



INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

DESARROLLO DE SISTEMAS DISTRIBUIDOS

Multicast confiable

Grupo: 4CM1

Integrantes:

Angel Lopez Manriquez

Profesor:

Coronilla Contreras Ukranio

Fecha de realización: 26 de abril de 2020

Implementacion del protocolo multicast confiable simple en C++ sobre Linux

López Manríquez Ángel
4CM1

26 de abril de 2020

Índice

1. Parte 1	2
1.1. Envio confiable	2
1.2. Recepcion confiable	2
2. Parte 2	4
2.1. Tres receptores con 100 depositos	4
2.2. Tres receptores con 100000 depositos	5
2.3. No hay suficientes receptores	5

1. Parte 1

Implementacion de las funciones miembro que permiten enviar y recibir datagramas a traves de la multidifusion de manera confiable, como se explica en el libro, es un protocolo simple en donde el unico requisito es conocer el numero de receptores.

1.1. Envio confiable

```
11 int MulticastSocket::sendReliable(DatagramPacket &p, uint8_t ttl, int receptors) {
12     int n_tries = 7;
13     DatagramSocket::setTimeout(7, 0);
14     while (n_tries--) {
15         const int send_code = send(p, ttl);
16         if (send_code < 0) {
17             cerr << ("Cannot send");
18             exit(EXIT_FAILURE);
19         }
20         int successful_delivery = 0;
21         for (int i = receptors; i; --i) {
22             int response = 0;
23             DatagramPacket pack((char *) &response, sizeof(response));
24             int receive_code = DatagramSocket::receive(pack);
25             if (0 < receive_code && response == 1)
26                 ++successful_delivery;
27             else
28                 break;
29         }
30         if (successful_delivery == receptors)
31             return 1;
32     }
33     return -1;
34 }
```

La idea de la funcion es bastante parecida a la funcion miembro de los sockets UDP unicast, se envia el mensaje por multicast. Como tenemos el argumento del *numero de receptores* esperamos respuesta de este numero de receptores, si todos nos responden entonces el paquete ha sido enviado exitosamente a este numero de receptores, en caso contrario se reenvia el mismo paquete asi sea que algunos ya lo hayan recibido.

1.2. Recepcion confiable

```
36 int MulticastSocket::receiveReliable(DatagramPacket &p) {
37     long secs = 15, u_secs = 0;
38     DatagramSocket::setTimeout(secs, u_secs);
39     int receive_code = DatagramSocket::receive(p);
40     int send_code = 1;
```

