

## Sample Midterm Questions

### Question 1

- a) Briefly explain the difference between a **hard** real-time system and a **soft** real-time system.
- b) Give an example of a **hard** real-time system. Why is it a **hard** real-time system?
- c) Give an example of a **soft** real-time system. Why is it a **soft** real-time system?

### Question 2

- a) List the five **protocol stack** conceptual **layers** from highest to lowest, with the lowest being the layer most closely related to the physical network.
- b) Give the names of the **messages** passed between the layers.

### Question 3

- a) Briefly explain the difference between **thread safe**, **conditionally thread safe**, and **thread compatible** classes. Give the name and a brief description of a Java class in each of these three categories. (You do not need to write out any code for the classes you choose.)
- b) Briefly explain the difference between Java methods `notify()` and `notifyAll()`.

### Question 4

- a) What does TFTP stand for?
- b) Which of the five protocol layers discussed in class does TFTP implement?
- c) What is the **full** name of the protocol that TFTP sits on top of (i.e. the layer below it)? Which of the five protocol layers discussed in class does your answer to part c) implement?

### Question 5

Given the *startWrite* method below, taken from a solution to the **first readers-writers problem** tackled in an assignment, answer the following questions:

- a) Briefly describe the meaning of *startWrite*'s *synchronized* modifier.
- b) What is the purpose of the *wait()* method invocation?
- c) What method(s) (invoked by another thread) will cause the *wait()* method to return?
- d) What would be the consequences of omitting the *dbWriting = true;* statement?
- e) Identify the error in this code. Explain why it is an error and how the error should be corrected.

```
public synchronized void startWrite()
{
    System.out.println(Thread.currentThread().getName() +
                        " beginning startWrite()");
    if ((dbReading) || (dbWriting)) {
        try {
            wait();
        } catch (InterruptedException e) {}
    }
    dbWriting = true;
    System.out.println(Thread.currentThread().getName() +
                        " is writing");
}
```

## Question 6

For each of parts (a) through (e), one or more of the statements is correct. Place an "x" or a check mark in the [ ] beside each correct statement.

(a) A newly created Java thread starts executing:

- ☐ immediately after the `Thread()` constructor returns.
- ☐ sometime after the thread's `start()` method is invoked, when the Java Virtual Machine (JVM) allocates the processor to the thread.
- ☐ only when the thread's `run()` method is explicitly invoked, and the JVM has allocated the processor to the thread.
- ☐ only after the thread that created the new thread terminates.

(b) The program's initial thread of control, which executes `main()`, will not terminate until all the threads it created terminate.

- ☐ True.
- ☐ False.

(c) The `put()` and `get()` methods in class `Box` are declared with the `synchronized` modifier. This means that:

- ☐ every `Box` object will be an active object
- ☐ the JVM will ensure that only one thread at a time will execute `get()` or `put()`, even if multiple threads concurrently invoke these methods
- ☐ if the JVM that is executing the program supports timeslicing, execution of the threads that invoke these methods will be synchronized with the platform's timer or clock facilities.
- ☐ a thread can execute a `Box` object's methods only when it holds the object's lock.
- ☐ a thread that invokes either of these methods relinquishes the processor and moves into the ready to run state.

(d) Using the levels of thread safety described in *Effective Java*, class `Box` should be categorized as:

- ☐ immutable.
- ☐ thread-safe.
- ☐ conditionally thread-safe.
- ☐ thread-compatible.
- ☐ thread-hostile.

(e) The wait-loop idiom in `get()`:

- ☐ is the recommended approach to enforcing mutual exclusion between the producer and consumer threads on the object stored in a `Box`.
- ☐ is the recommended approach to providing condition synchronization between the producer and consumer threads.
- ☐ causes the consumer thread to busy-wait by repeatedly invoking `wait()`, until an object is placed in the `Box`.

### Question 7

- Give the name and a brief description of a real world real-time computer system. Ensure that you clearly explain the purpose / function of the system that you have chosen.
- We have looked at several definitions of real-time systems, each with a list of characteristics that apply. Here is one of them:

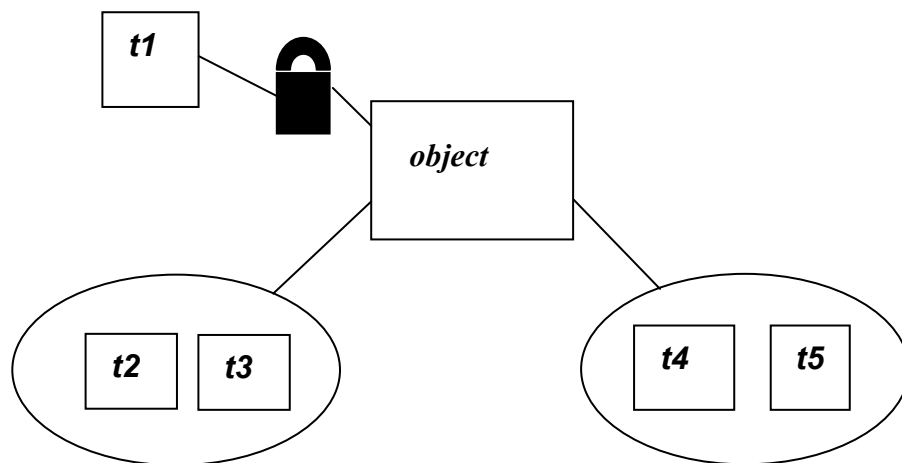
A computer-based system that must resolve a number of difficult issues simultaneously:

