PRÁCTICA 4 – INTERACCIONES CON LA NIC

FECHA: 11/10/21

NOMBRE DEL EQUIPO: EL SIUU TEAM

PARTICIPANTES: -FISCHER SALAZAR CÉSAR EDUARDO
-LÓPEZ GARCÍA JOSÉ EDUARDO
-MEZA VARGAS BRANDON DAVID

UNIDAD ACADÉMICA: REDES DE COMPUTADORAS PANORÁMICA

Introducción de interacciones con la nic

Como bien sabemos, con un socket podemos hacer que dos programas puedan intercambiar cualquier flujo de datos, de manera fiable y ordenada.

Ahora bien, cuando hacemos uso de sockets crudos nos permiten accesos a los protocolos de comunicaciones, con la posibilidad de hacer uso o no de protocolos de capa 3 (nivel de red) y 4 (nivel de transporte), y por tanto dándonos el acceso a los protocolos directamente y a la información que recibe en ellos. El uso de sockets de este tipo nos va a permitir la implementación de nuevos protocolos, y de alguna forma, modificar los ya existentes, de eta forma podemos interactuar con la nic.

Para hacer esta interacción, debemos usar los sockets raw de la familia AF_PACKET ya que son los de más bajo nivel, permitiéndonos leer y escribir cabeceras de protocolos de cualquier capa.

De esta manera, para la práctica presente y el programa, usaremos sockets raw o sockets crudos para leer la información de nuestra nic.

OBJETIVOS

OBJETIVO PRINCIPAL: PROGRAMAR SOCKETS CRUDOS Y A TRAVEZ DE ELLOS Y CON EL MANEJO DE PETICIONES, IMPRIMIR EL NOMBRE DE LA TARJETA DE RED ATRAVEZ DEL INDICE, DIRECCION IP,LA MASCARA DE SUBRED, LA MAC, EL MTU Y LA DIRECCION DE BROAD CAST.

OBJETIVO SECUNDARIO. OBTENER DATOS DE NUESTRA TARJETA DE RED

ESCENARIO

LA FORMA MÁS BASICA PARA INTERACTUAR CON LA NIC ES HACIENDO USO DE SOCKETS CRUDOS PARA COMUNICARNOS CON LOS PROTOCOLOS DE COMUNICACIONES TENIENDO ACCESO A LA INFORMACIÓN DE ELLOS.

EN ESTA PRÁCTICA SE DEBERÁ REALIZAR UN PROGRAMA QUE HAGA USO DE SOCKETS CRUDOS PARA OBTENER INFORMACIÓN DE LA TARJETA DE RED COMO LA MAC, IP, MTU, DIRECCIÓN DE BROADCAST, NOMRE E ÍNDICE

