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.

<table>
<thead>
<tr>
<th>Question</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definitions</td>
<td>20</td>
</tr>
<tr>
<td>Distributed Computing</td>
<td>20</td>
</tr>
<tr>
<td>Java Concurrency</td>
<td>30</td>
</tr>
<tr>
<td>Erlang Execution</td>
<td>20</td>
</tr>
<tr>
<td>Erlang Coding</td>
<td>10</td>
</tr>
<tr>
<td>Parallelization</td>
<td>10</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>110</strong></td>
</tr>
</tbody>
</table>

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.

--- Marshaling
(a) A technique for ensuring the atomicity of multiple method calls

--- Parallel GC
(b) Performing garbage collection using multiple parallel threads
(c) Performing garbage collection in parallel with the mutator
(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
(h) A property of executions in which the effects of separate threads’ operations can be viewed as happening in a total order

--- Client-side locking
(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
(n) Predicts how many threads you should create based on the number of processors and the ratio of wait time to compute time

--- Reentrant lock
(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 \texttt{a} using the code \texttt{a ! msg}.
Scala also allows you to send messages via the syntax \texttt{a !! msg} and the syntax \texttt{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 \texttt{MyString}:

```
Implementation (a):
1  public class MyString {
2      private final String contents;
3      public String getContents() { return contents; }
4  }

Implementation (b):
1  import java.rmi.Remote;
2  import java.rmi.RemoteException;
3  public interface RemoteString extends Remote {
4      public String getContents() throws RemoteException;
5  }
6  public class MyString implements RemoteString {
7      private final String contents;
8      public String getContents() throws RemoteException { return contents; }
9  }
```

Suppose \texttt{x} is a reference to a \textit{remote} object of type \texttt{Foo}, which has a method \texttt{f} that takes a \texttt{MyString} argument, and \texttt{y} is a reference to a local \texttt{MyString} object. If we invoke \texttt{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)

```java
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);
    }
}
```
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());
    }
}
```java
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();
    }
}
```

This program has a thread starvation deadlock because the Executor we are using has a fixed number of threads (8), and yet the recursive call to `nth` will eventually exceed this number, and thus each thread will be stuck waiting, and never complete.
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; }
}
```

```java
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 \textit{go}([1,2,3]) do?

\begin{verbatim}
go([],) -> 0;
go([H|T]) -> 1+go(T).
\end{verbatim}

(b) What will \textit{go( grill , burger)} do?

\begin{verbatim}
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.
\end{verbatim}
(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.)

```erlang
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.

```
1 public interface Grid {
2   public int getWidth(); // x axis length
3   public int getHeight(); // y axis length
4   public int get(int x, int y); // (0,0) is in the upper left
5   public void set(int x, int y, int v); // sets grid(x,y) to v
6 }
7
class Data {
8   private Grid grid;
9   public Data(Grid grid) {
10      this.grid = grid;
11   }
12
13   private int heat(int x, int y) {
14      int v = grid.get(x,y);
15      for (int i = x-1; i<=x+1; i++) {
16         for (int j = y-1; j<=y+1; j++) {
17            if (i >= 0 && i < grid.getWidth() &&
18               j >= 0 && j < grid.getHeight() &&
19               !(i == x && j == y)) {
20               v = v + (grid.get(i,j) / 2);
21            }
22         }
23      } return v;
24   }
25
26   public Grid heatMap() {
27      Grid results = new ArrayGrid(grid.getWidth(),grid.getHeight());
28      for (int i = 0; i<grid.getWidth(); i++) {
29         for (int j = 0; j<grid.getHeight(); j++) {
30            results.set(i,j,heat(i,j));
31         }
32      } return results ;
33   }
```
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 answer would have netted around 8 points: it is highly inefficient to spawn one thread per square.

```java
public Grid heatMap()
{
    int nproc = Runtime.getRuntime().availableProcessors();
    ExecutorService exec = Executors.newFixedThreadPool(nproc+1);
    final Grid results = new ArrayGrid(grid.getWidth(), grid.getHeight);
    for (int i = 0; i < grid.getWidth(); i++)
    {
        for (int j = 0; j < grid.getHeight(); j++)
        {
            final int a = i;
            final int b = j;
            exec.submit(new Runnable()
            {
                public void run()
                {
                    results.set(a, b, heat(a, b));
                }
            });
        }
    }
    exec.shutdown();
    exec.awaitTermination(Long.MAX_VALUE, TimeUnit.SECONDS);
    return results;
}
```

To get full credit, you would divide up the grid into roughly equal portions according to the number of processors available, while avoiding corner cases of tasks becoming too small.