



Instituto Politécnico Nacional
Escuela Superior de Cómputo



ALUMNO:MERCADO ROGEL MARTÍN ISAURO

BOLETA:2014090449

UNIDAD DE APRENDIZAJE: APLICACIONES PARA LA COMUNICACIÓN
EN RED

GRUPO:
CURSO DE RECUPERACIÓN ACADÉMICA

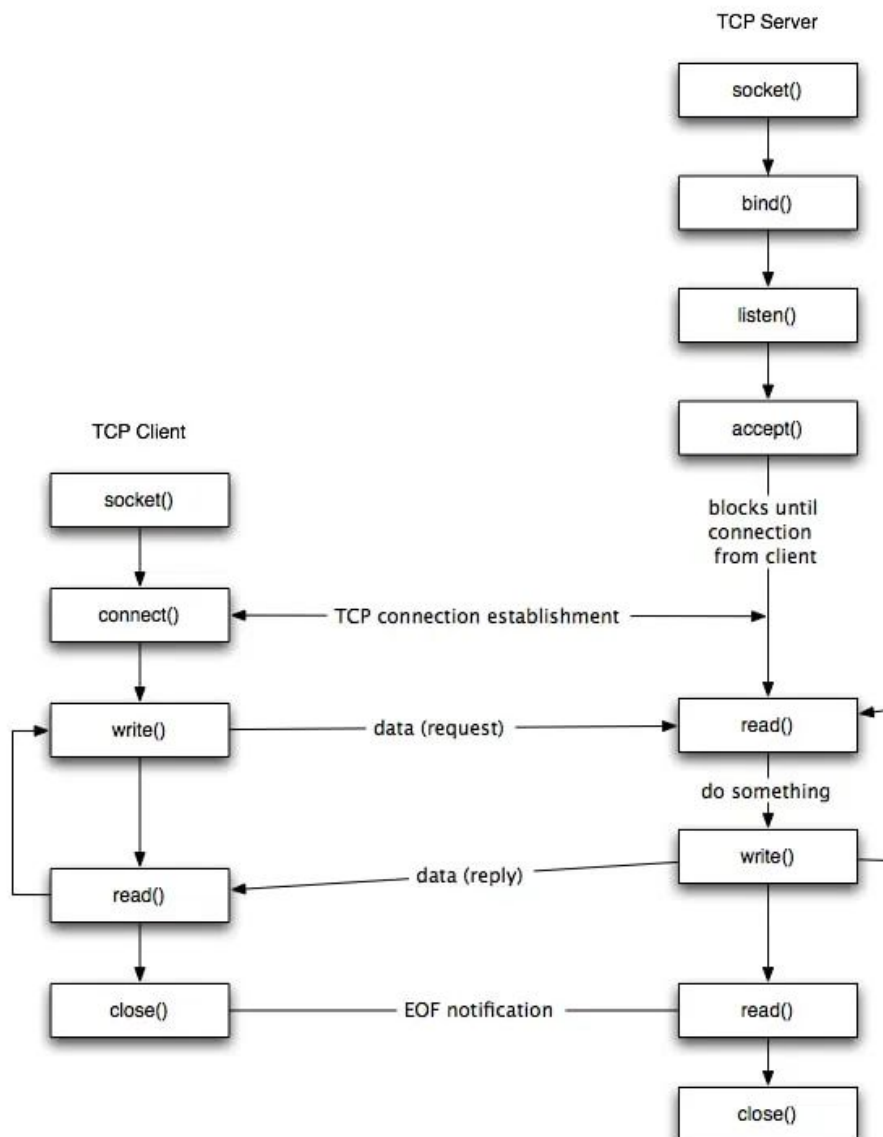
Tarea 2:

Programación de un cliente broadcast y servidor broadcast

Introducción

En el presente documento se mostrará como codificar un programa que actúe como un cliente broadcast en una red y a su vez un servidor que escuche estas peticiones.