RECURSOS NECESARIOS PARA REALIZAR LA PRÁCTICA

- Compilador Linux y editor de Linux
- Manuales de Linux

PARTE 1: DIAGRAMA DE FLUJO

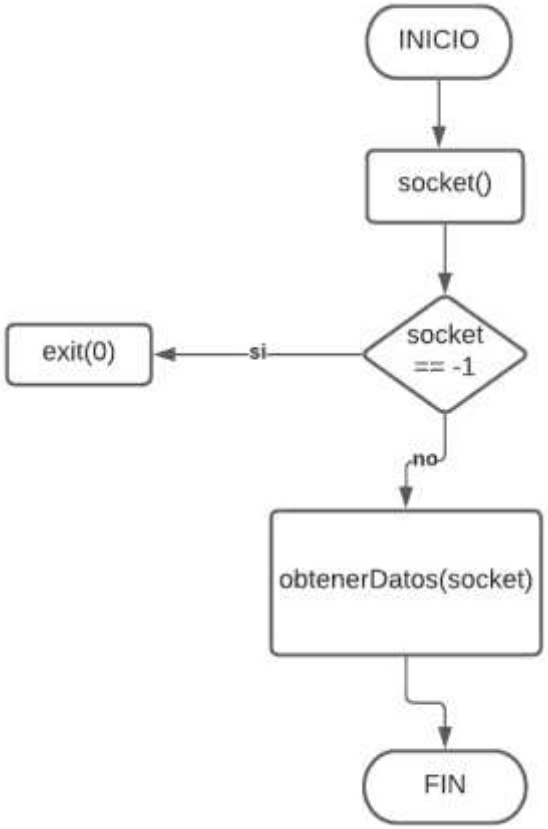

Figura 1. Diagrama de flujo del main

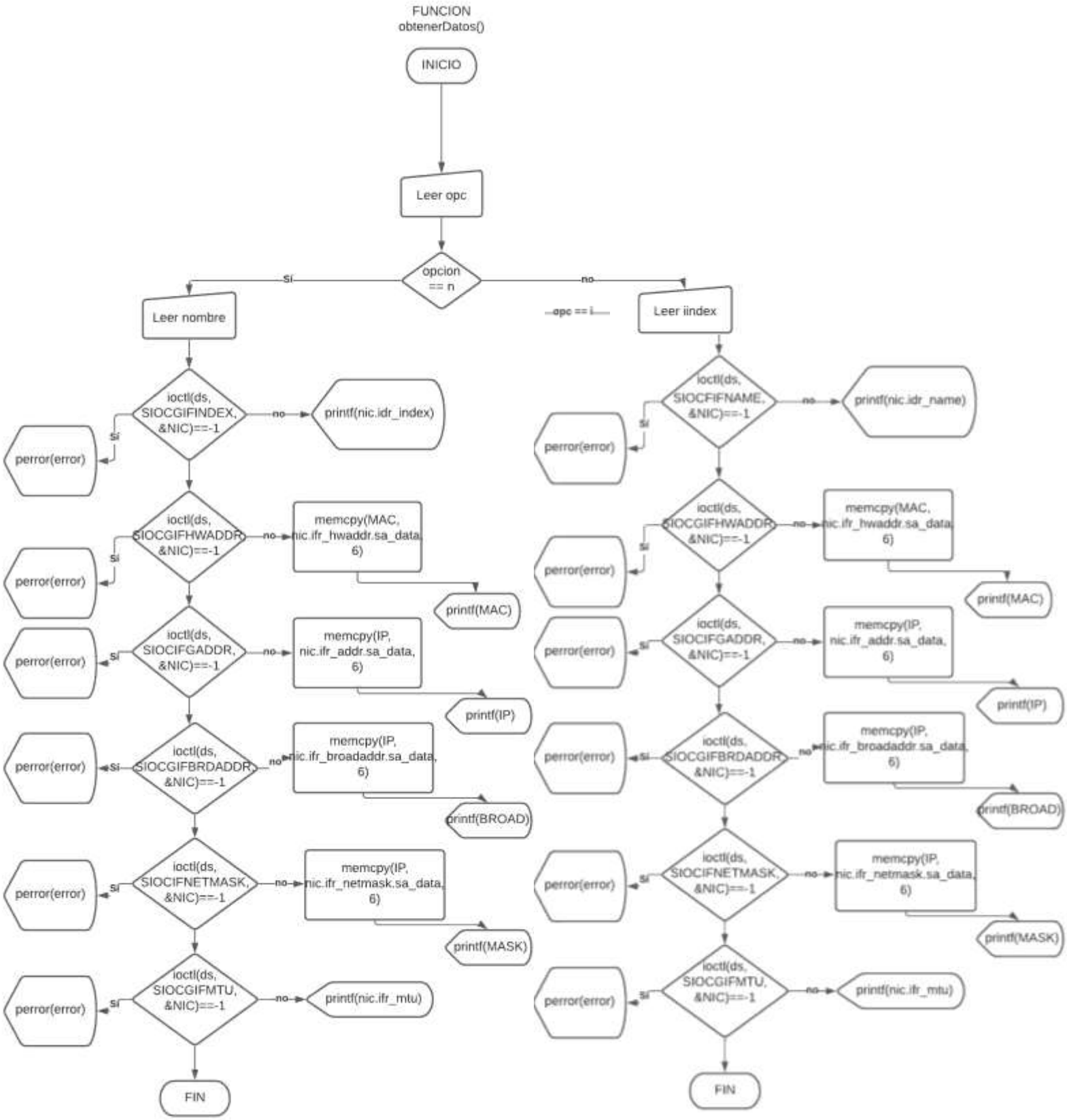

Figura 2. Diagrama de flujo de la función obtener datos.

