Homework - 1

Amit Lawanghare

Question 1 (20 points)
We have discussed in the class the implementation of the readers-writers problem in Java. However, the read and write tasks of the **reader** thread and the **writer** thread are not given. Implement these tasks in Java as reading and writing of a file named *counter.txt*, which contains an integer counter.

A **reader** thread

- reads the counter from the file, and
- prints out its thread name and the value of the counter.

A **writer** thread

- increments the value of the counter in the file,
- prints out its thread name and the new value of the counter.

Each thread repeats its task indefinitely in a random amount of time between 0 and 3000 ms.

Your **main** program should create 20 **reader** threads and 3 **writer** threads.

Besides the source code, turn in scripts showing that you compile and run the program successfully.

Turn in also some sample outputs.

Source code –

ReadWriteContainer.java

```java
package hw660;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Scanner;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 * Class to hold lock and condition
 *
 * @author Amit
 *
 */
public class ReadWriteContainer {
	private final String file = "counter.txt";
	private final Lock _mutex = new ReentrantLock();// create mutex instance
	private final Condition readerQueue = _mutex.newCondition();// Returns a new
									// Condition
									// instance for
									// reader that
									// is bound to
									// this Lock
									// instance.
	private final Condition writerQueue = _mutex.newCondition();// Returns a new
									// Condition
									// instance for
									// writer that
									// is bound to
									// this Lock
									// instance.
	private int readers_count = 0;// to store current readers count
	private int writers_count = 0;// to store current writers count

	public ReadWriteContainer() {
		try {
			FileWriter filewriter = new FileWriter(new File(file));
			filewriter.write(new Integer(0).toString());
			filewriter.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
	void readerrun() throws InterruptedException {
		_mutex.lock(); // acquire lock
		while (!(writers_count == 0)) {
			readerQueue.await();
		}
		readers_count++;
		_mutex.unlock();
		read(file);//read file
		_mutex.lock();
		if (--readers_count == 0) {
			writerQueue.signal();//signal writers
		}
		_mutex.unlock();//remove lock
	}

	void writerrun() throws InterruptedException {
		_mutex.lock();
		while (!((readers_count == 0) && (writers_count == 0))) {
			writerQueue.await();// when reader and writer is zero wait in
										// writerQueue
		}
		writers_count++; // increment writer
		_mutex.unlock();// remove lock
		write(file);// write to file
		_mutex.lock(); // acquire lock
		writers_count--; // only one writer at a time
		writerQueue.signal(); // signal writers
		readerQueue.signalAll(); // signal all readers
		_mutex.unlock();// remove lock
	}

	void read(String path) {
		try {
			Scanner reader = new Scanner(new FileInputStream(path));
			int x = reader.nextInt();
			System.out.printf(Thread.currentThread().getName() + " is reading ...");
			System.out.printf(" Counter value : %d\n", x);
			reader.close();
		} catch (IOException ex) {
			ex.printStackTrace();
		}

	}

	void write(String path) {
		int countw;
		try {
			Scanner reader = new Scanner(new FileInputStream(path));
			countw = (int) reader.nextInt();
			countw++;
			FileWriter f = new FileWriter(new File(path));
			f.write(new Integer(countw).toString());
			f.close();
			System.out.printf(Thread.currentThread().getName() + " Writing... ");
			System.out.printf(" Counter value : %d\n", countw);
			reader.close();
		} catch (IOException ex) {
			ex.printStackTrace();
		}
	}
}
```