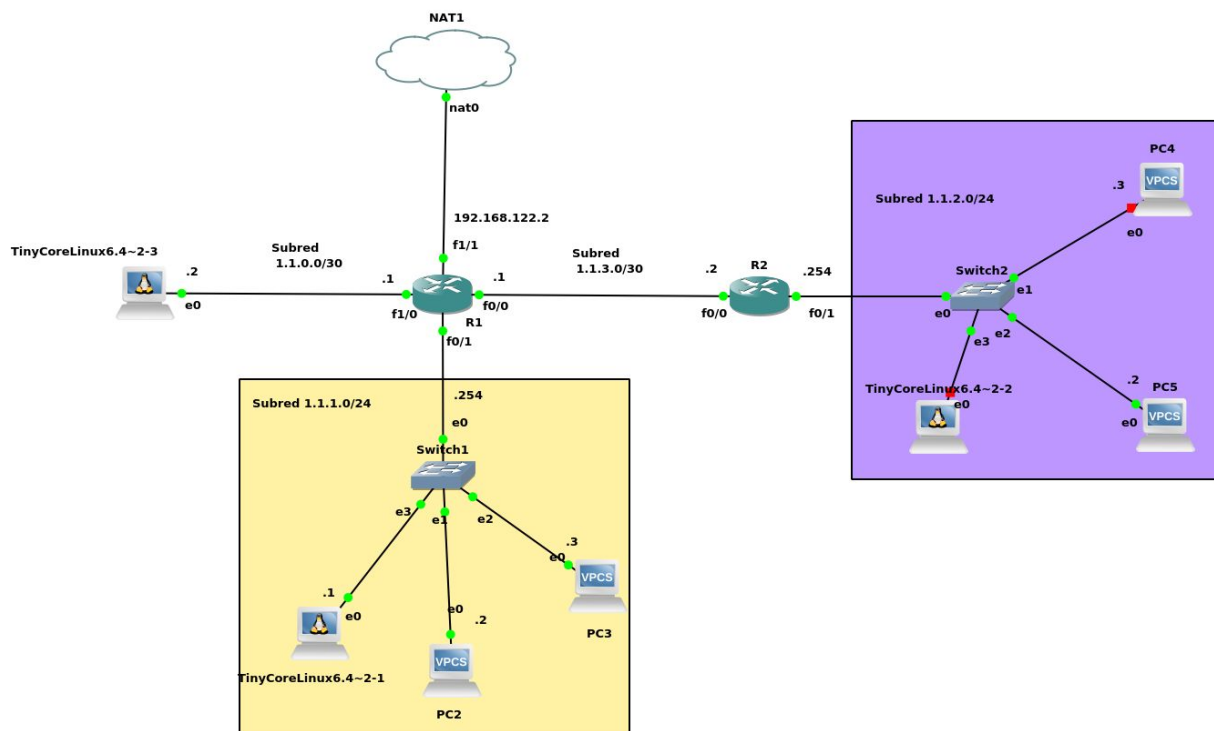
Para poder crear nuestro programa debemos entender como funcionan los sockets en el lenguaje C. La siguiente imagen lo resume perfectamente:



En este caso se trata de una comunicación TCP, proceso para UDP difiere sólo un poco y será aclarado más adelante. De igual forma en nuestro ejemplo usaremos `recv` y `send` sustituyendo a `read` y `write`.

Simple mensaje

Para poder generar nuestro programa broadcast entonces primero creemos una conexión simple donde el cliente envía un mensaje y el cliente recibe un mensaje. Consideremos la topología:



El programa que crearé tendrá como servidor a 1.1.0.2 y como cliente a 1.1.1.1. Así que en la siguiente captura se muestra el código y su ejecución:

```
}
int sock_des = socket(AF_INET,SOCK_STREAM,0); //Crear el socket
if(sock_des == -1)
{
    printf("No se pudo crear el socket ");
    return 0;
}
servidor.sin_family = AF_INET; //Protocolo
servidor.sin_port = htons(atoi(argv[2])); //Convertir el puerto
servidor.sin_addr.s_addr = inet_addr(argv[1]); //Convertir la cadena a un
bind(sock_des,(struct sockaddr *)&servidor,sizeof(servidor)); //Ligar pu

//Empezar a escuchar conexiones
listen(sock_des,2);

//Aceptar conexiones
int val = sizeof(cliente);
int accp_des = accept(sock_des,(struct sockaddr *)&cliente,&val);

//Fin configuracion

int tb = recv(accp_des,bufer,sizeof(bufer), 0); // CERO PARA QUE SEA BLO
bufer[tb] = '\0';

printf("El cliente envio %s\n",bufer);
strcpy(bufer,"Soy el servidor ");

tb = send(accp_des, bufer,strlen(bufer),0);
if(tb != strlen(bufer))
{
    return 0;
}
close(sock_des);
close(accp_des);
return 0;
}
serv.c 53/53 100%

struct sockaddr_in servidor,cliente;
if( argc != 3)
{
    printf("Se requiere puerto e ip\n");
    return 0;
}

int sock_fd = socket(AF_INET,SOCK_STREAM,0); //Crear el socket
if(sock_fd == -1)
{
    printf("No se pudo crear el socket ");
    return 0;
}

servidor.sin_family = AF_INET; //Protocolo
servidor.sin_port = htons(atoi(argv[2])); //Convertir el puerto
servidor.sin_addr.s_addr = inet_addr(argv[1]); //Convertir la cadena

if( connect(sock_fd, (struct sockaddr *) &servidor,sizeof(servidor))
{
    perror("Problema al conectarse al servidor");
    return -1;
}

strcpy(bufer,"Hola soy cliente");
int tb = send(sock_fd,bufer,strlen(bufer),0);
if(tb != strlen(bufer))
{
    printf("Error al enviar los datos al servidor");
    return 0;
}
tb = recv(sock_fd,bufer,sizeof(bufer),0);
bufer[tb] = '\0';
printf("El servidor envio: %s\n",bufer);
close(sock_fd);
return 0;
}
}
clien.c 48/48 100%
```

Servidor a la izquierda y cliente a la derecha, su ejecución:

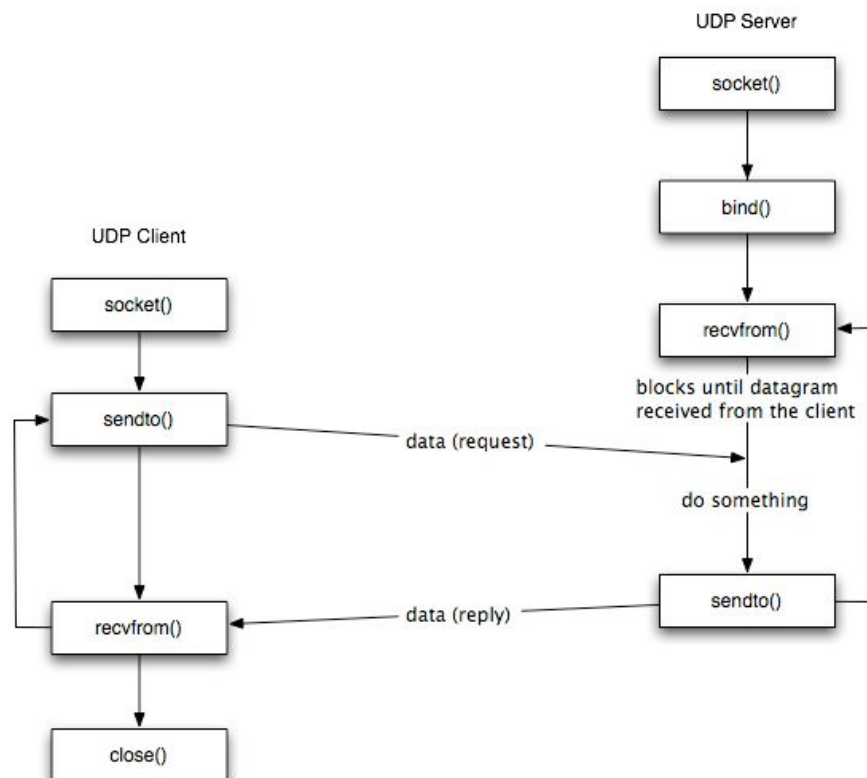
```
gns3@box:~/red$ vi serv.c
gns3@box:~/red$ ./serv 1.1.0.2 8000

El cliente envio Hola soy cliente
gns3@box:~/red$

El servidor envio: Soy el servidor
gns3@box:~/red$ vi clien.c
gns3@box:~/red$ vi clien.c
gns3@box:~/red$ ./clien 1.1.0.2 8000
El servidor envio: Soy el servidor
gns3@box:~/red$
```

