Final Exam

CMSC 433: Programming Language Technologies and Paradigms

December 14, 2014

Name ____

Instructions

- This exam has 15 pages (including this one); make sure you have them all.
- You have 120 minutes to complete the exam
- The exam is worth 110 points. Allocate your time wisely: some hard questions are worth only a few points, and some easy questions are worth a lot of points.
- If you have a question, please raise your hand and wait for the instructor.
- **Write neatly** and **clearly indicate your answers**.

	Question	Points	Score
	Definitions	20	
	Distributed Computing	20	
	Java Concurrency	30	
	Erlang Execution	20	
	Erlang Coding	10	
	Parallelization	10	
	Total	110	

Enjoy your break!

1. Definitions (20 points total, 2 points each).

   Next to each term on the left, write the letter on the right that corresponds to the definition of the term, or is a true statement about it.

	(a) A technique for ensuring the atomicity of multiple method calls
____ Marshaling	(b) Performing garbage collection using multiple parallel threads
	(c) Performing garbage collection in parallel with the mutator
____ Parallel GC	(d) A property of the Java Memory Model (JMM)
____ Sequential consistency	(e) A class that satisfies its specification when used by multiple threads
	(f) An example of which is calling `BlockingQueue.take`
____ Thread-safe class	(g) A lock that can be acquired by multiple threads
____ Client-side locking	(h) A property of executions in which the effects of separate threads' operations can be viewed as happening in a total order
	(i) Erlang uses this style of typing
____ State-dependent action	(j) A lock that is particularly important when synchronized methods call other synchronized methods
	(k) Predicts a task's possible speedup based on the number of processors and the amount of strictly-serial work
____ Cast (as in Erlang)	(l) An action involving an object implementing a finite state machine
____ Dynamic typing	(m) A message sent to a server that expects a response
____ Reentrant lock	(n) Predicts how many threads you should create based on the number of processors and the ratio of wait time to compute time
	(o) A message sent to a server that needs no response
____ Amdahl's Law	(p) A class with no data races
	(q) Java uses this style of typing
	(r) The process of preparing a remote method invocation

2. (Distributed computing, 20 points)

   (a) (Scala Actors, 4 points) You can send a message to a Scala actor a using the code a ! msg. Scala also allows you to send messages via the syntax a !! msg and the syntax a !? msg. How will these two invocations differ from the first?

   (b) (Java RMI, 4 points) Consider the following two possible definitions of a class MyString:

   Implementation (a):

```
public class MyString {
  private final String contents;
  public String getContents() { return contents; }
}
```

   Implementation (b):

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface RemoteString extends Remote {
  public String getContents() throws RemoteException;
}
public class MyString implements RemoteString {
  private final String contents;
  public String getContents() throws RemoteException { return contents; }
}
```

   Suppose x is a reference to a *remote* object of type Foo, which has a method f that takes a MyString argument, and y is a reference to a local MyString object. If we invoke x.f(y) what will be the difference in the case that we use implementation (a) vs. implementation (b) ?

(c) (MapReduce, 12 points) The following Hadoop code is based on the project 5 tutorial:

```java
public static class TokenizerMapper extends
    Mapper<Object, Text, IntWritable, IntWritable> {
  private final static IntWritable one = new IntWritable(1);
  private IntWritable count = new IntWritable(0);

  public void map(Object file, Text value, Context context)
      throws IOException, InterruptedException {
    StringTokenizer itr = new StringTokenizer(value.toString());
    while (itr.hasMoreTokens()) {
      int length = itr.nextToken().length();
      count.set(length);
      context.write(count, one);
} } }

public static class IntSumReducer extends
    Reducer<IntWritable, IntWritable, IntWritable, IntWritable> {
  private IntWritable result = new IntWritable();

  public void reduce(IntWritable key, Iterable<IntWritable> values,
      Context context) throws IOException, InterruptedException {
    int sum = 0;
    for (IntWritable val : values) {
      int v = val.get();
      sum += v;
    }
    result.set(sum);
    context.write(key, result);
} }
```

i. (6 points) What will the keys and values be in the final output file, assuming this code is run on a directory of text files (which consist of, say, the collected works of Shakespeare)?

ii. (6 points) Could IntSumReducer also be used as a combiner? Why or why not?

3. (Java concurrency, 30 points) Each of the next few pages contains a small program with at least two classes, one of which is called TestCase. Consider what would happen when running TestCase.main().

If the program would terminate normally and always print the same answer, indicate that answer. Otherwise, indicate *one or more* of the following things the program will do: (a) exhibit a data race, (b) exhibit an atomicity violation, (c) exhibit a deadlock, (d) run forever, (e) print different things on different runs.

Each problem is worth 6 points.

Be careful: I am only interested in what will happen for executions of the given TestCase.main, not hypothetical ways in which the classes could be used.

Also, note that in several cases we are ignoring that methods could throw InterruptedException, to keep the code shorter.

(a)

