Final Exam
May 11, 2012
CS162 Operating Systems

General Information:
This is a closed book and TWO 2-sided handwritten notes examination. You have 170 minutes to answer as many questions as possible. The number in parentheses at the beginning of each question indicates the number of points for that question. You should read all of the questions before starting the exam, 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. If there is something in a question that you believe is open to interpretation, then please ask us about it!

Good Luck!!

<table>
<thead>
<tr>
<th>QUESTION</th>
<th>POINTS ASSIGNED</th>
<th>POINTS OBTAINED</th>
</tr>
</thead>
</table>
1. (21 points total) Short answer questions.
   a. (12 points) True/False and Why? CIRCLE YOUR ANSWER.
      i) The size of a process’ Working Set may depend on the number of processes currently running.
         TRUE
         Why?
         FALSE
         Why?
   
   ii) Doubling the block size in the UNIX file system will double the maximum file size.
         TRUE
         Why?
         FALSE
         Why?
   
   iii) Adding a memory cache never hurts performance.
         TRUE
         Why?
         FALSE
         Why?
   
   iv) Address translation (virtual memory) is useful even if the total size of virtual memory as summed over all possible running programs is guaranteed to be smaller than a machine’s total physical memory.
         TRUE
         Why?
         FALSE
         Why?
b. (4 points) List two reasons why operating systems like Nachos disable interrupts when a thread/process sleeps, yields, or switches to a new thread/process?

c. (2 points) Give a definition for a non-blocking write operation.

d. (3 points) Explain what causes thrashing in a system that uses virtual memory management AND how to reduce the likelihood of thrashing occurring.
2. (19 points total) Networking.

a. (5 points) If the TCP transmission window for unacknowledged bytes is 1000 bytes, the one-way latency of a cross-country network link is 50 milliseconds, and the bandwidth of the link is 100 Megabits/second, then how long does it take TCP to transmit 100,000 bytes across the link? That is, how much time elapses from when the first byte is sent by the sender to when the sender knows that the receiver has received the last byte? You may ignore the TCP handshake and assume that no packets are lost for this particular problem (but remember that TCP doesn’t know that) and that ACKs are essentially 0 bytes long. Write out your answer in symbolic form for partial credit.

b. (6 points) Consider a distributed system with two communication primitives: SEND and RECEIVE. The SEND primitive sends a message to a specified process on a remote machine. The RECEIVE primitive specifies a remote process to receive from, and blocks if no message from that remote process is available, even though messages may be waiting from other remote processes. There are no shared resources, but processes need to communicate frequently. Do the primitives allow deadlock? Discuss why or why not. (Use no more than two sentences.)
c. (8 points). Your new boss at Orange Computer proposes a new sliding window-based reliable network transport protocol.

The protocol works as follows: instead of sending an acknowledgement for every packet, it only sends an acknowledgement for every 5 packets (by sequence number). Thus, ACK #1 is sent when packets with sequence numbers 0-4 have been received. ACK #2 is sent when packets with sequence number 5-9 have been received.

Acknowledgements do not cover packets from earlier sequence numbers and can be sent out of order (i.e., ACK #2 can be sent when packets with sequence numbers 5-9 have been received successfully, even if packets with sequence numbers 0-4 have not been received). The protocol also uses a retransmission timer for each packet sent.

List a one sentence advantage and one sentence disadvantage of using this new protocol.

i). Advantage:

ii). Disadvantage:
3. (24 points total) Shared Memory.

In this problem you will outline an implementation of shared memory in an operating system like Nachos. Shared memory will be implemented using two new system calls:

- RegionID SharedRegionCreate(int beginAddress, int endAddress)
- void SharedRegionAttach(RegionID region);

**SharedRegionCreate** creates a shared memory region in the caller’s address space defined by the begin and end addresses. *The addresses must be page-aligned.* If successful, it returns a RegionID identifying the shared region. **SharedRegionAttach** maps the existing shared region identified by the RegionID into the caller’s address space. A region can only be created by one process, but it can be attached by an arbitrary number of processes. Regions must be non-overlapping.

