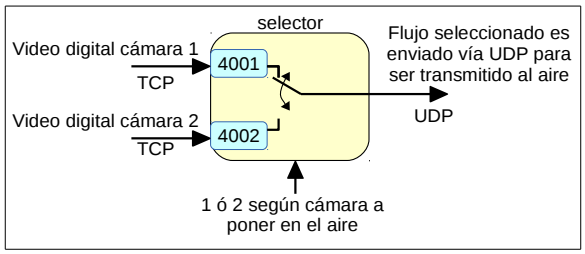
Segundo Certamen Tiempo 90 [min]

1.- **Contexto:** Como trabajo de titulación, uno de sus compañero desarrolló un programa *analizador* (analizador.cpp) de un flujo de datos correspondiente a una transmisión de televisión digital. El programa *analizador* lee todos los datos desde un archivo, ingresado como parámetro, y muestra en pantalla la descripción del flujo (resoluciones, codificación, idioma, close caption, cada canal de audio, etc).
Hoy el Departamento de Electrónica tiene un transmisor profesional de televisión digital (Eitv) capaz de generar, en tiempo real, el flujo procesado por el analizador. Una de las opciones de Eitv es enviar el flujo vía UDP hacia una IP y puerto dados. Pero el programa *analizador* sólo procesa datos desde archivo.
Problema: Para probar el programa *analizador* en condiciones reales, **se le pide a usted crear un programa servidor UDP, servidorTVD.c,** el cual recibe datos en un puerto ingresado como parámetro y luego los envíe a la salida estándar. Con su respuesta será posible usar el programa *analizador* haciendo:

```
$ mkfifo puenteFifo
$ analizador puenteFifo&    /* el programa analizador queda esperando datos desde puenteFifo */
$ servidorTVD  2345  > puenteFifo   /* redirigimos los datos de su programa al puenteFifo */
```

```
#include <netinet/in.h>
#include <stdlib.h>
#define DATAGRAM_SIZE 500 /* depende el paquete UDP
        más grande enviado por el servidor de video */

int main(int argc, char * argv[])
{
   char buf[DATAGRAM_SIZE];
   int s, n, len;
   struct sockaddr_in name;

   s = socket(AF_INET, SOCK_DGRAM, 0);
   name.sin_family = AF_INET;
   name.sin_port = htons(atoi(argv[1]));
   name.sin_addr.s_addr = htonl(INADDR_ANY);
   len = sizeof(struct sockaddr_in);

   bind(s, (struct sockaddr *) &name, len);

   while ((n = recv(s, buf, sizeof(buf), 0)) > 0)
      write(1, buf, n);

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

2.- **Contexto**: para transmitir TVD por aire, el equipo profesional del Departamento (Eitv) puede enviar su flujo digital a una tarjeta moduladora y antena en lugar de una IP y puerto UDP. En este caso la señal es puesta en el aire y puede ser vista por televisores que cumplen con la norma chilena. El equipo Eitv puede tomar el video digital a transmitir desde un archivo o desde un puerto UDP local (éste se configura en su interfaz web). Un problema es cómo cambiar en tiempo real el video que se envía al aire seleccionándolo a voluntad desde distintos orígenes.
Problema: Desarrolle el programa selector.c. Éste programa crea 2 puertos de escucha TCP (4001 y 4002) y pregunta por consola cuál de ellos debe ser re-enviado como flujo UDP hacia el puerto configurado en el equipo Eitv. La pregunta por consola se mantiene para permitir cambios cuando se quiera. Los datos que llegan al otro puerto TCP deben ser descartados. Su implementación debe usar hebras.




```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <netinet/in.h>
#include <netdb.h>
#include <string.h>

struct sockaddr_in udpTo;
int udpSock, len;
int camera=1;
pthread_mutex_t myMutex = PTHREAD_MUTEX_INITIALIZER;/* pudo ser rwlock*/

void * TCP_UDP_Forwarding(void * arg) {
  int port = *(int*)arg;
  int myCamera = port-4000;
  int s, ns, len, n;
  char buf[1024];
  struct sockaddr_in name;

  s = 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(s, (struct sockaddr *) &name, len)) {
    printf("bind error");
    exit(-1);
  }
  listen(s, 5);

  ns = accept(s, (struct sockaddr *) &name, &len);
  while ((n = recv(ns, buf, sizeof(buf), 0)) > 0) {
    pthread_mutex_lock( &myMutex );
    if (camera==mCamera)
      sendto(udpSock, buf, n, 0,(struct sockaddr*) &udpTo, len);
    pthread_mutex_unlock( &myMutex );
  }
  close(ns);
  close(s);
  return NULL;
}

int main(int argc, char * argv[]) {
  struct hostent *hp;
  pthread_t tid;
  int port1, port2, opcion;
  if (argc < 3) {
```