UDP y broadcast

Ahora para poder programar nuestro mensaje broadcast revisemos como funcionan los sockets UDP:



El **Protocolo de datagramas de usuario (UDP)** es un protocolo simple de capa de transporte. La aplicación escribe un mensaje en un socket UDP, que luego se encapsula en un datagrama UDP, que luego se encapsula en un datagrama IP, que se envía al destino.

No hay garantía de que un UDP llegue al destino, que el orden de los datagramas se conservará en la red o que los datagramas lleguen solo una vez.

El problema de UDP es su falta de confiabilidad: si un datagrama llega a su destino final pero la suma de control detecta un error, o si el datagrama se cae en la red, no se retransmite automáticamente.

Cada datagrama UDP se caracteriza por una longitud. La longitud de un datagrama se pasa a la aplicación receptora junto con los datos.

No se establece conexión entre el cliente y el servidor y, por este motivo, decimos que UDP proporciona un servicio sin conexión.

La programación de nuestro cliente/servidor quedaría como sigue:

Servidor

```

}

/* Construcción de la estructura de direccion local */
addr_broadcast.sin_family = PF_INET; /* Berkley socket
addr_broadcast.sin_port = htons(atoi(argv[2])); /* Puerto
addr_broadcast.sin_addr.s_addr = inet_addr(argv[1]); /*

/*
status = bind(sockfd, (struct sockaddr *)&addr_broadcast);
if(status == -1)
{
    perror("Error al hacer bind \n");
    return -1;
}
*/

/* Establecer socket para permitir broadcast */
permiso = 1;
status = setsockopt(sockfd, SOL_SOCKET, SO_BROADCAST, (void *)&permiso, sizeof(permiso));
if(status == -1)
{
    perror("Error al establecer el socket \n");
    return -1;
}

printf("Introduce la cadena a enviar:");
fgets(buffer, MAX_BUF, stdin);
buflen = strlen(buffer); /* Tamaño de la cadena a enviar
status = sendto(sockfd, buffer, buflen, 0, (struct sockaddr *)&addr_broadcast, sizeof(addr_broadcast));
if(buflen != status)
{
    perror("Hubo un error al enviar los datos...\n");
    return -1;
}
}
SBroadcast.c 33/70 47%

