Segundo Certamen
Tiempo 11:30 hrs - 13:10 hrs.
Responder un problema por página

1.- (50 puntos) En la actualidad el tráfico multicast generalmente no atraviesa routers. Se tiene dos subredes en Internet donde un grupo de amigos desea correr un juego que hace uso de comunicación multicast. Para intercambiar tráfico multicast entre ambas subredes, usted ofrece desarrollar la aplicación mfw (por multicast forwarding). Su sintaxis es:
$ mfw <dirección multicast> <puerto multicast> <puerto unicast> <IP remota>
mfw escucha todos los paquetes multicast dirigidos al "puerto multicast" y los reenvía al "puerto unicast" de la máquina remota. También escucha los paquetes UDP que llegan a ese puerto unicast y los reenvía hacia el mismo grupo multicast. Su programa termina con Control-C. Se sabe que los datagramas del juego no superan los 512 bytes.
Se pide programar la aplicación mfw en C y usando select para esperar por llegada de paquetes multicast locales o datagramas UDP desde la máquina remota.
Nota: Es OK considerar que todos los llamados son exitosos (no verificar errores en llamados al sistema)

```
/*
NAME:       multicast forwarding
SYNOPSIS:    mfw <multicast group> <multicast port> <unicat port> <remote server>
*/

#include <netinet/ip_icmp.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <string.h>

int main( int argc, char * argv[] )
{
 int  ms,us;
  struct   sockaddr_in mAddress,uAddress ;
  char buf[512];
  int n;
  struct ip_mreq mreq;  /* multicast group info structure */
  unsigned long int mcGroup;
  int reuse;
  char ttl, loop;
  fd_set readfds, readfdsCopy;

  /* Preparo socket de recepcion de trafico multicast */
  ms = socket (PF_INET,SOCK_DGRAM,0);                                  4 pts
  reuse=1;
  setsockopt(ms,SOL_SOCKET,SO_REUSEADDR, (char *) &reuse, sizeof(reuse));  3 pts

  mAddress.sin_family = AF_INET;
  mAddress.sin_addr.s_addr = INADDR_ANY;
  mAddress.sin_port = htons(atoi(argv[2]));
  bind(ms, (struct sockaddr *)&mAddress, sizeof(mAddress));    5 pts

  /* Join Multicast Group */
  mcGroup = inet_addr(argv[1]);
  mreq.imr_multiaddr.s_addr = mcGroup;
  mreq.imr_interface.s_addr = INADDR_ANY;
  setsockopt(ms,IPPROTO_IP,IP_ADD_MEMBERSHIP,(char *) &mreq, sizeof(mreq));    5 pts
  /* Preparo direccion multicast destino para lo llegado de otra subred. */
  mAddress.sin_addr.s_addr = mcGroup;
 /* loopback y TTL  */
  loop = 1;
```

```
  setsockopt(ms,IPPROTO_IP,IP_MULTICAST_LOOP,(char *) &loop, sizeof(u_char));   3 pts
  ttl = 1;
  setsockopt(ms,IPPROTO_IP,IP_MULTICAST_TTL,(char *) &ttl, sizeof(u_char));       3 pts
  /* Hasta aqui listo socket lado mutlicast */
  /* Ahora preparar socket lado Unicast */

   /* Create unicast socket. */
   us = socket(PF_INET, SOCK_DGRAM, 0);                                    4 pts

   /* Create the address of the server. */
   uAddress.sin_family = AF_INET;
   uAddress.sin_port = htons(atoi(argv[3]));
   uAddress.sin_addr.s_addr = htonl(INADDR_ANY); /* Use the wildcard address.*/
   bind(us, (struct sockaddr *) &uAddress, sizeof(uAddress));                          5 pts
  /* Preparo direccion unicast destino para enviar a otra subred. */
   uAddress.sin_addr.s_addr = inet_addr(argv[4]);

   FD_ZERO(&readfdsCopy);
   FD_SET(ms,&readfdsCopy);
   FD_SET(us,&readfdsCopy);
 /* forward data */
 for(;;){
   memcpy(&readfds, &readfdsCopy, sizeof(fd_set));
   n = select(FD_SETSIZE, &readfds, (fd_set *) 0, (fd_set *) 0, NULL);          Buen uso select 9 pts
   if (n > 0) {
     if (FD_ISSET(ms, &readfds)) {                                  Buen reenvío ambos casos 9 pts
       n = recv(ms, buf, sizeof(buf), 0);
       sendto(us, buf, n, 0, (struct sockaddr*) &uAddress, sizeof(uAddress));
     }
     if (FD_ISSET(us, &readfds)) {
       n = recv(us, buf, sizeof(buf), 0);
       sendto(ms, buf, n, 0, (struct sockaddr*) &mAddress, sizeof(mAddress));
     }
   }
 }
 /* Leave multicast group */
 setsockopt(ms,IPPROTO_IP,IP_DROP_MEMBERSHIP, (char *) &mreq,sizeof(mreq));
}
```

