CSC 469H1 F / CSC 2208H1F
ADVANCED OPERATING SYSTEMS

UNIVERSITY OF TORONTO
Fall 2007

Term Test #1

NO AIDS ALLOWED

Please PRINT in answering the following requests for information:

Family Name: ____________________________________________

Given Names: ____________________________________________

Student Number: |___|___|___|___|___|

Login (@cdf): ____________________________

Notes to students:
1. This test lasts for 110 minutes and consists of 78 marks. Budget your time accordingly.
2. This test has 7 questions and 11 pages (including this one); Check that you have all pages before starting.
4. Write your answers on this “question and answer” paper, in the spaces provided. Be concise. In general, the amount of space provided is an upper bound on the “size” of answer that is expected. If necessary, use space where available and provide explicit pointers.
5. State your assumptions and show your intermediate work, where appropriate.
6. Do not go beyond here until instructed to do so. Write your student number at the top of each succeeding page once you get going.

<table>
<thead>
<tr>
<th>Question</th>
<th>Marks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/15</td>
</tr>
<tr>
<td>2</td>
<td>/12</td>
</tr>
<tr>
<td>3</td>
<td>/12</td>
</tr>
<tr>
<td>4</td>
<td>/12</td>
</tr>
<tr>
<td>5</td>
<td>/12</td>
</tr>
<tr>
<td>6</td>
<td>/6</td>
</tr>
<tr>
<td>7</td>
<td>/9</td>
</tr>
<tr>
<td>Total</td>
<td>/78</td>
</tr>
</tbody>
</table>

../continued
1. [15 marks, 3 each] Definitions
Define (or explain) the following terms, in the context of this course:

a) End-to-end argument

b) Hosted virtual machine

c) Signal delivery

d) test-and-test-and-set

e) Backfilling
2. [12 marks; 6 each] Structure and Performance

Operating system primitives (such as system calls and trap handling) generally require architecture support, however, the performance improvements for these primitives from new hardware rarely match those seen for application code. At the same time, improvements in operating system structure often come at the price of increased overheads for applications with the result that new operating systems are slower than old ones. The data in the two tables below is extracted (and simplified) from a 1991 paper by Anderson et al. that explored these two trends:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Avg. Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System Primitives</td>
<td>3.9</td>
</tr>
<tr>
<td>Application Performance</td>
<td>6.7</td>
</tr>
</tbody>
</table>

Table 1: Average Speedup of CPU2 (newer) over CPU1 (older) on operating system primitives, and on application code.

<table>
<thead>
<tr>
<th></th>
<th>Time (sec.)</th>
<th>Address Space switches</th>
<th>System Calls</th>
<th>Emulated Test-and-Set</th>
<th>% Time In OS Primitives</th>
</tr>
</thead>
<tbody>
<tr>
<td>Monolithic</td>
<td>69.3</td>
<td>2336</td>
<td>5513</td>
<td>320</td>
<td>N/A</td>
</tr>
<tr>
<td>Microkernel</td>
<td>80.9</td>
<td>16208</td>
<td>16561</td>
<td>213781</td>
<td>5%</td>
</tr>
</tbody>
</table>

Table 2: Performance and number of operating system events for latex formatting a 150 page document on CPU2 with monolithic and microkernel OS. Time in OS primitives does not include time executing the system calls themselves, just the overhead of making them. The microkernel adds about 17% to the run time.

a) Suppose I have just replaced my uniprocessor desktop system (running the monolithic OS on CPU1) with a new uniprocessor system running the microkernel OS on CPU2. Suppose also that I spend most of my computing time formatting 150 page latex documents, and am annoyed to discover that my application speedup is only 5.6. Which of the two trends noted above is likely to be the largest contributor to reduced speedup, and why? Base your argument on the data in Tables 1 & 2 and Amdahl’s Law.
b) Explain the differences in (i) numbers of address space switches, (ii) system calls, and (iii) emulated test-and-set instructions shown in Table 2 based on the structural differences between microkernel and monolithic operating systems. (CPU2 did not provide a hardware atomic test-and-set instruction, it had to emulated in the kernel instead.)
3. [12 points] Performance Evaluation & Benchmarking

a) [6 points] For portability, lmbench uses the gettimeofday system call to obtain timing measurements. However, different systems provide different resolutions for gettimeofday, from 1 microsecond up to 10 milliseconds, and there is no portable system call to obtain the resolution. Write a function that experimentally determines the resolution of gettimeofday, returning the result in microseconds.

```c
struct timeval {
    long tv_sec; /* seconds */
    long tv_usec; /* microseconds */
};
/* simplified prototype for gettimeofday */
int gettimeofday(struct timeval *tv);

long get_resolution() {
}
```

b) [3 points] Why is it important to know the timer resolution when conducting a performance experiment?

