MULTICAST (VERSIÓN 4)

GRUPO: 4CM3
EQUIPO 3
INTEGRANTES:

- MAYA ROCHA LUIS EMMANUEL
- OSUNA BANDA ITZEL ARELY
- MIRANDA MOJICA ERICK
- BARBOSA PEÑA XAVIER MARISTIN

Parte 1:

Elabore la clase SocketMulticast para implementar los sockets de envío y recepción de mensajes multicast, su interfaz se muestra a continuación:

```
class SocketMulticast{
public:
SocketMulticast(int);
~SocketMulticast();
int recibe(PaqueteDatagrama & p);
int envia(PaqueteDatagrama & p, unsigned char ttl);
//Se une a un grupo multicast, recibe la IP multicast
void unirseGrupo(char *);
//Se sale de un grupo multicast, recibe la IP multicast
void salirseGrupo(char *);
private:
int s; //ID socket
};
```

En el método envia el segundo parámetro es el valor de TTL necesario para enviar el datagrama. En los métodos unirseGrupo para unirse al grupo de receptores y salirseGrupo para salir del grupo de receptores, el parámetro recibido es la dirección IP de multicast.

Algunos de los métodos de SocketMulticast implementados:

```
SocketMulticast::SocketMulticast(uint16_t iport, const std::string &addr){
    if ((s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
        throw "ERROR: Could not create socket " + std::string(strerror(errno));
    bzero((char *)&localAddress, sizeof(localAddress));
    localAddress.sin_family = AF_INET;
    localAddress.sin_addr.s_addr = inet_addr(addr.c_str());
    localAddress.sin_port = htons(iport);
    int reuse = 1;
    if (setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (const char*)&reuse, sizeof(reuse)) < 0)
        throw "ERROR: Could not set reuse port";
    if(addr != "0.0.0.0" || iport != 0)
        if (bind(s, (struct sockaddr *)&localAddress, sizeof(localAddress)) < 0)
            throw "ERROR: Could not bind: " + std::string(strerror(errno));
}

void SocketMulticast::joinGroup(char *IP){
    ip_mreq multicast;
    multicast.imr_multiaddr.s_addr = inet_addr(IP);
    multicast.imr_interface.s_addr = htonl(INADDR_ANY);
    setsockopt(s, IPPROTO_IP, IP_ADD_MEMBERSHIP, (void *)&multicast, sizeof(multicast));
}
```

```cpp
int SocketMulticast::send(DatagramPacket &p, unsigned char TTL){
    bzero((char *)&remoteAddress, sizeof(remoteAddress));
    remoteAddress.sin_family = PF_INET;
    remoteAddress.sin_addr.s_addr = inet_addr(p.getAddress().c_str());
    remoteAddress.sin_port = htons(p.getPort());
    if(setsockopt(s, IPPROTO_IP, IP_MULTICAST_TTL, (void *)&TTL, sizeof(TTL)) < 0)
        throw "ERROR: Could not set TTL paramater";
    return sendto(s, p.getData(), p.getLength(), 0, (struct sockaddr*)&remoteAddress, sizeof(remoteAddress));
}

int SocketMulticast::receive(DatagramPacket &p){
    //bzero((char *)&remoteAddress, sizeof(remoteAddress));//inicializa en null
    socklen_t len = sizeof(remoteAddress);
    int n = recvfrom(s, p.getData(), p.getLength(), 0, (struct sockaddr*)&remoteAddress, &len);
    //inet_ntop(AF_INET, &(remoteAddress.sin_addr.s_addr), (string(inet_ntoa(remoteAddress.sin_addr))).c_str()
    p.setPort(ntohs(remoteAddress.sin_port));
    p.setAddress(std::string(inet_ntoa(remoteAddress.sin_addr)));
    p.setLength(n);
    return n;
}
```

Para probar la clase SocketMulticast elabore un programa emisor que recibe como parámetros en línea de comandos la IP de multicast, el puerto, el valor de TTL y una cadena encerrada entre comillas. El receptor recibe como parámetros la IP de multicast y el puerto en el que escucha. Al ejecutarse ambos, el receptor debe imprimir la IP y puerto de quien le ha enviado el mensaje multicast, así como la cadena recibida.

