Final Exam Key
CSE 451
Fall 2006
Version A/B

Name:

Short Answer (worth 4 points each)

1) Suppose (for some strange reason) you were asked to exhaust all the inodes on a system. How might you do this?

Create a lot of small files or empty files

2) What is an advantage of a service-oriented architecture for distributed systems?

Modularity, simplified development, easier to expose services to third-party developers.

3) Larger page sizes tend to help TLB hit rates because the TLB maps a larger range of memory. However, this is not always the case. Can you describe a workload for which larger page sizes would decrease the TLB hit rate?

Any workload with poor sequential locality

4) Why is a TLB necessary, given that all page translation information is stored in the page tables? Be sure to describe what would happen if the system did *not* use a TLB.

The TLB is a cache; it ensures that address translation does not slow down the processor. Without a TLB, memory access time would be significantly worse: a single memory access would transform into three memory accesses (for two-level page tables).

5) Round-robin is a commonly used strategy for CPU scheduling across a set of competing processes (or threads). Is round-robin a good strategy for scheduling the disks requests of a set of competing processes? Why or why not?

Round-robin is not a good disk scheduling algorithm. This is because processes tend to access files (and hence the disk) sequentially. Round-robin would break up sequential accesses, resulting in much worse performance.

6) In the worst case, how many disk accesses are required to find the inode number for the following file:

Read inode data for /
Read directory for /
Read inode data for homes
Read directory for homes
Read inode data for andrew/
Read directory data for andrew/

[Only in test version A]
Read inode data for super-secret/
Read directory data for super-secret/

8 (Version B: 6)

7) Repeat problem 5, except that the link is symbolic instead of hard.

Read inode data for /
Read directory for /
Read inode data for homes
Read directory for homes
Read inode data for andrew/
Read directory data for andrew/

[Only in version A]
Read inode data for super-secret/
Read directory data for super-secret/

Read inode data for final-solution.pdf
Read disk block for symbolic link
Read inode data for ak/
Read directory data for ak/

12 (Version B: 10)

8) Suppose we have two identical computers, each running the Clock algorithm for page frame replacement. For computer A, the algorithm inspects an average of 25 page frames. For computer B, the algorithm inspects an average of 50 page frames. Which computer is closer to thrashing?

Version A: Computer B,
Version B: Computer A,

The clock algorithm skips a page when it has been referenced. Therefore, more memory is in use in the answer.

9) In class, we discussed how a virtual machine monitor supports multiple "guest" operating systems by efficiently cloning the underlying hardware architecture. In some cases, a VMM can (and does) expose a different virtual architecture than the underlying physical architecture:

9a) [5 points] Is it possible for the VMM to expose a different Ethernet device in the virtual architecture? For example, could the VMM expose an Ethernet card made by 3Com when the physical machine's Ethernet card is made by Intel? Explain your answer

Yes, this is possible. All interactions with the an I/O device are privileged operations that force a trap. Therefore, it is just as easy to emulate one Ethernet card as it is another.

9b) [5 points] Is it possible for the VMM to expose a virtual instruction set architecture that contains more registers than the underlying physical architecture? Explain your answer.

This is impossible to do with tolerable performance. Normally, a VMM allows most normal instructions to pass through to the underlying native hardware. Changing the number of registers would require interposing on an enormous number of machine instructions, resulting in very bad performance.

It is possible, however, to "compile" one architecture down to another. Answers like this were given full credit.

10) Consider a single web server that serves a large number of equally popular web pages. After benchmarking the web server, we learn that it can serve 100 req/sec.

Now suppose we add a second identical web server. A load balancer distributes traffic across these two servers.

10a) [6 points] After benchmarking the two web server configuration, we learn that it has a throughput of 10,000 requests per second for the same set of web pages. This is a 100X speedup over the single machine configuration. Can you explain how this is possible? Be sure to describe the behavior of the load balancer.

The web pages (or "working set") of the web server fits in the memory of two machines, but not in the memory of one machine. Therefore, a single web server thrashes, whereas a two machine configuration does not. To make this possible, the load balancer must be "locality aware". In other words, it always sends requests for a given web page to the same back-end server.

10b) [4 points] Suppose we further increased the server farm to include four web servers. Would you expect another 100X speedup? Why or why not?

No. The transition from a thrashing to a non-thrashing state is a one-time phenomenon. Once we have enough memory for the working set, additional memory provides no benefit. Therefore, any additional servers will provide (at most) a linear speedup.

11) [8 points] The C function `bzero` fills a memory buffer with zeroes:

```
// fill this memory buffer with zeroes
void bzero(void* buffer,long bufferSize);
```

Suppose you are asked to write an efficient `bzero` implementation. The programmer wants the ability to `bzero` large extents of memory. However, she does not know up front how much of the memory buffer will actually be used. In some cases, very little of the buffer will be used. How would you design such a `bzero` facility? You should provide enough detail so that a student outside this class could implement your design. You can use pseudocode, a detailed description, or a combination.

A correct answer should have the following components: We can use the paging hardware to lazily fill virtual memory pages with zeroes; But, we must eagerly bzero the ends of the buffer region if they are not properly page-aligned. We can't do anything fancy if the buffer region is smaller than a page. To use the paging hardware, we mark all complete virtual memory pages as invalid. Inside the page fault handler, we bzero a page and return control to the user program.

12)

Get rid of all condition variable references
Create two semphores: one for free slots, the other for available buffers
Initialize these with capacity and zero, respectively
Invoke down() on the freeSlots sempahore before executing a put()
  (do this regardless of isFull status())

Invoke up() on the availableBuffers semaphore after executing the put()

Invoke the corresponding semaphore operations for the take() method
  invoke down on the availableBuffers semaphore regardless of isEmpty status
  invoke up on freeSlots semaphore taking the value

Remove synchronized keyword from put() and take() methods
Add back smaller synchronized blocks to ensure thread-safe data access

```java
public class BoundedBuffer  {
   private final Object [] buf;
   private int tail;
   private int head;
   private int count;

   private Semaphore freeSlots;
   private Semaphore availableBuffers;

   public BoundedBuffer (int capacity) {
      this.buf =  new Object[capacity];
      freeSlots = new Semaphore(capacity);
      availableBuffers = new Semaphore(0);
   }

   // Add an element to the queue, or block if the queue is full
   public void put(Object v) throws InterruptedException {

      freeSlots.down();

      synchronized (this) {
         buf[tail] = v;
         if (++tail == buf.length)
            tail = 0;
         ++count;
      }

      availableBuffers.up();
   }

   // remove an element from the queue, or block if empty
   public Object take() throws InterruptedException {
      Object v;
      availableBuffers.down();

      synchronized (this) {
        v = buf[head];
        buf[head] = null;
        if (++head == buf.length)
           head = 0;
        --count;
      }

      freeSlots.up();
      return v;
   }

   public synchronized boolean isFull() {
      return count == buf.length;
   }
   public synchronized boolean isEmpty() {
      return count == 0;
    }
}
```