CSE 451
Fall 2006
Sample Exam Questions

* I/O instructions are always privileged instructions (i.e., they can only be executed in kernel mode). Why is this so?

* The book talks about the four conditions for deadlock (mutual exclusion, hold and wait, no preemption, and circular waiting). Explain why these conditions are necessary, but not sufficient for deadlock to occur. Draw a resource allocation graph, which depicts a situation when the four conditions are satisfied, but the system is not deadlocked.

* Give one reason why Linux uses a separate stack for each thread when executing in the kernel?

* An application could damage the system by installing its own interrupt handlers in place of the operating system's interrupt handlers. How does the OS prevent this from happening?

* The OS provides a read system call, which reads from the file system into a user-provided buffer. In Linux, this system call looks like:

ssize_t read(int fileDescriptor, void *buffer, size_t numBytes);

In class, we discussed the need to "verify" the buffer argument. What does this mean? What could go wrong if the kernel did not verify the buffer argument?

* Many parallel algorithms have the following flavor:

```
public void foo() {
   // i) break up the problem into N sub-problems
   // ii) run each sub-problem on its own processor
   // iii) reassemble the N results into one result
}
```

Suppose the computations in step (ii) represent 50% of the total processing time. What is the maximum speedup for this application on a 4 processor machine? On a 1000 processor machine? Justify your answer.

* Suppose you are given the task of parallelizing merge sort to run a dual processor machine. Do you think it is possible to get a 2X speedup? Why or why not?

* Consider the following statement: "In the future, Moore's law will automatically make my program run faster." Do you agree with this? Why or why not?

* Consider the following method to sum up the elements in a list:

```
public class MyClass {
   public int arraySum(int [] array) {
      int sum = 0;
      for (int i=0 ; i < array.length; i++)
         sum += array[i];

      return sum;
   }
}
```

Re-write this class and method to utilize multiple threads

Consider the following Producer/Consumer example:

```
1.  import java.util.List;
2.  import java.util.LinkedList;
3.
4.  class Consumer extends Thread {
5.     private List list;
6.     private Object conditionVariable;
7.
8.     public Consumer (List list,Object condVar) {
9.        this.list = list;
10.       this.conditionVariable = condVar;
11.    }
12.
13.    public void run () {
14.       // We must grab the lock to perform a condition variable operation
15.       synchronized (conditionVariable) {
16.
17.          // This must be a while statement to catch spurious wakeups
18.          while (list.isEmpty()) {
19.             try {
20.                conditionVariable.wait();
21.             }
22.             catch (InterruptedException ie) {
23.                System.err.println("Unexpected exception: " + ie);
24.             }
25.          }
26.
27.          System.out.println("Consuming object: " + list.remove(0));
28.       }
29.    }
30. }
31.
32. class Producer extends Thread {
33.    private List list;
34.    private Object conditionVariable;
35.
36.    public Producer (List list, Object condVar) {
37.       this.list = list;
38.       this.conditionVariable = condVar;
39.    }
40.
41.    public void run () {
42.
43.       list.add("Hello world");
44.
45.       // we must hold the lock to perform condition variable operations
46.       synchronized (conditionVariable) {
47.
48.          // wake somebody up...
49.          conditionVariable.notify();
50.       }
```

```
51.    }
52. }
53.
54. public class ProducerConsumer  {
55.    public static void main (String [] args) {
56.       Object condVar = new Object();
57.       List list = new LinkedList();
58.
59.       Thread consumer = new Consumer(list, condVar);
60.       consumer.start();
61.
62.       Thread producer = new Producer(list, condVar);
63.       producer.start();
64.    }
65. }
```

* The call to `list.add` on line 43 is problematic. Why?

* How would you fix this code?

Java provides two classes for constructing Strings. StringBuffer is thread-safe, whereas String-Builder is not. Consider the following code snippets:

```
// Snippet A
public class SnippetA {
   public String constructString() {
      StringBuffer sb = new StringBuffer();
      sb.append(“Hello”)
      sb.append(“ world”);
      sb.append(“\n”);

      return sb.toString();
   }
}

// Snippet B
public class SnippetB {
   private StringBuffer sb = new StringBuffer();

   public String constructString() {
      sb.append(“Hello”)
      sb.append(“ world”);
      sb.append(“\n”);

      String returnValue = sb.toString();

      // reset the buffer before returning
      sb.delete(0,sb.length());

      return returnValue;
   }
}
```

* Snippet A is:
A. Not thread safe.
B. Not optimal. We can safely replace StringBuffer with StringBuilder
C. Just right. The program as written is thread safe and optimal

* Snippet B is:
A. Not thread safe.
B. Not optimal. We can safely replace StringBuffer with StringBuilder
C. Just right. The program as written is thread safe and optimal

* True or false: On a multi-processor machine, a multi-threaded program always runs faster than a single-threaded programs. Explain your answer.

Consider the multi-threaded web server you wrote as part of project 2.

* True or false: Given current technology trends, the web server will be capable of handling many more requests in the future.

* True or false: Given current technology trends, the web server's per-request latency will be dramatically reduced in the future.

* Older versions of Linux maintained a "big kernel lock", which allowed only a single process or thread to run in the kernel at a given time. Do you think this would have a bigger impact on: 1) The parallel pi calculator; or 2) The multi-threaded web server. Why?

* True or false: It only makes sense to use a multi-threaded web server on a multi-processor machine. Explain your answer.

* True or false: It only makes sense to use a multi-threaded pi calculator on a multi-processor machine. Explain your answer.

* True or false: A stateless object is always thread-safe. Explain your answer.

* True or false: An immutable object is always thread-safe. Explain your answer.