Implementación del emisor:

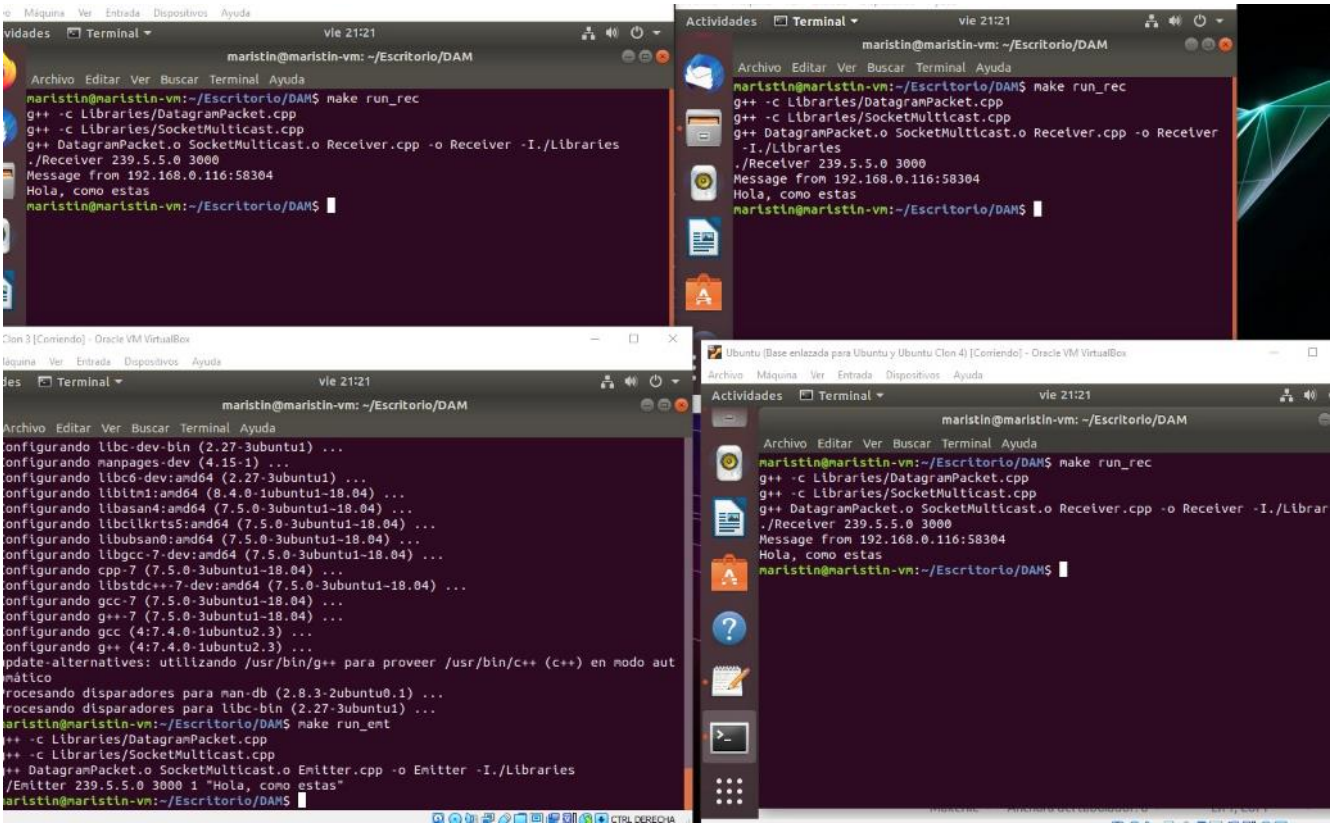
```cpp
#include "SocketMulticast.h"
#include <bits/stdc++.h>
using namespace std;

/*
    Programa Emisor
    Manda un mensaje a un grupo multicast
    Recibe:
    - IP multicast
    - Puerto
    - Parametro TTL
    - Mensaje
*/
int main(int argc, char* argv[]){
    try {
        string msg(argv[4]);
        SocketMulticast ms;
        DatagramPacket packet(argv[4], strlen(argv[4]) + 1, argv[1], atoi(argv[2]));
        ms.send(packet, atoi(argv[3]));
    } catch (string err){
        cout << err << endl;
    }
}
```