BootStrapper.java

```java
package hw660;

import java.util.Random;

/**
 * This class is responsible for creating threads with maximum 20 readers and 3 writers
 * @author Amit
 *
 */
public class Bootstrapper {
	final static int READTCount = 20;// as given in homework
	final static int WRITETCount = 3;// as given in homework
	static ReadWriteContainer readerWriterContainer = new ReadWriteContainer();
	static Random random = new Random();

	/** static ReaderThread class extends thread class to
	 * continue in a loop with random number of wait time between 0 - 3000 milliseconds
	 * @author Amit
	 *
	 */

	static class readerThread extends Thread {
		@Override
		public void run() {
			System.out.println("Reader " + getName() + ": Started");
			while (true) {
				try {
					readerWriterContainer.readerrun();
					Thread.sleep(random.nextInt(3000));
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		}
	}

	/**
	 * static ReaderThread class extends thread class to
	 * continue in a loop with random number of wait time between 0 - 3000 milliseconds
	 * @author Amit
	 *
	 */
	static class writerThread extends Thread {
		@Override
		public void run() {
			System.out.println("Writer " + getName() + ": Started");
			while (true) {
				try {
					readerWriterContainer.writerrun();
					Thread.sleep(random.nextInt(3000));
				} catch (InterruptedException ex) {
					ex.printStackTrace();
				}
			}
		}
	}

	/**
	 * Main bootstrapper for
	 * @param args
	 * Create threads with max numbers 20 and 3 for reader and writer respectively
	 */
	public static void main(String[] args) {
		readerThread readerThreads[] = new readerThread[READTCount];
		writerThread writerThreads[] = new writerThread[WRITETCount];
		System.out.println("Program Start");
		for (int i = 0; i < WRITETCount; ++i) {
			writerThreads[i] = new writerThread();
			writerThreads[i].start();
		}
		for (int i = 0; i < READTCount; ++i) {
			readerThreads[i] = new readerThread();
			readerThreads[i].start();
		}
	}
}
```

Output –

```
Program Start
Writer Thread-0: Started
Writer Thread-1: Started
Writer Thread-2: Started
Reader Thread-3: Started
Reader Thread-4: Started
Reader Thread-5: Started
Reader Thread-7: Started
Reader Thread-8: Started
Reader Thread-10: Started
Reader Thread-12: Started
Reader Thread-13: Started
Reader Thread-14: Started
Reader Thread-16: Started
Reader Thread-18: Started
Reader Thread-19: Started
Reader Thread-20: Started
Reader Thread-22: Started
Reader Thread-9: Started
Reader Thread-6: Started
Reader Thread-11: Started
Reader Thread-15: Started
Reader Thread-17: Started
Reader Thread-21: Started
Thread-1 Writing...  Counter value : 1
Thread-0 Writing...  Counter value : 2
Thread-2 Writing...  Counter value : 3
Thread-10 is reading ... Counter value : 3
Thread-3 is reading ... Counter value : 3
Thread-14 is reading ... Counter value : 3
Thread-13 is reading ... Counter value : 3
Thread-22 is reading ... Counter value : 3
Thread-8 is reading ... Counter value : 3
Thread-20 is reading ...Thread-12 is reading ... Counter value : 3
Thread-19 is reading ... Counter value : 3
Thread-16 is reading ... Counter value : 3
Thread-5 is reading ... Counter value : 3
Thread-7 is reading ... Counter value : 3
Thread-6 is reading ... Counter value : 3
Thread-9 is reading ... Counter value : 3
 Counter value : 3
Thread-11 is reading ... Counter value : 3
Thread-17 is reading ... Counter value : 3
Thread-18 is reading ... Counter value : 3
Thread-21 is reading ... Counter value : 3
Thread-15 is reading ... Counter value : 3
Thread-4 is reading ... Counter value : 3
Thread-10 is reading ... Counter value : 3
Thread-22 is reading ... Counter value : 3
Thread-13 is reading ... Counter value : 3
Thread-20 is reading ... Counter value : 3
Thread-16 is reading ... Counter value : 3
Thread-21 is reading ... Counter value : 3
Thread-21 is reading ... Counter value : 3
Thread-17 is reading ... Counter value : 3
Thread-20 is reading ... Counter value : 3
Thread-1 Writing...  Counter value : 4
Thread-21 is reading ... Counter value : 4
Thread-8 is reading ... Counter value : 4
Thread-9 is reading ... Counter value : 4
Thread-18 is reading ... Counter value : 4
Thread-0 Writing...  Counter value : 5
Thread-8 is reading ... Counter value : 5
Thread-12 is reading ... Counter value : 5
```

```
Thread-2 Writing...  Counter value : 6
Thread-3 is reading ... Counter value : 6
Thread-14 is reading ... Counter value : 6
Thread-5 is reading ... Counter value : 6
Thread-11 is reading ... Counter value : 6
Thread-19 is reading ... Counter value : 6
Thread-12 is reading ... Counter value : 6
Thread-20 is reading ... Counter value : 6
Thread-7 is reading ... Counter value : 6
Thread-6 is reading ... Counter value : 6
Thread-1 Writing...  Counter value : 7
Thread-15 is reading ... Counter value : 7
Thread-16 is reading ... Counter value : 7
Thread-11 is reading ... Counter value : 7
Thread-4 is reading ... Counter value : 7
Thread-0 Writing...  Counter value : 8
Thread-11 is reading ... Counter value : 8
Thread-10 is reading ... Counter value : 8
Thread-20 is reading ... Counter value : 8
Thread-2 Writing...  Counter value : 9
Thread-5 is reading ... Counter value : 9
Thread-17 is reading ... Counter value : 9
Thread-8 is reading ... Counter value : 9
Thread-21 is reading ... Counter value : 9
Thread-6 is reading ... Counter value : 9
Thread-16 is reading ... Counter value : 9
Thread-22 is reading ... Counter value : 9
Thread-13 is reading ... Counter value : 9
Thread-18 is reading ... Counter value : 9
Thread-13 is reading ... Counter value : 9
Thread-9 is reading ... Counter value : 9
Thread-19 is reading ... Counter value : 9
Thread-0 Writing...  Counter value : 10
```