* True or false: An object with nothing but final fields is always thread safe. Explain your answer.

* Name a significant problem associated with layered operating system designs like THE.

* Every Linux semaphore is implemented using a spin lock. Give a reason for this.

* The main method of the Fibonacci GUI is shown below:

```
public static void main(String[] args) {
    javax.swing.SwingUtilities.invokeLater(new Runnable() {
          public void run() {
              createAndShowGUI();
          }
      });
   }
}
```

Why do we need to inovke SwingUtilities.invokeLater to show the GUI? Why can't we directly call createAndShowGUI?

* For a multi-threaded web server, why is it better to use a thread pool rather than creating a new thread for every web request?

* What is the worst-case workload (in terms of turnaround time) for FIFO scheduling?

* What is the worst-case workload (in terms of turnaround time) for round-robin scheduling?

* Which of the following schedulers are guaranteed to be free from starvation: FIFO, shortest-job-first, round-robin.

* Can you locate a safety bug in this bounded buffer implementation? How would you fix it?

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

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

    public synchronized final void put(Object v) throws InterruptedException{
        while (isFull())
            wait();

        buf[tail] = v;
        if (++tail == buf.length)
            tail = 0;
        ++count;

        notify();
    }

    public synchronized final Object take() throws InterruptedException {
        while (isEmpty())
            wait();

        Object v = buf[head];
        buf[head] = null;
        if (++head == buf.length)
            head = 0;
        --count;

        notify();
        return v;
    }

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

* What is a disadvantage of Linux-style kernel modules relative to micro-kernel OS's?

* For which of the following applications would user threads provide some benefit:

1) A word-processor with a heavy-weight spell check operation
2) A multi-threaded merge sort implementation
3) A multi-threaded web server

* Give a reason why Linux 2.6 uses a ready queue per processor, as opposed to a single, shared ready queue.

* The following spinlock implementation is broken (it is not thread safe)? How would you fix it?

```
class SpinLock implements Lock {
   private volatile boolean isLocked = false;

   public void acquire() {
      while (isLocked) { ; } // busy wait
      isLocked = true;
   }

   public void release() {
      isLocked = false;
   }
}
```

* True or false: if any method of a class is synchronized, then all methods of the class should be synchronized. Explain your answer.

* What is the biggest drawback of the micro-kernel approach to operating system construction?

* What do we mean when we say “fork returns twice?”

* What value is printed by this C program:

```
int value = 5;

int main () {
  pid_t pid ;

  value = 7;

  pid = fork();
  if (pid == 0) {
     value += 15;
  }
  else {
    wait (NULL);
    printf("PARENT: value = %d\n",value );
  }
}
```

Consider the following example:

```
1.  public class Rendezvous {
2.
3.     private final Object barObject = new Object();
4.
5.     public synchronized void waitAround () throws InterruptedException {
6.          synchronized (barObject) {
7.             barObject.wait();
8.          }
9.      }
10.
11.    public synchronized void meetUp() {
12.       synchronized(barObject) {
13.          barObject.notify();
14.       }
15.    }
16. }
```

* Which lock(s) are held immediately before the invocation of wait (on line 7)?

* Which lock(s) are held during the invocation of wait?

* Which locks are held after returning from wait?

* Does this class have any thread-safety problems? If so, how would you fix them?

* Does this class have any liveness problems? If so, how would you fix them?

* Define deadlock

* During Linux's file rename operation, Linux must decrement the semaphores for both the old file and the new file. How does Linux avoid deadlock for this operation?

* How might you increase the amount of parallelism in this class?

```
public class MyClass {

  // invariant: x == y
  private int x = 0;
  private int y = 0;

  // invariant: w == z
  private int w = 0;
  private int z = 0;

  // invariant: k == j
  private int k = 4;
  private int j = 4;

  public synchronized void incrementXY() {
    x++;
    y++;
  }

  public synchronized void incrementWZ() {
    w++;
    z++;
  }

  public synchronized void incrementK() {
    k++;
    j++;
  }
}
```

* How is deadlock different than starvation?

* Why are event-driven systems (non-blocking I/O systems) more difficult to program than threaded systems?

* Generally speaking, which type of system devotes more overhead to synchronization operations: threaded systems or event-driven systems (that use non-blocking I/O)?

Consider the following program:

```
public class CountToTen implements Runnable {

   // this counter gets incremented every second
   static int count = 0;
   static final int TARGET = 10;

   public static void main(String[] args) {
      // start the counter thread
      Thread t = new Thread (new CountToTen());
      t.start();

      // wait TARGET seconds
      while (count < TARGET) { ; } // busy wait
      System.out.println(TARGET + " seconds has expired");
   }

   public void run() {
      while (count < TARGET) {
         try {
            Thread.sleep(1000); // sleep for one second
         } catch (InterruptedException e) {} // ignored
         count++;
      }
   }
}
```

* One problem with this program is that it uses busy waiting. Can you identify another concurrency problem with this program? How would you fix it?

* For project #1, we asked you to track statistics about each running process, and propagate these statistics up to the parent process. At the time, we disregarded concern for multi-threaded correctness. What race conditions could this create? How would you propose fixing them?

* For project #1, we asked you to add an execcounts command to the shell. This command reports the number of fork/exec/clone system calls made by a child process. Because the Linux task_struct disappears when a task completes, most solutions involve checking the shell's syscall counts before and after the child process runs.

Suppose we extended the shell with a command for running two programs in parallel:

parallel_execounts (prog1 arg arg arg ...) (prog2 arg arg arg ...)

Describe how we could accurately report separate syscall count statistics for both programs.