Implementación del receptor:

```
    Programa Receptor
    Manda un mensaje a un grupo multicast
    Recibe:
    - IP multicast
    - Puerto
 */
int main(int argc, char* argv[]) {
    try {
        char msg[100];
        SocketMulticast ms(atoi(argv[2]));
        ms.joinGroup(argv[1]);
        DatagramPacket packet(msg, sizeof(msg));
        //ms.receiveTimeout(packet, 3, 0);
        ms.receive(packet);
        cout << "Message from " << packet.getAddress() << ':' << packet.getPort() << endl;
        cout << msg << endl;
    }catch (string err) {
        cout << err << endl;
    }
}
```

Prueba:

Podemos observar como los receptores imprimen la IP y puerto de quien envió el mensaje y la cadena, la tercera terminal mostrada es el emisor. Estamos utilizando cuatro máquinas virtuales:



Parte 2:

En muchas ocasiones es necesario que dos procesos dentro de la misma computadora pertenezcan al mismo grupo de procesos multicast, lo cual implica que también ambos procesos se encuentren escuchando en el mismo puerto.
Para lograrlo solo es necesario añadir inmediatamente después de la función socket() en el constructor de la clase SocketMulticast lo siguiente:

```
int reuse = 1;
if (setsockopt(s, SOL_SOCKET, SO_REUSEPORT, &reuse, sizeof(reuse)) == -1) {
printf("Error al llamar a la función setsockopt\n");
exit(0);
}
```

Pruébelo con receptores en distintas computadoras cada una con dos receptores o más. El emisor deberá enviar un arreglo de dos enteros en un mensaje multicast, mientras que los receptores deberán imprimir los dos enteros recibidos, posteriormente deben sumar dichos números y regresar el resultado de la suma al emisor por un socket unicast.
Antes de que el emisor termine, deberá imprimir el resultado recibido por todas las computadoras, así como las IP´s de las computadoras que si contestaron. Es importante que el emisor ejecute el método recibeTimeout pues podría no recibir la respuesta de un receptor y quedarse bloqueado para siempre.

Implementación del método recibeTimeout:

```
int SocketMulticast::receiveTimeout(DatagramPacket &p, time_t seconds, suseconds_t microseconds){
    timeout.tv_sec = seconds;
    timeout.tv_usec = microseconds;
    setsockopt(s, SOL_SOCKET, SO_RCVTIMEO, (char *)&timeout, sizeof(timeout));

    socklen_t len = sizeof(remoteAddress);

    int n = recvfrom(s, p.getData(), p.getLength(), 0, (struct sockaddr*)&remoteAddress, &len);

    if (n < 0) {
        if (errno == EWOULDBLOCK)
            throw "ERROR: Timeout while receiving the packet";
        else
            throw "ERROR: While receiving the packet";
    }

    p.setPort(ntohs(remoteAddress.sin_port));
    p.setAddress(std::string(inet_ntoa(remoteAddress.sin_addr)));
    p.setLength(n);
    return n;
}
```

Implementación del emisor para este caso:

```
int main(int argc, char* argv[]){
    try {
        int num[1];
        int req[2];
        req[0] = atoi(argv[4]);
        req[1] = atoi(argv[5]);
        string msg(argv[4]);
        SocketMulticast ms;
        DatagramPacket packet((char*)req, 2 * sizeof(int), argv[1], atoi(argv[2]));
        ms.send(packet, atoi(argv[3]));
        DatagramPacket responsePacket((char*)num, sizeof(int));
        while(true){
            ms.receiveTimeout(responsePacket, 10, 0);
            cout << "Message from " << responsePacket.getAddress() << ":" << responsePacket.getPort() <<
            cout << "Result: " << (int)*responsePacket.getData() << endl;
        }
    } catch (string err){
        cout << err << endl;
    }
}
```