2.- (25 puntos) Se tienen la siguiente clase en Java:

```
class A {
  public int a1() { … }
  public int a2() { … }
  public int e1() { … }
  public int e2() { … }
…
}
```

Introduzca los cambios necesarios para que, en hebras que acceden a una instancia de A, los métodos a1 y a2 se ejecuten en forma alternada (a1 debe ir primero) y sus métodos e1 y e2 en forma excluyente. Ejecución de a1 o a2 puede ser concurrentes con e1 o e2.

```
class A {
  private String lock="";                Uso de variable para otro monitor 5 pts
  private boolean uno=true;              Uso de variable para alternancia 5 pts
  public int a1() {
    synchronized (lock) {                Buen uso de wait y notify 5 pts
      if (!uno)
        try {
```

```
            lock.wait();
          } catch (InterruptedException e) { }
        uno=false;
        ...
        lock.notify();
    }
    public int a2() {
      synchronized (lock) {
        if (uno)
          try {
            lock.wait();
          } catch (InterruptedException e) { }
        uno=true;
        ...
        lock.notify();
    }
    public synchronized int e1() { ... }      Exclusión mutua en e1 y e2 10 pts
    public synchronized int e2() { ... }
...
}
```

3.- (25 puntos) Se desea conocer y controlar el estado (encendido/apagado) de un interruptor remoto usando Remote Method Invocation (RMI). En el computador de la sala donde se encuentra el interruptor, usted dispone de la clase Controller la cual ya está implementada y tiene los métodos mostrados en su directorio de trabajo.

```
class Controller {
  public void setSwitchState(boolean on){  /* fija estado encendido o apagado de la luz */
        …
  }
  public boolean getSwitchStatus() {  /* retorna estado de luz, true es encendido */
        …
  }
}
```

a) Muestre el contenido de los archivos en lado **servidor** necesarios para cambiar el estado del interruptor remoto desde un programa cliente (no pedido) y usando RMI.

b) Muestre la secuencia de comandos necesarios en lado servidor para compilar y luego dejar el servicio disponible.

```
a) Lado servidor:
import java.rmi.*;
/**   The interface for remote Switch objects.*/
public interface Switch extends Remote {                    Interfaz 6 pts
   void setState(boolean on) throws RemoteException;
   void boolean getState(boolean on) throws RemoteException;
}
// Implementacion de Interfaz
public class SwitchImpl implements Switch {       Implementación de interfaz 6 pts
   public SwitchImpl(Controller c) {
        this.c = c;
   }
   public void setState( boolean on ) {
        return c.setSwitchState(on);
   }
   public boolean getState( {
        return c.getSwitchState();
   }
  private Controller c;
}
   // Programa Servidor que crea y deja disponible el servicio
import java.rmi.registry.Registry;
```

```
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class SwitchServer {                                    Programa servidor 5 pts
  public static void main(String args[]) {
    try   {
          Controller c = new Controller();
          SwitchImpl s = new SwitchImpl(c);
          Switch stub1 =(Switch) UnicastRemoteObject.exportObject(s, 0);
          Registry registry = LocateRegistry.getRegistry();
          registry.bind("switch", stub1);
    }
    catch(Exception e){
      System.err.println("Server exception: " + e.toString());
      e.printStackTrace();
    }
  }
}
```

Lado Cliente (**No pedido**):

```
import java.rmi.*;
/**   The interface for remote Switch objects.*/
public interface Switch extends Remote {
   void setState(boolean on) throws RemoteException;
   void boolean getState(boolean on) throws RemoteException;
}
/**** Programa  ****/
import java.rmi.*;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class ChangeSwitch  {
  public static void main(String[] args) {
    System.setProperty("java.security.policy", "client.policy");
    System.setSecurityManager(new RMISecurityManager());
    String host = (args.length < 1) ? null : args[0];  // args[0] debe ser la máquina
    try  {
      Registry registry = LocateRegistry.getRegistry(host);
      Switch s = (Switch)registry.lookup("switch");
      s.setState(!s.getState());
    }
    catch(Exception e)  {
      System.err.println("Client Exception: " + e.toString());
      e.printStackTrace();
    }
  }
}
/* Archivo de politica client.polcy*/
grant {
  permission java.net.SocketPermission
    "*:1024-65535", "connect,accept";
};
```

b)

Comandos en lado servidor:

```
$ javac SwitchServer.java        2 pts  cada comando, total 8 pts.
$ export CLASSPATH="."
$ rmiregister &
$ java SwitchServer &
```

Comandos en lado cliente **(No pedido)**:

```
$ javac ChangeSwitch.java
$ java ChangeSwitch
```