2. (20 points) We discussed in class the readers- writer problem with **writers' priority**, which can be solved in guarded commands:

void reader () **{** **when (writers == 0** **) [** **readers++;** **]** **//read** **[readers--;]** **}**	**void writer()** **{** **[writers++;]** **when ((readers == 0) &&** **(active_writers ==** **0))[** **active_writers++;** **]** **//write** **[writers--; active_writers--;]** **}**

Here *writers* represents the number of threads that are either writing or waiting to write. The variable *active_writers* represents the number of threads (0 or 1) that are currently writing.
Implement the solution using Java threads. Again simulate the tasks by reading from and writing to a file named *counter.txt* as in the previous problem.
Besides the source code, turn in scripts showing that you compile and run the program successfully. Turn in also some sample outputs.

Answer –

Source Code –
ReaderWriterThread.java
```
package ques2;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Random;
import java.util.Scanner;
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

/**
 * Read and writer implementation for thread
 * @author Amit
 *
 */
public class ReaderWriterThread {
	final Lock mutex = new ReentrantLock();
	final Condition readerQueue = mutex.newCondition();
	final Condition writerQueue = mutex.newCondition();
	int reader_count = 0;
	int writer_count = 0;
	int activeWriter_count = 0;
	String INP_FILE = "counter.txt";
	public static Random random = new Random();

	/**
	 * constructor
	 *
	 */
	public ReaderWriterThread() {
		FileWriter f;
		try {
			f = new FileWriter(new File(INP_FILE));
			f.write(new Integer(0).toString());
			f.close();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	/**
	 * Reader implementation
	 * @throws InterruptedException
	 */
	void reader() throws InterruptedException {
		mutex.lock();
		while (writer_count != 0) {
			readerQueue.await();
		}
		reader_count++;
		mutex.unlock();
		readFile(INP_FILE);
		mutex.lock();
		if (--reader_count == 0) {
			writerQueue.signal();
		}
		mutex.unlock();
	}

	/**
	 * writer implementation
	 * @throws InterruptedException
	 */
	void writer() throws InterruptedException {
		mutex.lock();
		writer_count++;
		while (!((reader_count == 0) && (activeWriter_count == 0))) {
			writerQueue.await();
		}
		activeWriter_count++;
		mutex.unlock();
		writeFile(INP_FILE);
		mutex.lock();
		activeWriter_count--;
		if (--writer_count == 0) {
			readerQueue.signalAll();
		} else {
			writerQueue.signal();
		}
		mutex.unlock();
	}

	/**
	 * increment read count
	 * @param path
	 */
	public void readFile(String path) {
		try {
			int inp = new Scanner(new FileInputStream(path)).nextInt();
			System.out.printf(Thread.currentThread().getName() + " Reader reading");
			System.out.printf(" Counter value: %d\n", inp);
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	/**
	 * increment writer count
	 * @param path
	 */
	public void writeFile(String path) {
		int writerCount;
		try {
			writerCount = (int) new Scanner(new FileInputStream(path)).nextInt();
			writerCount++;
			FileWriter writr = new FileWriter(new File(path));
			writr.write(new Integer(writerCount).toString());
			writr.close();
			System.out.printf(Thread.currentThread().getName() + " Writer writing");
			System.out.printf(" Counter value: %d\n", writerCount);
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
```

```
}
```

