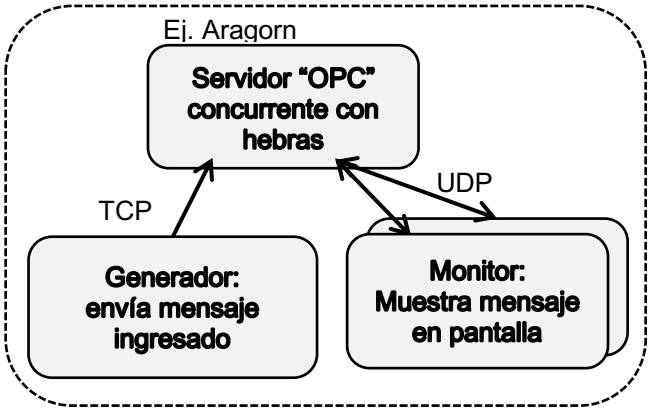
Segundo Certamen
Tiempo 150 min. Al terminar, entregar sus respuestas vía AULA.

Se desea implementar un sistema compuesto de tres programas:
generador <puerto> <servidor>: este programa establece una conexión TCP al puerto y servidor señalados que se encuentra en la nube (por ejemplo, Aragorn), y sube cada línea ingresada por el usuario vía teclado. Suponga que este programa ya existe.
opc <puerto>: este programa escucha por conexiones TCP y requerimientos UDP en puerto indicado en su ejecución. Para atender a múltiples clientes opc **debe usar hebras**. Ante una conexión TCP, opc almacena la última línea recibida desde el único cliente TCP que atiende (maneja un buffer para sólo una línea de hasta 256 bytes y atiende a sólo un cliente). opc termina cuando el cliente TCP cierra la conexión.
Ante la llegada de una letra "n" desde un cliente UDP, opc envía la última línea recibida al cliente que hizo el requerimiento.
monitor <puerto> <servidor>: este programa envía la letra "n" vía UDP al servidor opc cada vez que el usuario presiona "enter" o "return" (según su teclado). Luego, **monitor** muestra por pantalla la línea de texto enviada desde opc. Obs. Suponga que todos los mensajes UDP llegan a destino.



P1) 30 puntos. Programe el servidor opc en lenguaje C.

```
/* This program is an adaptation of question 1 solution in
http://profesores.elo.utfsm.cl/~agv/elo330/2s19/grades/c2_2s19_Sol.pdf */

#include <stdio.h>  /* printf */
#include <sys/socket.h>  /* socket, bind, listen */
#include <arpa/inet.h>  /* htonl */
#include <string.h>   /* strcpy */
#include <pthread.h>

pthread_mutex_t mutex=PTHREAD_MUTEX_INITIALIZER;
char line[256]="";

void * udpMonitorServer( void * p_port) {
   int port = *((int *)p_port);
   int udpSocket, rc, len;
   struct sockaddr_in name, from;
   char buf[10];
```

```
    udpSocket = socket(AF_INET, SOCK_DGRAM, 0);
    name.sin_family = AF_INET;
    name.sin_port = htons(port);
    name.sin_addr.s_addr = htonl(INADDR_ANY);
    len = sizeof(struct sockaddr_in);
    if( bind(udpSocket, (struct sockaddr *) &name, len))
        printf("bind error");
    len = sizeof(from);
    for(;;){
        printf("\n... UDP server is waiting...\n");
        if ((rc=recvfrom(udpSocket, buf, sizeof(buf), 0, (struct sockaddr *)&from, &len)) < 0)
            perror("receiving datagram  message");
        if ((rc > 0) && (buf[0]=='n') ){
            pthread_mutex_lock(&mutex);
            if (sendto(udpSocket, line, strlen(line), 0,(struct sockaddr *) &from, sizeof(from)) <0 )
                perror("sending line");
            pthread_mutex_unlock(&mutex);
        }
    }
}

#include <pthread.h>
#include <unistd.h>   /* read,  close */
#include <stdlib.h>  /* exit */

int main(int argc, char *argv[])  {
    int port = atoi(argv[1]);
    pthread_t tid;
    int welcomeSocket, generatorSocket, len;
    struct sockaddr_in name;
    int n;
    char buff[buffSize];

    pthread_create(&tid, NULL, udpMonitorServer, &port);

    welcomeSocket = socket(AF_INET, SOCK_STREAM, 0);
    name.sin_family = AF_INET;
    name.sin_port = htons(port);
    name.sin_addr.s_addr = htonl(INADDR_ANY);
    len = sizeof(struct sockaddr_in);
    if( bind(welcomeSocket, (struct sockaddr *) &name, len))
        printf("bind error");
    listen(welcomeSocket, 5);
    printf("\n... TCP server is waiting...\n");
    generatorSocket = accept(welcomeSocket, (struct sockaddr *) &name, &len);
    do {
        n = read(generatorSocket, buff, sizeof(buff));
        if (n > 0) {
            buff[n]='\0';
            printf("\n... TCP server received:%s\n", buff);
            pthread_mutex_lock(&mutex);
            strcpy(line, buff);
            pthread_mutex_unlock(&mutex);
        }
    } while ( n > 0);
    close(generatorSocket);
```

```
    close(welcomeSocket);
    exit(0);
}
```