PARTE 2: CÓDIGOS, COMANDOS Y EJECUCIÓN Y EXPLICACIÓN.

2.1 INCLUIR CODIGO EXPLICANDO LÍNEA POR LÍNEA, CAMBIAR NOMBRE DE SUS VARIABLES Y ESTRUCTURAS DE FORMA PERSONAL. RECUERDEN QUE LAS MEJORAS QUE LE HAGAN AL PROGRAMA VISTO EN CLASE AUMENTA SU CALIFICACION.

Se implementó una confición para dar a escoger al usuario como quiere obtener los datos, si a partir del índice de su interfaz de red o a partir del nombre, además de agregar la obtención de la dirección de broadcast y MTU, lo cual no fue indicado en clase, pero resultó útil para conocer más sobre lo que nos ofrecen los sockets crudos.

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/socket.h>
#include <linux/if_packet.h>
#include <net/ethernet.h> //L2 protocols
#include <arpa/inet.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <string.h>

void obtenerDatos(int ds);

/*
    En la compilacion se recomienda ejecutar en modo root
*/

unsigned char MAC[6], MASK[4], IP[4], BROAD[4]; //variables para obtener datos
int MTU;  //variable para obtener el MTU

void obtenerDatos(int ds)
{                   //ds: Descriptor de socket
    int i = 0;      //variable para loop
    char nombre[10]; //variable que guarda el nombre de la interfaz de red
    struct ifreq nic; //variable para usar la estructura ifreq
    int index;      //variable para guardar el index de la interfaz de red
    char opc;       //varibale para indicar como queremos obtener los datos n = a
                    // partir del nombre i = a partir del indice

    printf("\nPara Obtener datos a partir del nombre introduzca n");
    printf("\nPara Obtener datos a partir del indice introduzca i");
    printf("\nOpcion: ");
    scanf("%c", &opc); //leemos el indice

    if (opc == 'n') //si es n, obtenemos datos a partir del nombre
    {
```

```c
//leemos el nombre de la interfaz de red de la que queremos saber los datos
printf("\nIntroduce la interfaz de red: ");
scanf("%s", nombre);
strcpy(nic.ifr_name, nombre); //copiamos el nombre introducdio a la variable ifr_name: nombre de la interfaz



/*
    La función ioctl la usamos para manipular los valores de los paramatros de un socket.
    Proporcinna una iterfaz para controlar el comportamiento de dispositivos y de sus descriptores,
    en este caso de sockets
*/
// obtener el indice usando SIOCGIFINDEX
if (ioctl(ds, SIOCGIFINDEX, &nic) == -1)
    perror("Error al obtener el indice"); //si hay error manda este error
else
    printf("\nEl indice es: %d", nic.ifr_ifindex); //imprimimos el indice de la interfaz de red

//obtener la MAC usando SIOCGIFHWADDR
if (ioctl(ds, SIOCGIFHWADDR, &nic) == -1)
    perror("Error al obtener la MAC"); //si hay error manda este error
else
{
    //copiamos los 6 primeros datos de la estructura nic.ifr_hwaddr.sa_data
    //ya que este es el espacio correspondiente a la MAC
    memcpy(MAC, nic.ifr_hwaddr.sa_data, 6);
    printf("\nMAC: ");
    //recorremos nuestro arreglo de 6 posiciones para imrpimir la MAC
    for (i = 0; i < 6; i++)
        printf("%2x ", MAC[i]);
}