Bootstrapper.java

```java
package ques2;

import java.util.Random;
/**
 * Starting point of program
 * @author Amit
 *
 */
public class bootstrapper {
	public final static int READ_MAX = 20;//maximum number of readers
	public final static int WRITE_MAX = 3;//maximum number of writers
	public static ReaderWriterThread readerWriterthread = new ReaderWriterThread();//instantiate reader writer thread
	public static Random random = new Random();//create random variable

	static class readerThread extends Thread {
		public void run() {
			System.out.println("Reader :" + getName() + ": Start");
			while (true) {
				try {
					readerWriterthread.reader();
					int time = random.nextInt(3000);
					Thread.sleep(time);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		}
	}

	static class writerThread extends Thread {
		public void run() {
			System.out.print("Writer :" + getName() + ": Start");
			while (true) {
				try {
					readerWriterthread.writer();
					Thread.sleep(random.nextInt(3000));
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
			}
		}
	}

	public static void main(String[] args) {
		readerThread readerThreads[] = new readerThread[READ_MAX];
		writerThread writerThreads[] = new writerThread[WRITE_MAX];
		System.out.println("start");
		for (int i = 0; i < WRITE_MAX; ++i) {
			writerThreads[i] = new writerThread();
			writerThreads[i].start();
		}
		for (int i = 0; i < READ_MAX; ++i) {
			readerThreads[i] = new readerThread();
			readerThreads[i].start();
		}
	}
}
```

