13

University of Cape Town

Department of Computer Science

Test

Enter the following details AND shade in the corresponding blocks to the right with your Student Number.

Student Number : NTSLUV003

Name (optional) : LUVO

Course : CSC 2002 S

Date : 22 -August-2016

Instructions:

a) Answer all questions.
b) Write your answers in PEN in the spaces provided.
c) Show all calculations where applicable.

A	□	□	□	□	□	□	■	■	□	0
B	□	□	□	□	□	□	□	□	□	1
C	□	□	□	□	□	□	□	□	□	2
D	□	□	□	□	□	□	□	□	■	3
E	□	□	□	□	□	□	□	□	□	4
F	□	□	□	□	□	□	□	□	□	5
G	□	□	□	□	□	□	□	□	□	6
H	□	□	□	□	□	□	□	□	□	7
I	□	□	□	□	□	□	□	□	□	8
J	□	□	□	□	□	□	□	□	□	9
K	□	□	□	□	□	□				
L	□	□	□	■	□	□				
M	□	□	□	□	□	□				
N	■	□	□	□	□	□				
O	□	□	□	□	□	□				
P	□	□	□	□	□	□				
Q	□	□	□	□	□	□				
R	□	□	□	□	□	□				
S	□	□	■	□	□	□				
T	□	■	□	□	□	□				
U	□	□	□	□	■	□				
V	□	□	□	□	□	■				
W	□	□	□	□	□	□				
X	□	□	□	□	□	□				
Y	□	□	□	□	□	□				
Z	□	□	□	□	□	□				

FOR OFFICIAL USE ONLY:

Question		1		2		3		4		5		6		7		8	
Max																	
Marks	0	■	□	■	□	■	□	□	□	□	□	□	□	□	□	□	□
	1	□	□	□	□	□	■	□	□	□	□	□	□	□	□	□	□
	2	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
	3	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
	4	□	■	□	□	□	□	□	□	□	□	□	□	□	□	□	□
	5	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
	6	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
	7	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
	8	□	□	□	■	□	□	□	□	□	□	□	□	□	□	□	□
	9	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□	□
Marker		MM															

University of Cape Town

Computer Science CSC2002S

Class Test 1, 22nd August 2016

Marks: 35 **Time:** 45 minutes
Instructions: *Answer all questions*

Question 1 – Multiple Choice [6 marks]
Circle the corresponding letter of the correct answer.

(1) A divide-and-conquer parallel algorithm for calculating the sum of n numbers has a computational **span** of:
- [✓ A. O(log n)]
- B. O(n)
- C. O(n/logn)
- D. O(nlogn) ✓
- E. ∞

(2) If you have a parallel computer with 500 processors, what is the **minimum fraction** of your program that must be parallel in order to obtain a speedup of 500?
- A. 0.01
- B. 0.5
- C. 0.9 ✓
- [✓ D. 1]
- E. A speedup of 500 is not possible in this case.

(3) If I obtain ideal speedup for a parallelized program on P processes, this means that:
- A. $T_P < T_\infty$
- [✓ B. $T_p = T_1/P$]
- C. $T_p < T_1/P$ ✓
- D. $T_p = 1 - T_1$
- E. All of the above

(4) In Java, an example of a synchronizer is:
- [✓ A. Thread.join]
- B. a latch
- C. a barrier ✓
- D. A blocking queue
- E. All of the above

(5) In Java, you can solve the *Dinining Philosophers* concurrency problem using:
- A. Using the built-in locks in a synchronized block
- B. Calling wait() and notify() on a condition variable ?
- C. Declaring a variable volatile
- D. Calling join() on a thread.
- E. None of the above

(6) In Java, the method Thread.stop is deprecated because:
- A. It is inherently unsafe
- B. It is prone to deadlock
- [✓ C. The Java fork-join framework has superseded traditional threads] ✗
- D. The Producer-Consumer problem
- E. A and B

Question 2: Parallelism [15 Marks]

Examine the following Java code (assume all necessary imports):

```
public class Para extends RecursiveTask<Integer>  {
    int lo,hi,x;
    int[] arr;
    static int CUTOFF=5;

    Para(int[] a, int val, int l, int h) {arr=a; x=val; hi=h;lo=l;}

    protected Integer compute() {
        if((hi-lo) < CUTOFF) {
            int count=0;
            for(int i=lo; i < hi; i++)
                if (arr[i]== x) count++;
            return count;
        } else {
            Para left  = new Para(arr,x,lo,(hi+lo)/2);
            Para right = new Para(arr,x,(hi+lo)/2,hi);
            left.fork(); // #1
            right.fork();// #2
            int b= right.join();// #3
            int a= left.join(); // #4
            return a+b;
        }
    }
   }
```

a) Does this class implement a map, or a reduction, or neither? Justify your answer. [2]