```
    printf("Usage: %s <UDP server name>  <port>\n", argv[0]);
    exit(0);
  }
  udpSock = socket(AF_INET, SOCK_DGRAM, 0);
  hp = gethostbyname(argv[1]);
  udpTo.sin_family = AF_INET;
  udpTo.sin_port = htons(atoi(argv[2]));
  memcpy(&udpTo.sin_addr, hp->h_addr_list[0], hp->h_length);
  len = sizeof(struct sockaddr_in);

  port1=4001;
  pthread_create(&tid, NULL, TCP_UDP_Forwarding, &port1);
  port2=4002;
  pthread_create(&tid, NULL, TCP_UDP_Forwarding, &port2);
  do {
    printf("Seleccione camera al aire:\n 1: Camara 1\n 2: Camara 2 \n 3: exit\n Opcion:");
    scanf("%d",&opcion);
    switch(opcion) {
    case 1:
    case 2:
      pthread_mutex_lock( &myMutex );
      camera=opcion;
      pthread_mutex_unlock( &myMutex );
      break;
    case 3: exit(0);
    }
  } while(opcion !=3 );
}
```

3.- Repita el problema 2 pero en Java.

Ayuda: Sobre programación de socket en Java, puede dar una mirada a:
http://profesores.elo.utfsm.cl/~agv/elo322/SocketsProgramming/aPantalla/

Para leer desde teclado puede revisar:
http://profesores.elo.utfsm.cl/~agv/elo329/JavaProg/InputExample/InputExample.java

```
import java.io.*;
import java.net.*;
import java.util.Scanner;
class UDPclient {
    private DatagramSocket clientSock;
    private InetAddress IPAddress;
    private int port;
    private int camera;

    public UDPclient (String server, int _port) {
        try {
            clientSock = new DatagramSocket();
            IPAddress = InetAddress.getByName(server);
        } catch (Exception e) {
        }
        port = _port;
        camera=1;
    }
    public void sendBuff(byte[] buf) {
        DatagramPacket packet =
```

```
            new DatagramPacket(buf, buf.length, IPAddress, port);
        try{
            clientSock.send(packet);
        } catch (IOException e) {}
    }
    public synchronized int getCamera(){
        return camera;
    }
    public synchronized void setCamera(int cam){
        camera = cam;
    }
}

class TCP_UDP_Forwarding extends Thread {
    private ServerSocket welcomeSocket;
    private UDPclient udpClient;
    private int myCamera;

    public TCP_UDP_Forwarding (int listeningPort, UDPclient _udpClient){
        try {
            welcomeSocket = new ServerSocket(listeningPort);
        }catch (IOException e){}
        udpClient = _udpClient;
        myCamera=listeningPort-4000;
    }
    public void run () {
        InputStream inFromClient=null;
        int rc;
        try {
            Socket connectionSocket = welcomeSocket.accept();
            inFromClient =  connectionSocket.getInputStream();
        }catch (IOException e) {}
        byte [] buf = new byte[1024];
        while(true) {
            try {
                rc=inFromClient.read(buf);
            } catch (IOException e){}
            if (udpClient.getCamera()==myCamera)
                udpClient.sendBuff(buf, rc);
        }
    }
}

class Selector {
    public static void main (String argv[]) throws Exception {
        if ( argv.length !=2 ){
            System.out.println("Usage: java selector  <UDP server name>  <port>\n");
            System.exit(-1);
        }
        UDPclient udp = new UDPclient(argv[0], Integer.parseInt(argv[1]));
        TCP_UDP_Forwarding tcpToUdp1 = new TCP_UDP_Forwarding(4001, udp);
        TCP_UDP_Forwarding tcpToUdp2 = new TCP_UDP_Forwarding(4002, udp);
        tcpToUdp1.start();
        tcpToUdp2.start();
        Scanner s= new Scanner (System.in);
        int opcion;
        do {
```

```
                System.out.print("Seleccione camara al aire:\n 1: Camara 1\n 2: Camara 2 \n 3: exit\n Opcion:");
                opcion = s.nextInt();
                switch (opcion) {
                case 1:
                case 2: udp.setCamera(opcion); break;
                case 3: System.exit(0);
                }
            } while (true);
    }
}
```