Output-

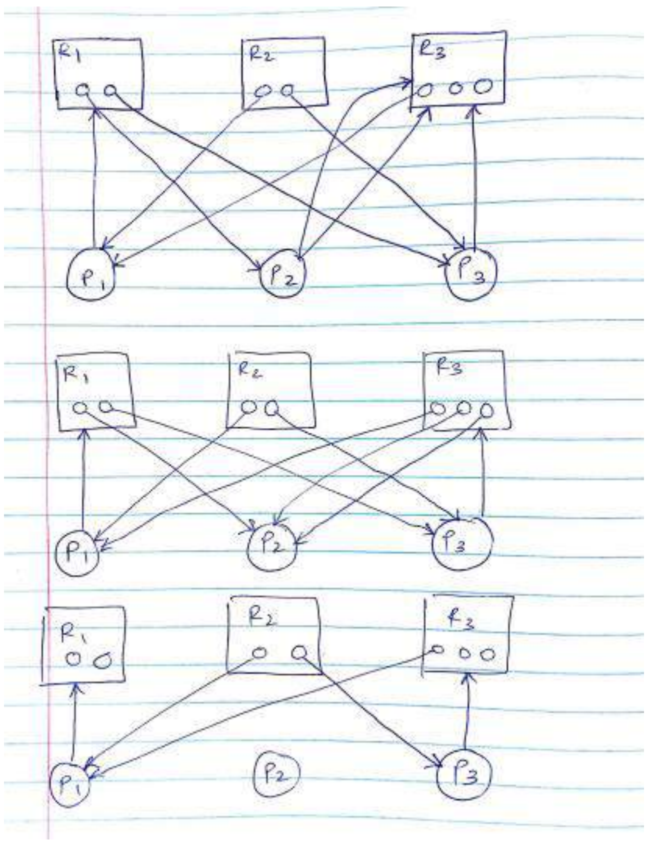
```
start
Writer :Thread-0: StartWriter :Thread-1: StartWriter :Thread-2: StartReader :Thread-3: Start
Reader :Thread-4: Start
Reader :Thread-5: Start
Reader :Thread-7: Start
Reader :Thread-8: Start
Reader :Thread-9: Start
Reader :Thread-6: Start
Reader :Thread-10: Start
Reader :Thread-11: Start
Reader :Thread-12: Start
Reader :Thread-13: Start
Reader :Thread-14: Start
Reader :Thread-15: Start
Reader :Thread-16: Start
Reader :Thread-18: Start
Reader :Thread-19: Start
Reader :Thread-21: Start
Reader :Thread-22: Start
Reader :Thread-17: Start
Reader :Thread-20: Start
Thread-0 Writer writing Counter value: 1
Thread-1 Writer writing Counter value: 2
Thread-2 Writer writing Counter value: 3
Thread-20 Reader reading Counter value: 3
Thread-3 Reader reading Counter value: 3
Thread-4 Reader readingThread-21 Reader reading Counter value: 3
 Counter value: 3
Thread-7 Reader reading Counter value: 3
Thread-9 Reader reading Counter value: 3
Thread-17 Reader reading Counter value: 3
Thread-13 Reader reading Counter value: 3
Thread-8 Reader reading Counter value: 3
Thread-18 Reader reading Counter value: 3
Thread-19 Reader reading Counter value: 3
Thread-16 Reader reading Counter value: 3
Thread-5 Reader reading Counter value: 3
Thread-12 Reader reading Counter value: 3
Thread-22 Reader reading Counter value: 3
Thread-10 Reader reading Counter value: 3
Thread-14 Reader reading Counter value: 3
Thread-11 Reader reading Counter value: 3
Thread-15 Reader reading Counter value: 3
Thread-6 Reader reading Counter value: 3
Thread-18 Reader reading Counter value: 3
Thread-14 Reader reading Counter value: 3
Thread-5 Reader reading Counter value: 3
Thread-7 Reader reading Counter value: 3
Thread-14 Reader reading Counter value: 3
Thread-22 Reader reading Counter value: 3
Thread-5 Reader reading Counter value: 3
Thread-20 Reader reading Counter value: 3
Thread-4 Reader reading Counter value: 3
Thread-1 Writer writing Counter value: 4
Thread-11 Reader reading Counter value: 4
Thread-17 Reader reading Counter value: 4
Thread-9 Reader reading Counter value: 4
Thread-4 Reader reading Counter value: 4
Thread-8 Reader reading Counter value: 4
Thread-14 Reader reading Counter value: 4
Thread-12 Reader reading Counter value: 4
Thread-0 Writer writing Counter value: 5
Thread-1 Writer writing Counter value: 6
Thread-14 Reader reading Counter value: 6
Thread-3 Reader reading Counter value: 6
Thread-6 Reader reading Counter value: 6
Thread-2 Writer writing Counter value: 7
Thread-5 Reader reading Counter value: 7
Thread-19 Reader reading Counter value: 7
Thread-20 Reader reading Counter value: 7
Thread-21 Reader reading Counter value: 7
Thread-2 Writer writing Counter value: 8
Thread-16 Reader reading Counter value: 8
Thread-7 Reader reading Counter value: 8
Thread-21 Reader reading Counter value: 8
Thread-3 Reader reading Counter value: 8
Thread-13 Reader reading Counter value: 8
Thread-18 Reader reading Counter value: 8
Thread-15 Reader reading Counter value: 8
Thread-10 Reader reading Counter value: 8
Thread-9 Reader reading Counter value: 8
Thread-6 Reader reading Counter value: 8
Thread-4 Reader reading Counter value: 8
Thread-15 Reader reading Counter value: 8
Thread-22 Reader reading Counter value: 8
Thread-5 Reader reading Counter value: 8
Thread-9 Reader reading Counter value: 8
```

