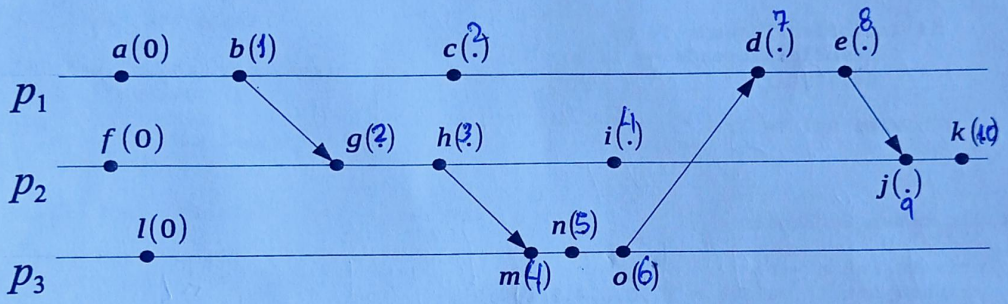
Parte A (12 points)

1. Explain what is *fault tolerance* and why it is so important in the context of distributed systems. Which are the basic strategies commonly used to handle failures? Present clearly your claims. (***3 points***)
2. Describe schematically, adding the comments you deem appropriate about their functionality, the advantages and disadvantages, of the three variants of the client-server model that were studied. Can they be implemented using either the message passing or the remote objects paradigms, or are they specific to one of these paradigms? Present clearly your reasoning. (***3 points***)
3. The schematics bellow describes the *temporal* evolution of three processes whose local clocks are scalar logical clocks synchronized according to the Lamport algorithm. (***3 points***)



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

b) State for the following event pairs, *a-j*, *f-c*, *b-i*, *e-m*, *i-o* and *i-e*, if they are concurrent events (||) or sequential ones (->).

4. Suppose we want to ensure service availability in a *client-server* model. In order to achieve this aim, a *resource replication* variant where the server is installed in two hardware platforms, is implemented. In principle, only one of the servers is active at a time and interacts with the clients. When it fails, the other starts operating immediately, replacing it.
   Draw a functional diagram that depicts the organization and list three problems which must be solved for the system to work properly. Justify clearly your claims. (***3 points***)

Parte 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};
   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 StoreRegion
{
   private int mem = 0;
   private Semaphore access;
   private Semaphore [] stat;
   public StoreRegion ()
   {
      access = new Semaphore ();
      access.up ();
      stat = new Semaphore [2];
      for (int i = 0; i < 2; i++)
        stat[i] = new Semaphore ();
      stat[0].up ();
   }
```

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

public class Resource
{
   private Semaphore access;
   public Resource ()
   {
      access = new Semaphore ();
      access.up ();
   }
   public void printVal (int id, int val)
   {
      access.down ();
      if ((val % 100) != 0)
         System.out.println ("O valor processado por " + (val/100) + " e por " +
                             id + " foi " + (val%100) + ".");
      access.up ();
   }
}

public class ThreadType1 extends Thread
{
   private int id;
   private GenRegion gen;
   private StoreRegion [] store;
   public ThreadType1 (int id, GenRegion gen, StoreRegion [] store)
   {
      this.id = id;
      this.gen = gen;
      this.store = store;
   }
   public void run ()
   {
      int ind, val;
      do
      { try
        { Thread.sleep ((long) (1 + 10*Math.random ()));
        }
        catch (InterruptedException e) {};
        val = gen.produceVal ();
        ind = (val == 0) ? id - 1 : val%4;
        store[ind].putVal (val + 100 * id);
      } while (val != 0);
   }
}
```

```
public class ThreadType2 extends Thread
{
   private int id;
   private StoreRegion [] store;
   private Resource writer;
   public ThreadType2 (int id, StoreRegion [] store, Resource writer)
   {
      this.id = id;
      this.store = store;
      this.writer = writer;
   }
   public void run ()
   {
      int val0, val1;
      do
      { try
        { Thread.sleep ((long) (1 + 10*Math.random ()));
        }
        catch (InterruptedException e) {};
        val0 = store[0].getVal ();
        writer.printVal (id, val0);
        val1 = store[1].getVal ();
        writer.printVal (id, val1);
      } while ((val0 % 100 + val1 % 100) != 0);
   }
}

public class SimulSituation
{
   public static void main (String [] args)
   {
      GenRegion gen = new GenRegion ();
      StoreRegion [] store = new StoreRegion [4];
      StoreRegion [] hStore0 = new StoreRegion [2], hStore1 = new StoreRegion [2];
      Resource writer = new Resource ();
      for (int i = 0; i < 4; i++)
      { store[i] = new StoreRegion ();
        if (i%2 == 0)
           hStore0[i/2] = store[i];
           else hStore1[i/2] = store[i];
      }
      ThreadType1 [] thr1 = new ThreadType1 [4];
      for (int i = 0; i < 4; i++)
        thr1[i] = new ThreadType1 (i+1, gen, store);
      ThreadType2 [] thr2 = new ThreadType2 [2];
      for (int i = 0; i < 2; i++)
        thr2[i] = new ThreadType2 (i+1, (i == 0) ? hStore0 : hStore1, writer);
      for (int i = 0; i < 2; i++)
        thr2[i].start ();
      for (int i = 0; i < 4; 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 performed 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 the elements of the array stat of data type StoreRegion in the interaction? Explain in clear terms how the application ends. (***2 points***)
4. Change the program so that the *threads* of type 2 process a single value, instead of two values, at each stage of the iterative cycle as it happens now. Start by pointing out the changes that have to be made in the code. They should be minimal. (***2 points***)