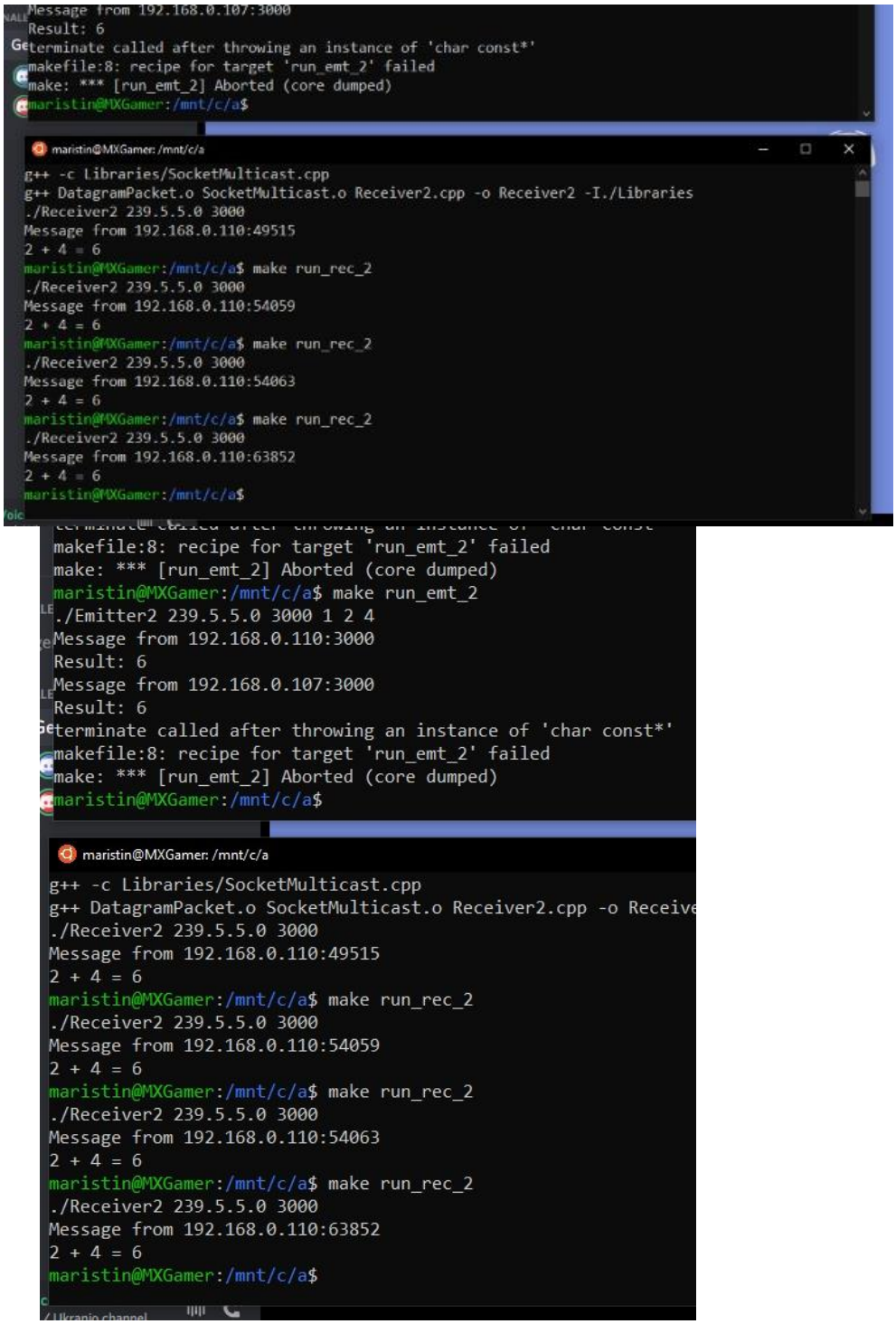
Implementación del receptor para este caso:

```
int main(int argc, char* argv[]) {
    try {
        int num[2], result;
        SocketMulticast ms(atoi(argv[2]));
        ms.joinGroup(argv[1]);
        DatagramPacket packet((char*)num, 2 * sizeof(int));
        //ms.receiveTimeout(packet, 3, 0);
        ms.receive(packet);
        result = num[0] + num[1];
        cout << "Message from " << packet.getAddress() << ':' << packet.getPort() << endl;
        cout << num[0] << " + " << num[1] << " = " << result << endl;
        /* Devolvemos el resultado */
        DatagramPacket resultPacket((char*)&result, sizeof(int), packet.getAddress(), packet.getPort())
        ms.send(resultPacket, 10);
    }catch (string err) {
        cout << err << endl;
    }
}
```

