, otherwise the job is served with the last server deadline $d_{s,k}$ using the current budget.*

Here, $U_s$ is the server utilization ($Q_s/T_s$). Explain the three domains of operation here (i.e. why they make sense): (1) $c_s > (d_{s,k} - r_{i,j}) U_s$, (2) $c_s = (d_{s,k} - r_{i,j}) U_s$ and (3) $c_s < (d_{s,k} - r_{i,j}) U_s$.

**Problem 4g[2pts]**: Here is a table of realtime periodic tasks with their associated arrival times, computational times/activation and periods.

<table>
<thead>
<tr>
<th>Task</th>
<th>Arrival Time</th>
<th>CPU Usage per period</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task A</td>
<td>0</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Task B</td>
<td>0</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Task C</td>
<td>0</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>Task D</td>
<td>0</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>

Would it be possible to schedule these jobs with RMS? How about EDF? Explain.
**Problem 4h[8pts]:** Show the scheduling order for these processes given in (4h) under 3 policies: Round-Robin (RR) with timeslice quantum = 1, Rate Monotonic Scheduling (RMS), and Earliest Deadline First (EDF). *For RR, assume that context switch overhead is 0 and that each period adds runtime to tasks still on the run-queue and adds unscheduled tasks to the back of the queue between time slots.* We filled out the first line for you. Don’t forget the number of missed deadlines at bottom:

<table>
<thead>
<tr>
<th>Time Slot</th>
<th>Newly Scheduled</th>
<th>RR: Q=1</th>
<th>RMS</th>
<th>EDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>ABCD</td>
<td>A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Missed Deadlines*
Problem 5: Potpourri [18pts]

In Lab 2 you are implementing a kernel snapshotting interface that is designed to allow you to easily introspect your kernel from userspace code. In case you don't remember, the prototype for the snapshot() library routine is as follows:

```c
/*
 * Generates a set of snapshots: These snapshots end up in statically
 * allocated kernel buffers, so there is a maximum number of events you can
 * ask for at once.
 *
 * events An array of length "n" of the events to trigger on
 * triggers An array of length "n" of the trigger types
 * n The length of those arrays
 *
 * return 0 on success, -1 on failure. This call is non-blocking
 *
 * EINVAL The kernel cannot take "n" snapshots at once.
 */
int snapshot(enum snap_event *events, enum snap_trig *triggers, size_t n);
```

Problem 5a[2pts]: Describe on reason why we chose to use the snapshot interface given in Lab 2, as opposed to testing CBS by using the existing kernel/proc interfaces (for instance manually walking the /proc tree):

Problem 5b[4pts]: Give an example of a Cucumber feature (show code) that could utilize the snapshot interface in order to test that your scheduler was successfully performing EDF scheduling of hard realtime tasks:
Problem 5c[3pts]: The snapshot interface was designed to be easily extensible such that it can be used to test many different types of kernel functionality. Describe how you might extend the given snapshot interface to support testing of the sub-process clone() limits you implemented in Lab 1.

Problem 5d[2pts]: In Lab 1, we tested subprocess clone() limits by creating a dummy application that set the limit, attempted to clone() many times, and verified that the limit was enforced. Describe one advantage that use of your extensions in (5c) would have over this previous testing methodology:

Problem 5e[2pts]: Suppose we have 2 users of a cloud system which contains 20 CPUs and 200GB of memory. Suppose that their job requirements are as follows:

<table>
<thead>
<tr>
<th>User Name</th>
<th>CPU Requirements</th>
<th>Memory Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>User A</td>
<td>1 CPU</td>
<td>20 GB</td>
</tr>
<tr>
<td>User B</td>
<td>6 CPU</td>
<td>20 GB</td>
</tr>
</tbody>
</table>

How many jobs for User A and User B will be granted if we have a DRF policy? How much resource of each type will be unused with this allocation?
Problem 5f [2pts]: Suppose you build a device driver that receives characters via an interrupt (such as a serial port) and places them on a FIFO queue in memory for later retrieval by applications. Explain (1) why you need to have some sort of synchronization between the device driver and system call code that retrieves information from the FIFO, and (2) explain why you cannot use a semaphore for this purpose.

Problem 5g [3pts]: You are trying to find Waldo! You've managed to do all sorts of fancy image processing magic such that you've identified each person, the color of their hat, and their position on the screen. This data has been placed into the following linked list, where "struct list_head link" uses the Linux kernel's linked list implementation:

```c
struct person {
    struct list_head link;
    u8 hat_r, hat_g, hat_b;
    u16 posn;
}
```

Write a method that searches a given list of people for the person whose hat color matches the given person's hat color (given as a tuple of red, green, blue). This function should return the position of the matching person, or "-1" on failure. Here is a prototype for one possibly useful macro:

```c
 /**
 * list_for_each_entry-iterate over list of given type
 * @pos:    the type * to use as a loop cursor.
 * @head:   the head for your list.
 * @member: the name of the list_struct within the struct.
 */
#define list_for_each_entry(pos, head, member)
```

Place your code here:

```c
u16 where_is_waldo(struct person *list, struct person *to_match) {
```
[Scratch Page: Do not put answers here!]
[Scratch Page: Do not put answers here!]