//obtener la IP usando SIOCGIFADDR
if (ioctl(ds, SIOCGIFADDR, &nic) == -1)
    perror("Error al obtener la IP"); //si hay error manda este error
else
{
    //copiamos los 6 primeros datos de la estructura nic.addr.sa_data
    memcpy(IP, nic.ifr_addr.sa_data, 6);
    printf("\nIP: ");
    //como trabajamos a nivel de bytes, para la ip los datos que imprimiremos seran del 2 al 6
    for (i = 2; i < 6; i++)
        printf("%2d.", IP[i]);
}

//obtener la direccion de broadcast usando SIOCGIFBRDADDR
if (ioctl(ds, SIOCGIFBRDADDR, &nic) == -1)
    perror("Error al obtener la direccion de broadcast"); //si hay error manda este error
else
{
    //copiamos los 6 primeros datos de la estructura nic.ifr_broadaddr.sa_data
    memcpy(BROAD, nic.ifr_broadaddr.sa_data, 6);
    printf("\nBroadcast: ");
    //como trabajamos a nivel de bytes, para la direccion de broadcast los datos que imprimiremos seran del 2 al
    for (i = 2; i < 6; i++)
        printf("%2d.", BROAD[i]);
}

//obtener la mascara de subred SIOCGIFNETMASK
if (ioctl(ds, SIOCGIFNETMASK, &nic) == -1)
    perror("Error al obtener la MAC"); //si hay error manda este error
else
{
    //copiamos los 6 primeros datos de la estructura nic.ifr_netmask.sa_data
    memcpy(MASK, nic.ifr_netmask.sa_data, 6);
    //imprimimos la mascara de subred con un for que recorre las posiciones, las imprimimos como decimales
    printf("\nMASK: ");
    for (i = 2; i < 6; i++)
        printf("%2d.", MASK[i]);
}

//obtener el MTU (Maximum Transfer UNit) usando SIOCGIFMTU
if (ioctl(ds, SIOCGIFMTU, &nic) == -1)
    perror("Error al obtener el MTU"); //si hay error manda este error
```

```
    else
    {
        printf("\nMTU: %d ", nic.ifr_mtu); //imprimimos el MTU de la interfaz de red
    }
}
else //si no obtenemos datos a partir del indice
{
    //Esta parte es para obtener los datos de la interfaz a partir de su indice
    printf("\nIntroduce el indice de la interfaz de red: ");
    scanf("%d", &index); //leemos el indice

    nic.ifr_ifindex = index; //establecemos el indice en la estructura

    //obtenemos el nombre de la interfaz de red con SIOCGIFNAME
    if (ioctl(ds, SIOCGIFNAME, &nic) == -1)
        perror("Error al obtener el nombre"); //si hay error manda este error
    else
        printf("\nEl nombre es: %s", nic.ifr_name); //imprimimos el indice de la interfaz de red

    //obtener la MAC usando SIOCGIFHWADDR
    if (ioctl(ds, SIOCGIFHWADDR, &nic) == -1)
        perror("Error al obtener la MAC"); //si hay error manda este error
    else
    {
        //copiamos los 6 primeros datos de la estructura nic.ifr_hwaddr.sa_data
        //ya que este es el espacio correspondiente a la MAC
        memcpy(MAC, nic.ifr_hwaddr.sa_data, 6);
        printf("\nMAC: ");
        //recorremos nuestro arreglo de 6 posiciones para imrpimir la MAC
        for (i = 0; i < 6; i++)
            printf("%2x:", MAC[i]);
    }

    //obtener la IP usando SIOCGIFADDR
    if (ioctl(ds, SIOCGIFADDR, &nic) == -1)
    perror("Error al obtener la IP"); //si hay error manda este error
else
{
    //copiamos los 6 primeros datos de la estructura nic.ifr_addr.sa_data
    memcpy(IP, nic.ifr_addr.sa_data, 6);
    printf("\nIP: ");
    //como trabajamos a nivel de bytes, para la ip los datos que imprimiremos seran del 2 al 6
    for (i = 2; i < 6; i++)
        printf("%2d.", IP[i]);
}