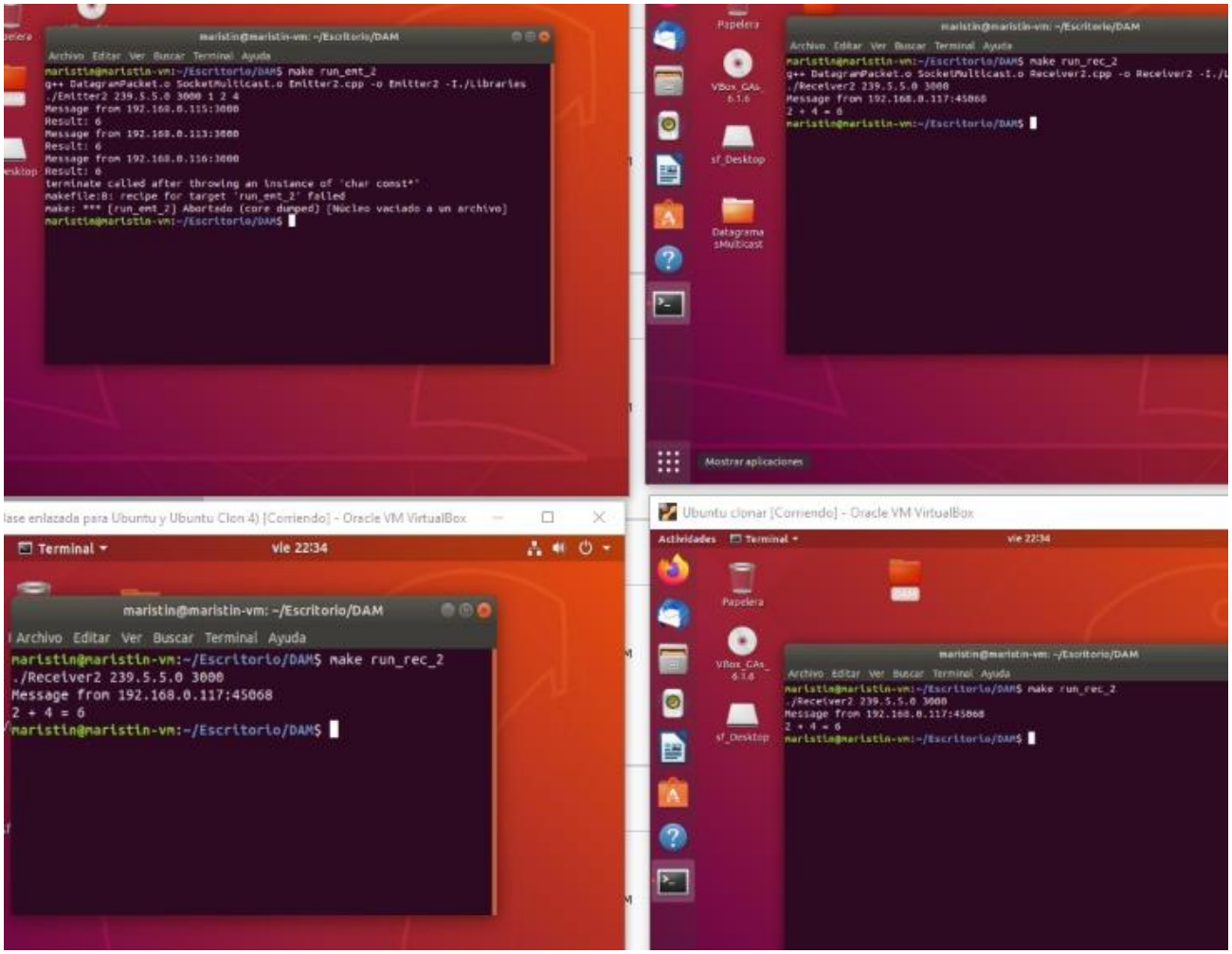
Prueba 1:

```
xavier@xmaristin:~$ cd da
xavier@xmaristin:~/da$ make run_re_2
make: *** No hay ninguna regla para construir el objetivo 'run_re_2'.  Alto.
xavier@xmaristin:~/da$ make run_rec_2
g++ -c Libraries/DatagramPacket.cpp
g++ -c Libraries/SocketMulticast.cpp
g++ DatagramPacket.o SocketMulticast.o Receiver2.cpp -o Receiver2 -I./Libra
./Receiver2 239.5.5.0 3000
Message from 192.168.0.110:63852
2 + 4 = 6
```