c) [3 points] Aside from timer resolution, what else should you be careful of when using gettimeofday to measure elapsed time?
4. [12 points; 4 each] Interrupts & IPC

a) With respect to BSD Unix (pre FreeBSD 5.2), identify two limitations of executing in “interrupt context”, and explain how interrupt handlers deal with these limitations.

b) Explain how FreeBSD determines to which thread a signal should be posted in a multi-threaded process.

c) Identify (i) why select is not usable for event notification when large numbers of descriptors are involved, (ii) the primary problem with using poll instead, and the general solution to this problem as implemented by Linux epoll, Solaris /dev/poll, or FreeBSD kqueue.
5. [12 points] Read-Copy Update

a) [2 points] Suppose we have an implementation of linked lists using read-copy update, in which threads acquire a spinlock when writing, but readers can proceed without acquiring any locks. QSBR ensures that memory is reclaimed safely. Is this implementation non-blocking? Why or why not?

b) [1 point] Suppose that the spinlock is replaced with a reader-writer lock. Is this new implementation non-blocking? Why or why not?

c) [1 point] Assume in the picture below that prior to time $t_I$, no thread has gone through a quiescent state. Black boxes represent quiescent states. Identify as a time interval $[a,b]$ the smallest grace period in the system (assume that quiescent states happen instantaneously at the times marked).
d) [4 points] Below is an incorrect implementation of an insertion function for a doubly-linked list using RCU. The function is given a new node to be linked in between two existing consecutive nodes (prev and next). The caller is responsible for acquiring the necessary writer locks. Explain what is wrong with the existing function, and then write the corrected body of the function. Assume sequential consistency in your answer.

```
void list_add_rcu(
    struct list_head *new,
    struct list_head *prev,
    struct list_head *next)
{
    prev->next = new;
    next->prev = new;
    new->prev = prev;
    new->next = next;
}
```

Corrected body:
```
void list_add_rcu(
    struct list_head *new,
    struct list_head *prev,
    struct list_head *next)
{
    prev->next = new;
    next->prev = new;
    new->prev = prev;
    new->next = next;
}
```

e) [4 points] Suppose we have several reader threads that obtain data from a structure containing many elements. Periodically, a single writer will modify the contents of this structure, and we require that readers observe either all the updates, or none of them. To avoid locking we keep two copies of the structure in an array (named copies); a variable (named active) stores the index of the active copy used by readers, while the writer updates the inactive copy and then updates the active variable so that readers will start using the new version of the structure. Code for the writer is shown below:

```
void update_structure(int newval1, int newval2, int newval3)
{
    /* get index of inactive copy */
    int inactive_index = 1 - active;
    /* update fields in inactive copy */
    copies[inactive_index].field1 = newval1;
    copies[inactive_index].field2 = newval2;
    copies[inactive_index].field3 = newval3;
    /* make inactive copy the new active copy */
    active = 1-active;
}
```

If we do not assume sequential consistency, show the fence(s) that are needed on the code above (identify the position and type of fence), and explain why they are needed.
6. [6 points] Transactional Memory

Below is the code for the push and pop operations on a stack that uses transactional memory (TM). Show that the TM stack does not suffer from the ABA problem that was demonstrated for the simple non-blocking stack implementation in the lecture. Assume T1 is doing a pop() and is interrupted just before it commits. Show the contents of the read and write sets for T1, and feasible values for the relevant locations in memory at the time that the interrupted thread tries to commit as part of your answer.

```c
typedef struct node_s {
    int val;
    struct node_s *next;
} node_t;

node_t *S; /* the stack */

void push(node_t **stack, node_t *n) {
    atomic {
        n->next = *stack;
        *stack = n;
    }
}

node_t * pop(node_t **stack) {
    node_t *result = NULL;
    atomic {
        result = *stack;
        *stack = *stack->next;
    }
    return result;
}

T1:
...
node_t *my_node = pop(&S);
...
```

Initial stack. The notation M[X] indicates the address of each item. For example, node C in the stack has address Y.

```
S     A     C     B
```
7. [9 points: 3 each] Multiprocessor Scheduling
(a) Briefly discuss the tradeoff between load balancing and processor affinity in multiprocessor scheduling.

(b) Explain why different scheduling techniques are needed for parallel jobs vs. a standard workload with large numbers of threads.

(c) Describe the EASY scheduling algorithm, and identify one problem with it.
Total marks = (78)
End of test