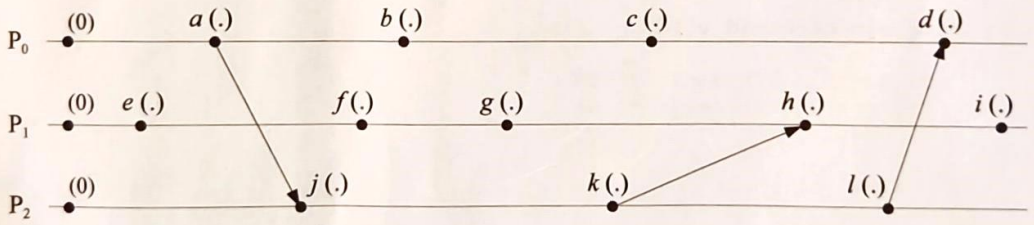
Part A (*12 points*)

1. Tanenbaum defines a *distributed system* as a *collection of independent computers that appears to its users as a single coherent system*. Using this definition as the starting point, try to elicit some of the distinctive features that this kind of systems present, namely *communication through message passing*, *failure handling* and *global internal state*. (***3 points***)

2. What distinguishes the *client-server* model from *peer-to-peer communication*? Give an example of each and explain how relevant they are in the examples you have presented. (***3 points***)

3. The diagram depicts the *time* evolution of three processes whose local clocks are scalar logical clocks synchronized according to the Lamport algorithm. (***3 points***)

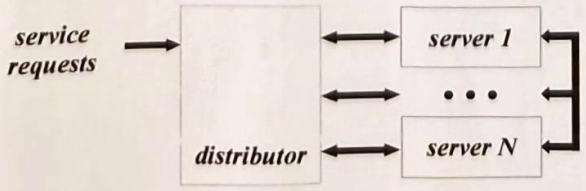


   i. Assign to the different events, specified by small letters (*a* ... *l*), their associated time stamp.

   ii. Give three examples of pairs of *concurrent* events and events *connected by a causality nexus* present in the schematics and justify why you have classified them in a such a way.

   iii. What is the difference between *total* and *partial* ordering of a group of events? Justify clearly your claim.

4. The schematics below depicts a simple solution for *load balancing* in a client-server model.



The role of the *distributor* is to forward service requests to any of the *servers*. The decision on which *server* a specific *service request* is forward to, is based on the *working load* each is presently enduring as perceived by the *distributor*. Bear also in mind that the copies of the shared region present in the *servers* must be continuously synchronized.
Describe in general terms how the whole system should operate. What kind of criteria should the distributor use to estimate the *working load* of a server? What kind of protocol should the *distributor* implement to assess if a given server is operating properly, or has failed? Present clearly your reasoning. (***3 points***)

Part B (8 points)

```
public class Semaphore
{
   private int val = 0;
   private int numbBlockThreads = 0;
   public synchronized void down ()
   {
      if (val == 0)
         { numbBlockThreads += 1;
           try
           { wait ();
           }
           catch (InterruptedException e) { }
         }
         else val -= 1;
   }
   public synchronized void up ()
   {
      if (numbBlockThreads != 0)
         { numbBlockThreads -= 1;
           notify ();
         }
         else val += 1;
   }
}

public class GenRegion
{
   private int n = 0;
   private int [] valSet = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
   private Semaphore access;
   public GenRegion ()
   {
      access = new Semaphore ();
      access.up ();
   }
   public int produceVal ()
   {
      int val = 0;
      access.down ();
      if (n < valSet.length)
         { val = valSet[n];
           n += 1;
         }
      access.up ();
      return val;
   }
}

public class LinRegion
   private int mem = 0;
   private Semaphore [] stat;
   public LinRegion ()
   {
      stat = new Semaphore[3];
      for (int i = 0; i < 3; i++)
      { stat[i] = new Semaphore ();
        if (i != 2) stat[i].up ();
      }
   }
```

```
  public void putVal (int val)
  {
     stat[1].down ();
     stat[0].down ();
     if ((val % 2) == 0)
        mem = val + val % 100;
        else mem = val;
     stat[0].up ();
     stat[2].up ();
  }

  public int getVal ()
  {
     int val;
     stat[2].down();
     stat[0].down();
     val = mem;
     stat[0].up ();
     stat[1].up ();
     return val;
  }
}

public class ThreadType1 extends Thread
{
   private int id;
   private GenRegion gen = null;
   private LinRegion inStore = null, outStore = null;
   public ThreadType1 (int id, GenRegion gen, LinRegion inStore, LinRegion outStore)
   {
      this.id = id;
      this.gen = gen;
      this.inStore = inStore;
      this.outStore = outStore;
   }
   public void run ()
   {
      int gVal, tVal;
      boolean val = true, transf = (inStore != null);
      do
      { try
        { sleep ((int) (1 + 10*Math.random ()));
        }
        catch (InterruptedException e) {};
        if (val)
           { gVal = gen.produceVal ();
             if ((gVal != 0) || !transf)
                outStore.putVal (100 * id + gVal);
             val = (gVal != 0);
           }
        try
        { sleep ((int) (1 + 10*Math.random ()));
        }
        catch (InterruptedException e) {};
        if (transf)
           { tVal = inStore.getVal ();
             if (((tVal % 100) != 0) || !val)
                outStore.putVal (tVal);
             transf = ((tVal % 100) != 0);
           }
      } while (val || transf);
   }
}
```

```
public class ThreadType2 extends Thread
{
   private LinRegion inStore = null;
   public ThreadType2 (LinRegion inStore)
   {
      this.inStore = inStore;
   }
   public void run ()
   {
      int cVal, pVal;
      do
      { cVal = inStore.getVal ();
        pVal = cVal % 100;
        if (pVal != 0)
           { cVal = cVal / 100;
             System.out.print ("The value produced by thread " + cVal + " was " +
                               pVal + "!\n");
           }
       } while (pVal != 0);
   }
}

public class SimulSituation
{
   public static void main (String [] args)
   {
      LinRegion [] store = new LinRegion [3];
      for (int i = 0; i < 3; i++)
        store[i] = new LinRegion ();
      GenRegion gen = new GenRegion ();
      ThreadType1 [] thr1 = new ThreadType1[3];
      for (int i = 0; i < 3; i++)
        if (i != 0)
           thr1[i] = new ThreadType1 (i+1, gen, store[i-1], store[i]);
           else thr1[i] = new ThreadType1 (i+1, gen, null, store[i]);
      ThreadType2 thr2 = new ThreadType2 (store[2]);
      thr2.start ();
      for (int i = 0; i < 3; i++)
        thr1[i].start ();   }
}
```

1. Representing the active entities by circles and the passive entities by squares, sketch a diagram which illustrates the interaction that is taking place and describe in simple words the role played by the *threads* of each type (do not use more than one or two sentences in your explanation). (***2 points***)
2. Write the output of a single run. Bear in mind that, because of the concurrency and the randomness which were introduced, there is not an unique printing. (***2 points***)
3. What is the role played by each of the elements of the array `stat` of data type `LinRegion` in the interaction? Explain in clear terms how the application ends. (***2 points***)
4. Change the program so that the *thread* of type 2 will process, whenever possible, pairs of values, by adding them up, instead of a single value in each stage of the iterative cycle, as it happens now. Start by pointing out the changes that have to be made to the code. (***2 points***)