You can assume that each PTE has an additional flag Shared, which should be true if that virtual page is being shared. You can also assume that a shared memory region can only be created within a defined region of a process’ virtual address space. These operations can be used as follows. A parent process uses **SharedRegionCreate** to create a shared memory region, and then **Exec**’s a child process and passes the returned RegionID for the shared region to the child as an argument to **Exec**. The child then uses **SharedRegionAttach** to map that region into its address space so that it can share memory with its parent. As a result, whatever data the parent places and modifies in the shared region, the child can access and modify (and vice versa).

Answer each of the following questions descriptively *at a high level*. Your answers should be brief, and capture the essence of what needs to be implemented at each stage. *You do not need to provide pseudo-code.*

a. (5 points) What should SharedRegionCreate do to the caller’s page table and any other system data structures? For part a, you may assume that all the shared pages are in memory.

b. (4 points) What should SharedRegionAttach do to the caller’s page table and any other system data structures?
c. (4 points) What additional page table operations must be performed when a shared page is paged out (evicted) from physical memory?

d. (3 points) What additional page table operations must be performed when a shared page is paged in from backing store?

e. (2 points) How should the process of destroying an address space change to account for shared pages?

f. (3 points) List THREE error conditions that implementations of SharedRegionCreate and SharedRegionAttach will have to check for.

g. (3 points) What changes would your solution need if a shared region was shared by more than two processes (e.g., a parent and two child processes).
4. (12 points total) Two-Phase Commit.
You’re hired by Macrohard Corporation to develop a system for distributing updates to their operating system. To ensure proper operation, all client systems must be running the same version of the software. New OS versions are first distributed to clients using a Content Distribution Network. Next, two-phase commit is used in order to ensure that, despite computer crashes, either (a) everyone eventually switches to the new OS version identified by the version number, NEWVER, or (b) no one switches to the new OS version and everyone continues using OLDVER. Assume that a machine takes some significant amount of time to reboot and recover after a crash.

The participants in this problem are the OS Version Server (OSVS) and three client nodes (C1, C2, C3). The steps of the two-phase commit protocol are listed below, in time order:
1. OSVS: write “begin transaction” to its log
2. OSVS → C1: “New version is NEWVER.” (→ means OSVS sends message to C1)
3. C1: write “New version is NEWVER.” to its log
4. C1 → OSVS: “Prepared to commit”
5. OSVS → C2: “New version is NEWVER.”
6. C2: write “New version is NEWVER.” to its log
7. C2 → OSVS: “Prepared to commit”
8. OSVS → C3: “New version is NEWVER.”
9. C3: write “New version is NEWVER.” to its log
10. C3 → OSVS: “Prepared to commit”
11. OSVS: write “New version is NEWVER.” to its log
12. OSVS: write “commit” to its log
13. OSVS → C1: “commit”
14. C1: write “got commit” to its log
15. C1: OS Version = NEWVER
16. C1 → OSVS: “ok”
17. OSVS → C2: “commit”
18. C2: write “got commit” to its log
19. C2: OS Version = NEWVER
20. C2 → OSVS: “ok”
21. OSVS → C3: “commit”
22. C3: write “got commit” to its log
23. C3: OS Version = NEWVER
24. C3 → OSVS: “ok”
25. OSVS: OS Version = NEWVER

a. (3 points) If OSVS crashes after step 11 and no one else fails, what OS Version will everyone end up using, once OSVS reboots and recovers? Give the reason why.
b. (3 points) If C3 crashes after step 9 and no one else fails, what key will everyone end up using, once C3 reboots and recovers? *Give the reason why.*

c. (3 points) If C3 crashes after step 12, what recovery steps must it take after it reboots in order to achieve the correct global state with respect to which OS Version to use?

d. (3 points) If OSVS crashes after step 23, what recovery steps must it take after it reboots in order to achieve the correct global state with respect to which OS Version to use?
5. (12 points total) Caching.
Consider a computer with 8-bit addresses that uses an LRU (Least Recently Used) cache consisting of four 4-byte blocks. Assume a program that accesses the memory according to the following sequence: 0x20, 0x45, 0x11, 0x44, 0x20, 0x70, 0x71, 0x72, 0x10, 0x44, 0x32, and 0x12. Initially, the cache is empty.

a) (4 points) Assume a direct mapped cache. Show the size (in bits) of the cache tag, cache index, and byte select fields. In addition, show the content of every cache block after each memory access: each entry of the table below should list all the addresses of the bytes that are cached in the corresponding block.