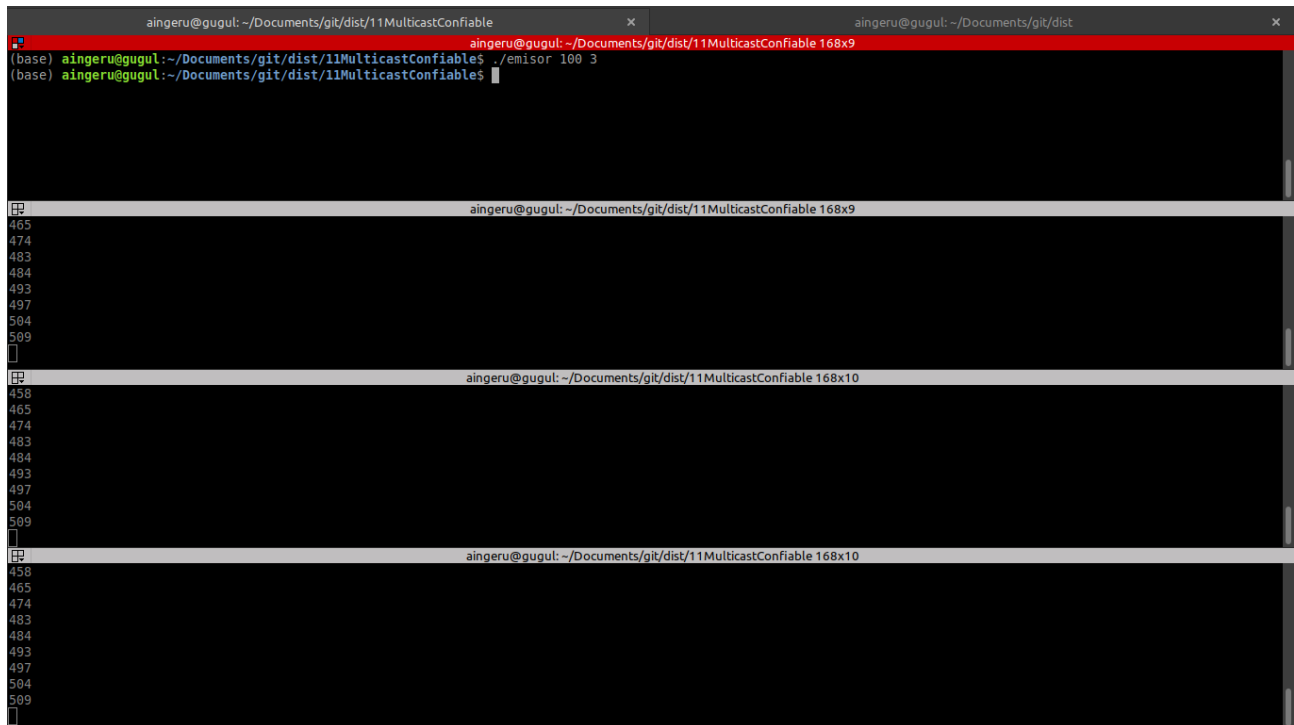
```
41     DatagramPacket reply_pack((char *) &send_code, sizeof(send_code),  
    ↪     p.getAddress(), p.getPort());  
42     DatagramSocket::send(reply_pack);  
43     return receive_code;  
44 }
```

Para el caso de la recepcion la implementacion es mas simple, solo se recibe el mensaje y le comunicamos al emisor que el paquete ha sido enviado exitosamente. Cabe destacar que no se hizo uso de el campo de acuse pues tenemos un entorno relativamente controlado entre las maquinas, pero para hacerlo mas realista se necesitarian de al menos conocer un conjunto de direcciones mac.

2. Parte 2

En la presente parte se resuelve un caso hipotetico de una persona que tiene su cuenta bancaria replicada en tres servidores. Claro esta que se interesa en conservar de manera identica el valor identico en todos los servidores. A continuacion se presentan capturas de pantalla del apropiado funcionamiento del protocolo confiable implementado. El uso de los temporizadores y las operaciones de idempotencia sobre mensajes hacen el adecuado funcionamiento del protocolo.

2.1. Tres receptores con 100 depositos



```
aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable
(base) aiingeru@gugul:~/Documents/git/dist/11MulticastConfiable$ ./emisor 100 3
(base) aiingeru@gugul:~/Documents/git/dist/11MulticastConfiable$

aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable 168x9
465
474
483
484
493
497
504
509

aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable 168x10
458
465
474
483
484
493
497
504
509

aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable 168x10
458
465
474
483
484
493
497
504
509
```