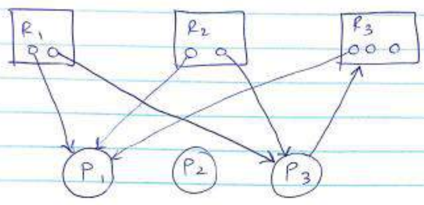
```
Thread-17 Reader reading Counter value: 8
Thread-20 Reader reading Counter value: 8
Thread-0 Writer writing Counter value: 9
Thread-3 Reader reading Counter value: 9
Thread-4 Reader reading Counter value: 9
Thread-10 Reader reading Counter value: 9
Thread-21 Reader reading Counter value: 9
Thread-8 Reader reading Counter value: 9
Thread-11 Reader reading Counter value: 9
Thread-9 Reader reading Counter value: 9
Thread-16 Reader reading Counter value: 9
Thread-8 Reader reading Counter value: 9
Thread-19 Reader reading Counter value: 9
```

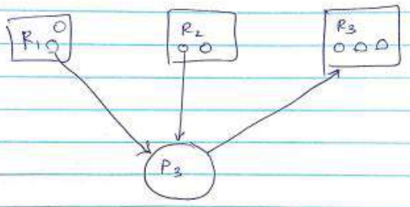
Question 3.
Consider a chain of processes P1, P2, ..., Pn implementing a multitiered client-server architecture. Process Pi is client of process Pi+1, and Pi will return a reply to Pi-1 only after receiving a reply from Pi+1. What are the main problems with this organization when looking at the request-reply performance at process P1?

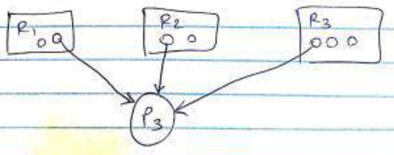
Answer –
P1 needs to wait until pi reaches last process. Then it waits for all processes to give reply in reverse order.
(If reply gets stuck in any of process whole thing will fail).

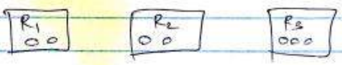
Question 4. Construct a general resource graph for the following scenario and determine if the graph is completely reducible: R1, R2, and R3 are reusable resources with a total of two, two, and three units. Process P1 is allocated one unit each of R2 and R3 and is requesting one unit of R1. Process P2 is allocated one unit of R1 and is requesting two units of R3. Process P3 is allocated one unit each of R1 and R2 and is requesting one unit of R3.



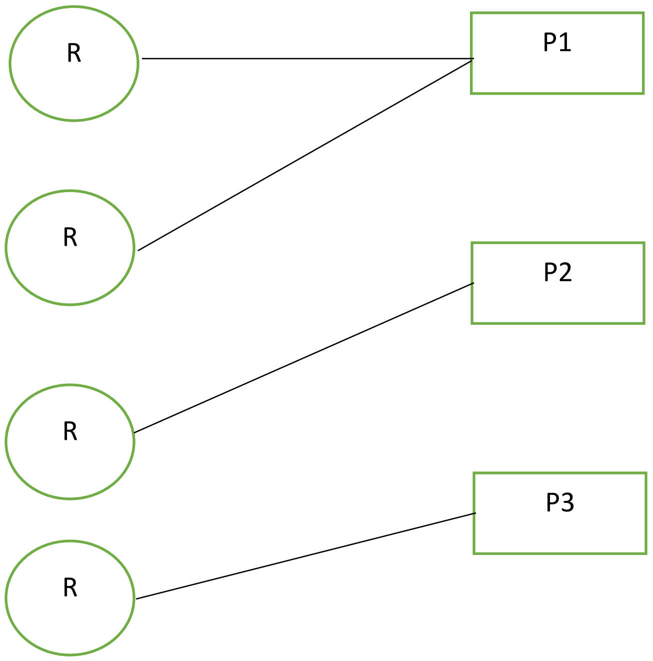

R1
R2
R3
P1
P2
P3


R1
R2
R3
P3


R1
R2
R3
P3


R1
R2
R3

Question 5. Consider a computer system that has 4 identical units of resource R. There are 3 processes each with a maximum claim of 2 units of resource R. Processes can request these resources in any way, i.e. two in one shot or one by one. The system always satisfies a request if enough resources are available. If the processes don't request any other kind of resource, is deadlock possible in this system? Why? (Show your steps clearly.)



It will not cause deadlock since it can utilize one resource of a kind. # of process < # of resources of different kind.
Consumed resource < available resources.

Discussion -

Question 5 is as per my knowledge not sure if it is totally correct.

Marks given – 69/70