Podemos ver que se estan ejecutando 3 receptores distintos en 2 diferentes máquinas virtuales, además de un emisor. Efectivamente el receptor imprime de donde viene el mensaje y la suma, mientras que receptor imprime el resultado de la suma que se obtiene de cada receptor:



Prueba 2:


maristin@maristin-vm: ~/Escritorio/DAM
Archivo Editar Ver Buscar Terminal Ayuda
maristin@maristin-vm:~/Escritorio/DAM$ make run_ent_2
g++ DatagramPacket.o SocketMulticast.o Emitter2.cpp -o Emitter2 -I./Libraries
./Emitter2 239.5.5.0 3000 1 2 4
Message from 192.168.0.115:3000
Result: 6
Message from 192.168.0.113:3000
Result: 6
Message from 192.168.0.116:3000
Result: 0
terminate called after throwing an instance of 'char const*'
makefile:8: recipe for target 'run_ent_2' failed
make: *** [run_ent_2] Abortado (core dumped) [Núcleo vaciado a un archivo]
maristin@maristin-vm:~/Escritorio/DAM$
Papelera
VBox_GAs_6.1.6
sf_Desktop
Datagrama sMulticast
maristin@maristin-vm: ~/Escritorio/DAM
Archivo Editar Ver Buscar Terminal Ayuda
maristin@maristin-vm:~/Escritorio/DAM$ make run_rec_2
g++ DatagramPacket.o SocketMulticast.o Receiver2.cpp -o Receiver2 -I./
./Receiver2 239.5.5.0 3000
Message from 192.168.0.117:45068
2 + 4 = 6
maristin@maristin-vm:~/Escritorio/DAM$
Mostrar aplicaciones
lase enlazada para Ubuntu y Ubuntu Clon 4) [Corriendo] - Oracle VM VirtualBox
Terminal
vie 22:34
maristin@maristin-vm: ~/Escritorio/DAM
Archivo Editar Ver Buscar Terminal Ayuda
maristin@maristin-vm:~/Escritorio/DAM$ make run_rec_2
./Receiver2 239.5.5.0 3000
Message from 192.168.0.117:45068
2 + 4 = 6
maristin@maristin-vm:~/Escritorio/DAM$
Ubuntu clonar [Corriendo] - Oracle VM VirtualBox
Actividades
Terminal
vie 22:34
Papelera
VBox_GAs_6.1.6
sf_Desktop
maristin@maristin-vm: ~/Escritorio/DAM
Archivo Editar Ver Buscar Terminal Ayuda
maristin@maristin-vm:~/Escritorio/DAM$ make run_rec_2
./Receiver2 239.5.5.0 3000
Message from 192.168.0.117:45068
2 + 4 = 6
maristin@maristin-vm:~/Escritorio/DAM$