```
public class GlobalRef {
  public int x = 0;
  public Object y = new Object();
  public void increment() { synchronized(y) { x = x + 1; } }
}

public class TestCase {
  public static void main(String args[]) {
    final GlobalRef r = new GlobalRef();
    new Thread() { public void run() { r.increment(); } }.start();
    new Thread() { public void run() { r.increment(); } }.start();
    Thread.yield();
    System.out.println(r.x);
  }
}
```

(b)

```
public class OnePlaceBuffer {
  private String value = null;
  public synchronized String get() {
    while (value == null) wait(); // ignore IE
    String ret = value;
    value = null;
    notify();
    return ret;
  }
  public synchronized void put(String x) {
    while (value != null) wait(); // ignore IE
    value = x;
    notify();
  }
}

public class TestCase {
  public static void main(String args[]) {
    final OnePlaceBuffer b = new OnePlaceBuffer();
    Runnable p = new Runnable() {
      public void run() { b.put(”Hello”); }
    };
    new Thread(p).start();
    new Thread(p).start();
    System.out.println(b.get());
  }
}
```

(c)

```
public class SillyFact {
  private ExecutorService exec;
  public SillyFact(ExecutorService e) { exec = e; }
  public int nth(int n) {
    if (n == 0) return 1;
    else {
      final int m = n-1;
      final SillyFact me = this;
      Callable<Integer> c = new Callable<Integer>() {
        public Integer call() { return me.nth(m); }
      };
      Future<Integer> f = exec.submit(c);
      return n + f.get(); // ignore IE
    }
  }
}

public class TestCase {
  public static void main(String args[]) {
    ExecutorService e = Executors.newFixedThreadPool(4);
    SillyFact f = new SillyFact(e);
    System.out.println(f.nth(6));
    e.shutdown();
  }
}
```

(d)

```
public class Pair {
  public final int left , right ;
  public Pair(int x, int y) { left = x; right = y; }
}
public class ConcPair {
  private Pair p = new Pair(0,0);
  public void updateLeft(int x) {
    int y = p.right ;
    p = new Pair(x,y);
  }
  public void updateRight(int x) {
    int y = p.left ;
    p = new Pair(y,x);
  }
  public int getLeft () { return p.left ; }
  public int getRight() { return p.right ; }
}

public class TestCase {
  public static void main(String args[]) throws InterruptedException {
    final ConcPair m = new ConcPair ();
    Thread t1 = new Thread() { public void run() { m.updateLeft(1); } };
    Thread t2 = new Thread() { public void run() { m.updateRight(1); } };
    t1.start (); t2.start ();
    t1.join (); t2.join ();
    System.out.println ("("+m.getLeft()+","+m.getRight()+")");
  }
}
```

(e) Note: Here, OptWhiteList is a different implementation of ConcPair from the previous question; the Pair class is the same as before; the TestCase class is the same, but refers to OptWhiteList

```java
public class OptWhiteList {
  private AtomicReference<Pair> p;
  public OptWhiteList() { p = new AtomicReference<Pair>(new Pair(0,0)); }
  private void update(boolean doLeft, int x) {
    while (true) {
      Pair q, old = p.get();
      if (doLeft) q = new Pair(x,old.right);
      else        q = new Pair(old.left,x);
      if (p.compareAndSet(old,q)) break;
    }
  }
  public void updateLeft(int x) { update(true,x); }
  public void updateRight(int x) { update(false,x); }
  public int getLeft() { return p.get().left; }
  public int getRight() { return p.get().right; }
}

public class TestCase {
  public static void main(String args[]) throws InterruptedException {
    final OptWhiteList m = new OptWhiteList();
    Thread t1 = new Thread() { public void run() { m.updateLeft(1); } };
    Thread t2 = new Thread() { public void run() { m.updateRight(1); } };
    t1.start(); t2.start();
    t1.join(); t2.join();
    System.out.println("("+m.getLeft()+","+m.getRight()+")");
  }
}
```

4. (Erlang Execution, 20 points)

Look at each of the Erlang programs below. Along with each program, we will provide a call; say what the call will do. If it returns, indicate the value returned; if it fails with some exception, indicate the exception; or if it does not return, say so. Explain your reasoning if you hope for partial credit. You may assume all of the programs compile. Each question is worth 5 points.

(a) What will `go([1,2,3])` do?