<table>
<thead>
<tr>
<th>cache tag</th>
<th>cache index</th>
<th>byte select</th>
</tr>
</thead>
</table>

Cache content

<table>
<thead>
<tr>
<th>Block</th>
<th>0x20</th>
<th>0x45</th>
<th>0x11</th>
<th>0x44</th>
<th>0x20</th>
<th>0x70</th>
<th>0x71</th>
<th>0x72</th>
<th>0x10</th>
<th>0x44</th>
<th>0x32</th>
<th>0x12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

b) (4 points) Repeat question (a) for a fully associative cache.

<table>
<thead>
<tr>
<th>cache tag</th>
<th>byte select</th>
</tr>
</thead>
</table>

Cache content

<table>
<thead>
<tr>
<th>Block</th>
<th>0x20</th>
<th>0x45</th>
<th>0x11</th>
<th>0x44</th>
<th>0x20</th>
<th>0x70</th>
<th>0x71</th>
<th>0x72</th>
<th>0x10</th>
<th>0x44</th>
<th>0x32</th>
<th>0x12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
c) (4 points) Repeat question (a) for a 2-way associative cache. For each block indicate the set is belonging to.

<table>
<thead>
<tr>
<th>cache tag</th>
<th>cache index</th>
<th>byte select</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Cache content

<table>
<thead>
<tr>
<th>Set</th>
<th>Block</th>
<th>0x20</th>
<th>0x45</th>
<th>0x11</th>
<th>0x44</th>
<th>0x20</th>
<th>0x70</th>
<th>0x71</th>
<th>0x72</th>
<th>0x10</th>
<th>0x44</th>
<th>0x32</th>
<th>0x12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

6. (12 points total) Circular queue and concurrency.
Circular queue is a data structure often used to implement various functionalities in the OS. For example, the sending and receiving buffers in TCP are implemented using circular queues. Consider a circular queue of size N. The figure below shows one instance of a circular queue for N=8, where the head points to entry 1, and the tail to entry 4. The head points to the first element in the queue, while the tail to the last element in the queue. If the queue is empty, both head and tail are initialized to -1. Note that both the head and tail are incremented using modulo N operator. The pseudocode below implements the enqueue() operation. We assume that both these functions are synchronized, i.e., they lock the queue object when executing.

```cpp
// enqueue new item "data"; return false if queue full synchronized bool enqueue(item data) {
    if (head == -1) { // queue is empty
        head = tail = 0;
        queue[tail] = data;
    } else { // there is at least one item in the queue
        if ((tail + 1) modulo N != head) {
            tail = (tail + 1) modulo N;
            queue[tail] = data;
        } else { // queue full
            return false;
        }
    }
    return true;
}
```

Figure: A circular queue of size N=8 storing 4 entries at positions 1, 2, 3, and 4, respectively.
a) (4 points) Write the pseudocode for the `dequeue()` operation. `dequeue()` should return the item at the head of the queue if queue not empty, and `null`, if queue is empty.

```java
synchronized item dequeue() {
```

b) (2 points) Assume N = 8, and assume there were 100 `enqueue()` operations, out of which 5 have failed, and 98 `dequeue()` operations, out of which 7 have failed. Assume initially the queue is empty, and it never gets empty again after the first item is inserted. What are the values of `head` and `tail` after all `enqueue()` and `dequeue()` operations take place?
c) (6 points) Assume circular queue of integers with N = 3, and assume each of the following three operations is called by a different threads. In particular, thread T1 calls enqueue(10), thread T2 calls enqueue(12), while thread T3 calls dequeue(). Initially, the circular queue is empty. Please specify what are the possible entries in the circular queue after all three operations are performed, and what is the value returned by the dequeue() operation. Use the following table to show your answer. (If a queue entry is outside [head:tail], then use “-“ to represent its content.)

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>