Reduction — because it doesnt return the output size same as the input size it returns a single number as sequential

b) Explain clearly what this class computes. [2]

It computes ~~the~~ if the the value entered equal to some number in the array then count Increment or else it cuts the array into half and have left side of the Array and the right side of the Array then it Joined them counting how many number the Array has.

c) The Drive class listed below was written to test the Para class above. Write down the exact output when this code is run (assume all necessary imports). [2]

```
public class Driver {
    static final ForkJoinPool fjPool = new ForkJoinPool();
    static int  work(int[] arr, int key){
           return fjPool.invoke(new Para(arr,key,0,arr.length));
    }
    public static void main(String[] args) {
            int max =1000000;
            int [] arr = new int[max];
            Random random = new Random();
            for (int i=0;i<max;i++) { arr[i]=random.nextInt(100); }

            int key=500;
            int output= work(arr,key);
            System.out.println(key+": "+output);
        }
}
```

1

500 : 0

d) Would it be a good idea to replace these lines –

```
left.fork(); // #1
right.fork();// #2
int b= right.join();// #3
int a= left.join(); // #4
```

- with the lines below?

```
left.fork();
int b=right.compute();
int a=left.join();
return a+b;
```

Justify and explain your answer. [4]

Yes, because the fork method start the threads and the Join() method waits and finishes the execution ~~the first lines~~ all of this is occuring in the compute method so calling the compute method same as calling fork() and Join() which is what is happening in the lines so the 2 stacks of lines are basically the same

1

e) Complete the compute() method for the RecursiveAction class listed below. This class performs matrix addition on the one-dimensional arrays *a* and *b:* the compute function must calculate all the values for the one-dimensional matrix (or array) *c*, where *c[i]=a[i] +b[i]*. You can assume that the matrices a, b and c are all of equal length.

```
public class ParaSupp extends RecursiveAction  {
    int lo,hi;
    int[] a; int [] b; int[] output;
    static int CUTOFF=500;

    ParaSupp(int[] A, int [] B, int [] out, int l, int h)
        {a=A; b=B; output=out;  hi=h;lo=l;}

    protected void compute() {  //complete this method
    }
}
```

[5]

```
if ((hi - lo) < (CUTOFF) {  → int[ ] c = new int[hi];
        for (int i = lo ; i < hi ; i++) {
            ~~int [i] c = new int [ ]~~
            ~~int [ ] c =~~
            ~~int [ ] c = new int [hi]~~
                c[i] = a[i] + b[i]
            ~~c[i] . f~~ }
            c.fork();
            c.join();

    }
```

$\boxed{2}$ ✓ ? ✗

Question 3 : Concurrency [14 Marks]

```
public class Histogram {
    private AtomicInteger [] buckets;
    private int Size;

    Histogram(int s) {
        buckets = new AtomicInteger[s];
        Size=s;
        for (int i=0;i<Size;i++) {buckets[i]= new AtomicInteger(0);}
    }

    synchronized public int getSize() {return Size;}

    public int getCount(int b) {return buckets[b].get();}

    public void incrCount(int b) {
        buckets[b].getAndIncrement(); }

    public synchronized void addHist(Histogram otherH) {
            if(Size==otherH.getSize()) {
                for (int i=0;i<Size;i++)
                  buckets[i].set(buckets[i].get()+otherH.getCount(i));
            }
    }
}
```

a) Does the code above follow the *Java monitor pattern*? Justify your answer. [2]

yes, It is Java That use the atomicity Abtract ~~c~~lasses.

0

b) In Java, AtomicIntegers guarantee visibility. Explain the concept of visibility in the context of concurrency. [2]

As concurrency is ~~the et~~ computes the efficiency ~~and~~ of execution of simulation then ~~visit~~ the meaning of visibility is that ~~the~~ it is known which Item of thread that is executed at a specific time

0

c) In the code above, why is it **not necessary** to synchronize the methods getCount() and incrCount()? [2]

~~becau~~ because they already exist is the ~~H~~ Atomic Integer ~~metho~~ class and also there cant be race Problems in those ~~fo~~ methods

0

d) Why is it not **desirable** to synchronize the methods getCount() and incrCount() (i.e. why would it be a bad idea to do this)? [2]

e) Explain why it is necessary to synchronize the addHist() method. [2]

it is necessary to encounter race problems like race condition

1

f) Explain clearly the code adjustments necessary to ensure that the Histogram class is deadlock free when shared between threads. [4]

ADDITIONAL SPACE – PLEASE USE THIS SPACE TO COMPLETE YOUR ANSWERS SHOULD YOU NEED ADDITIONAL SPACE. WRITE THE CORRESPONDING QUESTION NUMBER NEXT TO EACH ANSWER.