2.2. Tres receptores con 100000 depositos



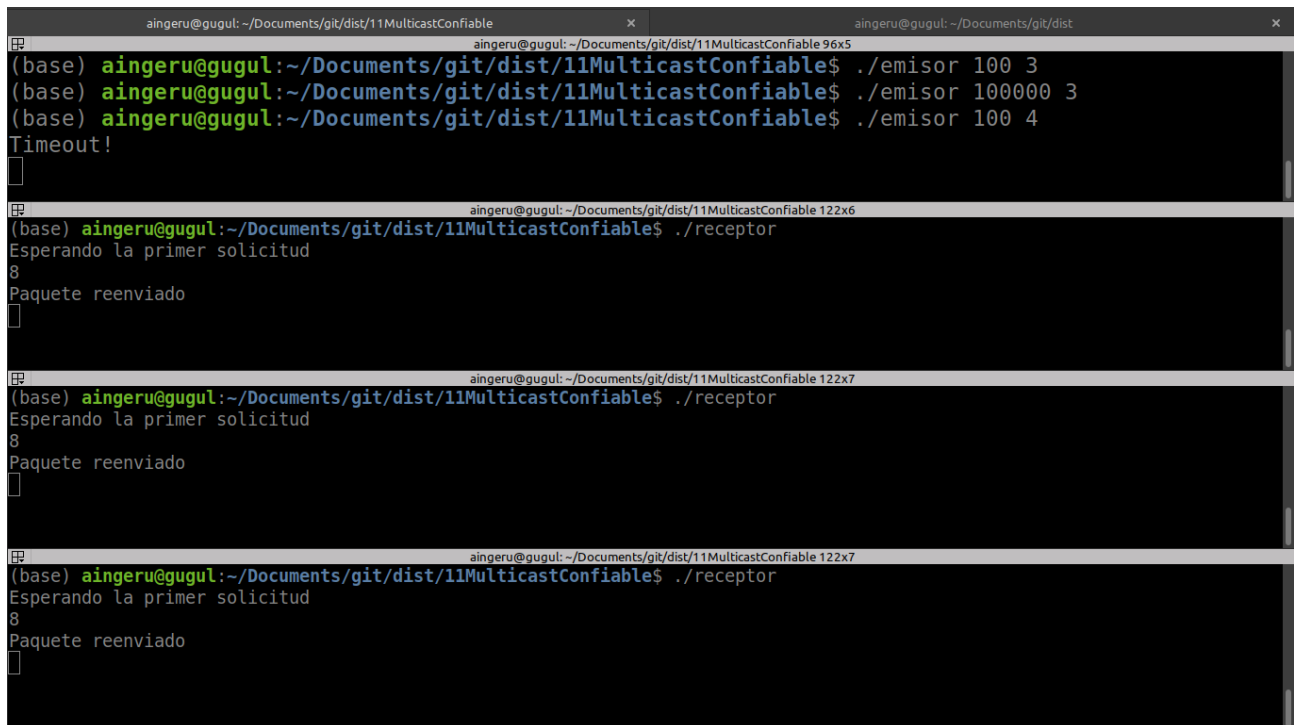
```
aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable x aiingeru@gugul: ~/Documents/git/dist x
aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable 96x5
(base) aiingeru@gugul:~/Documents/git/dist/11MulticastConfiable$ ./emisor 100 3
(base) aiingeru@gugul:~/Documents/git/dist/11MulticastConfiable$ ./emisor 100000 3
(base) aiingeru@gugul:~/Documents/git/dist/11MulticastConfiable$

aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable 168x9
497451
497454
497461
497467
497469
497477
497485
497492

aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable 168x10
497445
497451
497454
497461
497467
497469
497477
497485
497492

aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable 168x10
497445
497451
497454
497461
497467
497469
497477
497485
497492
```

2.3. No hay suficientes receptores



```
aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable x aiingeru@gugul: ~/Documents/git/dist x
aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable 96x5
(base) aiingeru@gugul:~/Documents/git/dist/11MulticastConfiable$ ./emisor 100 3
(base) aiingeru@gugul:~/Documents/git/dist/11MulticastConfiable$ ./emisor 100000 3
(base) aiingeru@gugul:~/Documents/git/dist/11MulticastConfiable$ ./emisor 100 4
Timeout!

aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable 122x6
(base) aiingeru@gugul:~/Documents/git/dist/11MulticastConfiable$ ./receptor
Esperando la primer solicitud
8
Paquete reenviado

aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable 122x7
(base) aiingeru@gugul:~/Documents/git/dist/11MulticastConfiable$ ./receptor
Esperando la primer solicitud
8
Paquete reenviado

aiingeru@gugul: ~/Documents/git/dist/11MulticastConfiable 122x7
(base) aiingeru@gugul:~/Documents/git/dist/11MulticastConfiable$ ./receptor
Esperando la primer solicitud
8
Paquete reenviado
```

En este caso se fuerza a tener menos receptores de los que supone el emisor, esto demuestra que en caso de no haber llevado con éxito el envío de paquetes a todos los receptores se reenvían los paquetes siete veces.