```

Cliente1

```

/* Creación datagrama sockets udp */
if ((socket_fd = socket(PF_INET, SOCK_DGRAM, IPPROTO_UDP)) < 0)
{
    perror("Error al crear socket\n");
    return -1;
}

/* Asociar puerto a la dirección */
memset(&dir_brd, 0, sizeof(dir_brd)); /* Llenar de ceros estructura */
dir_brd.sin_family = AF_INET;
dir_brd.sin_addr.s_addr = htonl(INADDR_ANY); /* Recibir en cualquier interfaz
dir_brd.sin_port = htons(puerto_brd); /* Puerto broadcast */

/* Bind al puerto broadcast */
if (bind(socket_fd, (struct sockaddr *)&dir_brd, sizeof(dir_brd)) < 0)
{
    perror("Error al hacer bind\n");
    return -1;
}

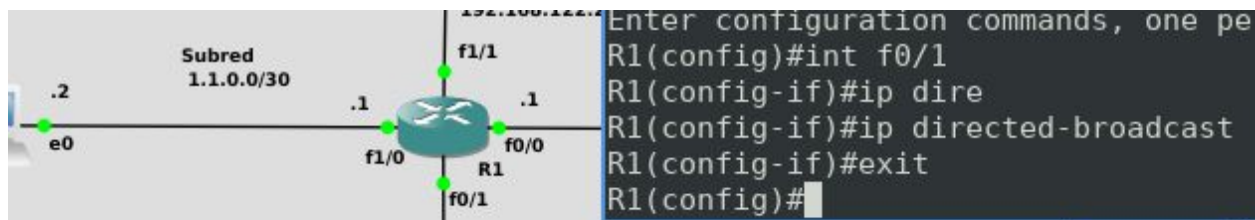
int permiso = 1;
int status = setsockopt(socket_fd, SOL_SOCKET, SO_BROADCAST, (void *)&permiso, sizeof(permiso));
if(status == -1)
{
    perror("Error al establecer permisos \n");
    return -1;
}

/* Recibir datagramas del servidor */
struct sockaddr_in dir_serv;
socklen_t tam_serv = sizeof(dir_serv);
int tam_recv = recvfrom(socket_fd, cad_recv, MAX_RECV, 0, (struct sockaddr *)&dir_serv, &tam_serv);
cad_recv[tam_recv] = '\0';
printf("El servidor envió: %s\n", cad_recv);
close(socket_fd);
CBroadcast.c 25/61 40%

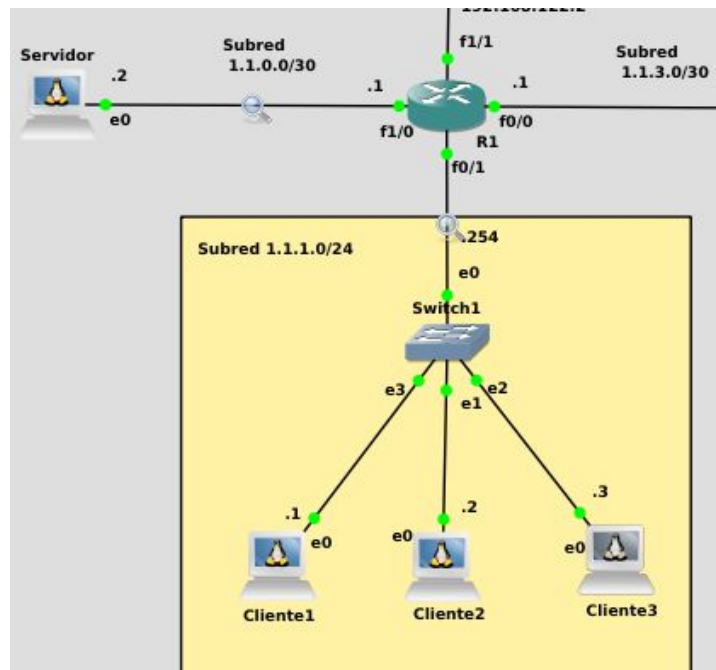
```

A la izquierda el servidor y a la derecha el cliente.

Antes de poder probar nuestro programa debemos de activar en nuestro router la opción para que pueda recibir peticiones broadcast:



Enviaré una petición de la subred 1.1.0.0 a la subred 1.1.1.0, veamos de nuevo las direcciones correspondientes:



Ahora sí probemos nuestro programa broadcast:

The screenshot shows a terminal window with three panes. The top pane is the server's terminal, and the bottom two panes are the clients' terminals. The server terminal shows the following commands and output:

```
gns3@box:~/red/broad$ ls
SBroadcast.c  cbroadcast
gns3@box:~/red/broad$ ./sbroadcast 1.1.1.255 8000
Introduce la cadena a enviar:Hola clientes
gns3@box:~/red/broad$
```

The client terminals show the following commands and output:

```
gns3@box:~/red/broad$ ls
CBroadcast.c  cbroadcast
gns3@box:~/red/broad$ ./cbroadcast
Uso: ./cbroadcast <Puerto de escucha>
gns3@box:~/red/broad$ ./cbroadcast 8000
Bind ok
Config ok
Antes
Despues
El servidor envio: Hola clientes
gns3@box:~/red/broad$
```

The client terminals also show the output of the 'cbroadcast' program, which is 'El servidor envio: Hola clientes'.

Conclusiones

En esta tarea primero investigué cómo enviar mensajes sobre una conexión TCP y lo probé con un sólo cliente y un servidor. Después tuve que buscar que era UDP ya que debía enviar mensajes no orientados a conexión, esto es, que no hubiera un cliente en específico sino todos los nodos de una red. Aprendí algunas cosas porque la tarea no quedó a la primera y tuve que indagar un poco más en el tema.