```
go([]) -> 0;
go([H|T]) -> 1+go(T).
```

(b) What will `go( grill ,burger)` do?

```
machine_map() ->
    dict : from_list ([{ grill , {burger, 500}},
                       { frier , { fries , 250}},
                       {soda_fountain, {coke, 100}}]).
go(Machine,Food) ->
  Dict = machine_map(),
  {Food,Time} = dict:fetch(Machine,Dict),
  Time.
```

(c) What will go(goo) do?

```
init () -> register(s1, spawn(fun () -> receive {msg,Pid} -> Pid ! ok end end)).
go(Msg) ->
      init (),
      s1 ! {Msg,self()},
      receive
            ok -> ok
      end.
```

(d) What will go() do?

```
pmap(F,L) ->
   Me = self(),
   Pids = lists :map(fun (M) ->
                spawn(fun () ->
                   Pid = self (),
                   R = F(M),
                   Me ! {Pid,R} end) end,
                  L),
    lists :map(fun (Pid) ->
                   receive
                      {Pid,R} -> R
                   end
                 end, Pids).

go() ->
      pmap(fun ([H|T]) -> H end, [[1,2,3],[3,2,1],[3,2],[1,7,5,4]] ).
```

5. (Erlang coding, 10 points) The following function go() calls a function f for which we have not provided a definition. This function will return a process ID to which go subsequently sends messages. Give an implementation of f so that go() returns correctly with the atom **ok**. (You may define other functions too, if you need them.)

```
go() ->
  P = f (),
  P ! [1, 3, 7],
  receive
    {P,3} -> ok
  end,
  P ! [4, 5],
  receive
    {P,2} -> ok
  end,
  P ! done,
  receive
    finished -> ok
  end,
  ok.
```

Give your definition of f here:

6. (Java parallelization, 10 points) A Data object has a Grid as a private field. Data also defines a method heatMap to compute this grid's "heat map", which is itself a Grid. Assume that ArrayGrid is an implementation of the Grid interface backed by an array.

   On the next page, provide the implementation of a new Data class method parHeatMap that produces the same result as heatMap, but does it in parallel.

   Extra credit: Assume that the time to create and run a new task is roughly the same as the time to compute the heat map of 100 grid squares. For extra credit, incorporate this information in your solution.

```java
public interface Grid {
  public int getWidth(); // x axis length
  public int getHeight(); // y axis length
  public int get(int x, int y); // (0,0) is in the upper left
  public void set(int x, int y, int v); // sets grid(x,y) to v
}
public class Data {
  final private Grid grid;
  public Data(Grid grid) {
    this.grid = grid;
  }
  private int heat(int x, int y) {
    int v = grid.get(x,y);
    for (int i = x-1; i<=x+1; i++) {
      for (int j = y-1; j<=y+1; j++) {
        if (i >= 0 && i < grid.getWidth() &&
            j >= 0 && j < grid.getHeight() &&
            !(i == x && j == y)) {
          v = v + (grid.get(i,j) / 2);
        }
      }
    }
    return v;
  }
  public Grid heatMap() {
    Grid results = new ArrayGrid(grid.getWidth(),grid.getHeight());
    for (int i = 0; i<grid.getWidth(); i++) {
      for (int j = 0; j<grid.getHeight(); j++) {
        results.set(i,j,heat(i,j));
      }
    }
    return results;
} }
```

You may find the following API calls useful:

Runtime.getRuntime().availableProcessors()	returns the number of available processors
e.awaitTermination(Long.MAX_VALUE, TimeUnit.SECONDS)	when e is an ExecutorService, this calls waits until all its tasks terminate.

Also see question 3(c) for some useful API calls.

(This page intentionally left blank; feel free to use as scratch paper.)