P2a) 30 puntos. Programe el cliente monitor en Java.

```
/* This program is an adaptation of
http://profesores.elo.utfsm.cl/~agv/elo330/Java/Networking/UDP_Client.java  */
import java.net.*;
import java.io.*;

public class Monitor {
  // Can listen & send on the same socket:
  private DatagramSocket s;
  private InetAddress hostAddress;
  private byte[] buf = new byte[256];
  private DatagramPacket dp = new DatagramPacket(buf, buf.length);
  private int id, port;

  public Monitor(String port_s, String serverName) {
    try {
      // Auto-assign port number:
      s = new DatagramSocket();
      hostAddress = InetAddress.getByName(serverName);
      port = Integer.parseInt(port_s);
    } catch(UnknownHostException e) {
      System.err.println("Cannot find host");
      System.exit(1);
    } catch(SocketException e) {
      System.err.println("Can't open socket");
      e.printStackTrace();
      System.exit(1);
    }
    System.out.println("Monitor starting");
  }
  public void sendData() {
    String outMessage = "n";
    // Make and send a datagram:
    byte[] sendData = outMessage.getBytes();
    DatagramPacket request = new DatagramPacket(sendData, sendData.length, hostAddress, port);
    BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
    try {
      while (true) {
        in.readLine();
        s.send(request);
        // Block until it reply is back:
        s.receive(dp);
        // Print out the reply:
        String rcvd = new String(dp.getData(), 0, dp.getLength());
        System.out.println(rcvd);
      }
    } catch(IOException e) {
      e.printStackTrace();
      System.exit(1);
    }
  }
```

```
  public static void main(String[] args) {
     new Monitor(args[0], args[1]).sendData();
  }
}
```

P2b) 10 puntos. Para la solución que usted entregó, describa brevemente qué idea daría usted a otro programador para que su solución no se cuelgue (quede en estado de espera y no reaccione a nuevos ingresos de "return") ante la pérdida de un datagrama UDP.

Ante la pérdida de un datagrama, este cliente se cuelga pues queda esperando la respuesta al requerimiento que nunca llega. Una solución sería programar la recepción de la respuesta en una hebra separada que se crearía por cada requerimiento enviado. Esta hebra debería incluir además un temporizador que luego de un tiempo destruya la hebra. Otra opción (autoría de Tomás Ibaceta), es invocar el método setSoTimeout(int timeout) de DatagramSocket para poner límite a la espera.

P3) 30 puntos. Programe el servidor opc en Java.

```
/* This solution is an adaptation of:
http://profesores.elo.utfsm.cl/~agv/elo330/Java/networking/Threaded/ThreadedEchoServer.java
and
http://profesores.elo.utfsm.cl/~agv/elo330/Java/Networking/UDP_EchoServer.java
*/
import java.io.*;
import java.net.*;
public class Opc {
   public static void main(String[] args ) {
      int port = Integer.parseInt(args[0]);
      Line line = new Line();
      try {
         new UdpMonitorServer(port, line).start();
         ServerSocket s = new ServerSocket(port);
         Socket incoming = s.accept( );
         BufferedReader in = new BufferedReader(new InputStreamReader(incoming.getInputStream()));
         boolean done = false;
         while (!done) {
            String str = in.readLine();
            if (str == null)
               done = true;
            else
               line.setLine(str);
         }
         incoming.close();
      }
      catch (Exception e) {
         e.printStackTrace();
      }
      System.exit(0);
   }
}

class Line {
   private String line;
   public synchronized void setLine(String l) {
      line = l;
```

```
  }
  public synchronized String getLine() {
    return line;
  }
}

class UdpMonitorServer extends Thread {
  private Line line;
  private DatagramSocket socket;
  public UdpMonitorServer(int port, Line l) {
    try {
      socket = new DatagramSocket(port);
    } catch (Exception e) {
      e.printStackTrace();
    }
    line=l;
    System.out.println("Server started");
  }

  public void run() {
    byte[] buf = new byte[256];
    DatagramPacket dp = new DatagramPacket(buf, buf.length);
    try {
      while (true) {
        socket.receive(dp);
        if (dp.getData()[0]=='n') {
          dp.setData (line.getLine().getBytes());
          dp.setAddress(dp.getAddress());
          dp.setPort(dp.getPort());
          socket.send(dp);
         }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```