- rapid response
- continuous operation
- bursty stimuli
- failures of its components and connections
- uncertainty about communications and processing delays
- uncertainty about the state of the environment
- geographical distribution
- need to adapt over time while continuously operating

Pick three (3) of the characteristics listed above and briefly describe how your real-time system demonstrates or resolves each of these issues.

- Is your system a hard real-time system or a soft real-time system? Ensure that your answer demonstrates your understanding of the difference between hard and soft real-time systems.

### Question 8



- Draw the new diagram that results if `t1` executes `notify()` and explain your diagram.
- Now draw a new diagram showing the state after `t1` has finished and the JVM has granted the lock to another process. Again, explain your diagram.

## Question 9

Here is the code for two Java classes. Each of these classes is to be used to create new Java threads.

```
class Example1 extends Thread
{
    BoundedBuffer buffer

    public Example1 (BoundedBuffer buf)
    {
        buffer = buf;
    }
}

class Example2 implements Runnable
{
    BoundedBuffer buffer

    public Example2 (BoundedBuffer buf)
    {
        buffer = buf;
    }
}
```

- Briefly explain the two different methods of thread creation used here. Ensure that you explain the relationship between `Example1` and class `Thread`, and between `Example2` and class `Thread`.
- In each case, will the code given compile? If so, what would an instance of this class do? If not, why not?

## Question 10

In class we discussed how a client process and a server process establish a TFTP "Write Request" (WRQ) connection between the two processes to initiate the transfer of a file from the client to the server. In a server with a multithreaded architecture, one thread (listener thread) waits on port 69 for UDP datagrams containing TFTP request packets. After a valid request is received, the server creates another thread (the *client connection thread*) that is responsible for communicating with the client to transfer the file.

Draw a **Timing Diagram** showing the transfer of file "sample.txt" containing 1024 bytes from the client to the server. The diagram should start with the initial "Read Request" by the client, and continue until the entire file has been transferred, showing all the packets sent during this transfer (from client to server and vice versa). Your diagram should have **3** (three) vertical timelines: one for the client, one for the server thread that waits for request packets (i.e. listener thread), and one for the client connection thread. Ensure that you show when threads are created/destroyed, if this happens during the time shown on your diagram. Assume that no errors occur during the file transfer, and that no packets are lost, delayed, or duplicated.

**For each TFTP packet, you must clearly show the TFTP packet type, sending TID, receiving TID, block number (if applicable), and number of bytes of data (if applicable).**

### Question 11

In this question you are going to modify the Box class discussed in the lectures. The Box class is to become a BigBox class, in which we can store many objects, not just one. Instead of modeling our contents with a single Object as in Box, the contents of our BigBox are modeled by an ArrayList of Objects. Items are removed from the box in the order in which they were added (i.e. FIFO). **Do not** use any of Java's synchronized collection wrapper classes in this question, just an ArrayList. (You can look at java.sun.com's ArrayList API if you need help with the methods – students were given a summary of methods that is missing here.)

a) Complete the BigBox class here. Ensure that your class will work properly in the presence of multiple producer and consumer threads. (16 marks)

```
public class BigBox
{
    // The contents of the box.
    private _____ contents = _____;

    // Add other fields (member variables) here as required.

    // put: Puts an object (the Object argument obj) into the
    // big box. Returns (void) when the object argument has
    // been added.

    // get: Removes and returns the object that has been in the
    // big box the longest. Returns when there's an object in
    // the big box that can be removed. This method has no
    // arguments.

    // replace: Replaces the object that has been in the big box
    // the longest with the Object argument obj. Returns (void)
    // when there is an object to replace.

    // isEmpty: This method returns a boolean value indicating
    // whether or not the big box is currently empty (true if it
    // is false otherwise). This method has no arguments.

    // look: Returns the object that has been in the big box the
    // longest without removing it. Returns when there is an
    // object to look at. This method has no arguments.
}
```

- b) Which of the five levels of thread safety discussed in class does your class BigBox (above) fall into? Briefly explain your answer, demonstrating that you understand the definitions of the different levels of thread safety.
- c) Draw a UML Class Diagram for your class BigBox (above). For each attribute, give full details including the visibility, type and initial value, if applicable. For each of the operations, give full details including the signature, visibility, and synchronization property (even if it is the default).