  //obtener la direccion de broadcast usando SIOCGIFBRDADDR
if (ioctl(ds, SIOCGIFBRDADDR, &nic) == -1)
    perror("Error al obtener la direccion de broadcast"); //si hay error manda este error
else
{
    //copiamos los 6 primeros datos de la estructura nic.ifr_broadaddr.sa_data
    memcpy(BROAD, nic.ifr_broadaddr.sa_data, 6);
    printf("\nBroadcast: ");
    //como trabajamos a nivel de bytes, para la direccion de broadcast los datos que imprimiremos seran del 2 al
    for (i = 2; i < 6; i++)
        printf("%2d.", BROAD[i]);
}

//obtener la mascara de subred SIOCGIFNETMASK
if (ioctl(ds, SIOCGIFNETMASK, &nic) == -1)
    perror("Error al obtener la MAC"); //si hay error manda este error
else
{
    //copiamos los 6 primeros datos de la estructura nic.ifr_netmask.sa_data
    memcpy(MASK, nic.ifr_netmask.sa_data, 6);
    //imprimimos la mascara de subred con un for que recorre las posiciones, las imprimimos como decimales
    printf("\nMASK: ");
    for (i = 2; i < 6; i++)
        printf("%2d.", MASK[i]);
}
```

```
        }

        //obtener el MTU (Maximum Transfer UNit) usando SIOCGIFNETMASK
        if (ioctl(ds, SIOCGIFMTU, &nic) == -1)
            perror("Error al obtener el MTU"); //si hay error manda este error
        else
        {
            printf("\nMTU: %d ", nic.ifr_mtu); //imprimimos el MTU de la interfaz de red
        }
    }
}

int main()
{
    int packet_socket; //variable para el socket

    /*
        primer parametro: familia a trabajar
        segundo parametro: SOCK_RAW: para trabajar en la capa de enlace de datos
        tercer parametro: colocamos todos los protocolos
        socket Devuelve un valor entero
    */
    packet_socket = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));

    if (packet_socket == -1)
    { //-1 si no se pudo abrir el socket
        perror("Error al abrir el socket");
    }
    else
    {
        perror("Exito al abrir el socket");
        obtenerDatos(packet_socket); //mandamos a llamar a al función que obtiene todos los datos
    }
    close(packet_socket); //cerramos el socket
    printf("\n");

    return 0;
}
```

Figura 3. Código del programa

2.2 CON LA AYUDA DEL COMANDO IFCONFIG OBTEN LOS DATOS DE TU NIC INCLUYE CAPTURA DE PANTALLA.

```
root@BDMV:/media/root/Mas espacio/Redes 1# ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe69:7a5f  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:69:7a:5f  txqueuelen 1000  (Ethernet)
        RX packets 15183  bytes 20321070 (19.3 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 6713  bytes 534260 (521.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 147  bytes 384723 (375.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 147  bytes 384723 (375.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Figura 4. Ejecución del comando ifconfig

Como podemos ver en la figura 4, tenemos dos interfaces de red, la loopback y la llamada enp0s3, las pruebas realizadas en el programa se realizaron sobre la llamada enp0s3

2.3 CON LA AYUDA DEL COMANDO IFCONFIG APAGA Y PRENDE TU TARJETA DE RED, INCLUYE IMÁGENES DE LA TARJETA ENCENDIDA Y APAGADA

Primeramente, apaguemos la tarjeta de red, ya que por defecto se encuentra prendida, esto lo hacemos con el comando ***ifconfig nombreInterfaz down*** como se ve en la figura 5.

```
root@BDMV:/media/root/Mas espacio/Redes 1# ifconfig enp0s3 down
root@BDMV:/media/root/Mas espacio/Redes 1# ifconfig
lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 679  bytes 428767 (418.7 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 679  bytes 428767 (418.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@BDMV:/media/root/Mas espacio/Redes 1#
```

Figura 5. Tarjeta de red enp0s3 apagada.

Como vemos, apagamos la tarjeta de red llamada enp0s3, para verificarlo volvemos ejecutar el comando ifconfig y ya no nos aparece la información de la tarjeta de red, de igual forma si queremos ingresar a internet no nos deja como se ve en la figura 6.

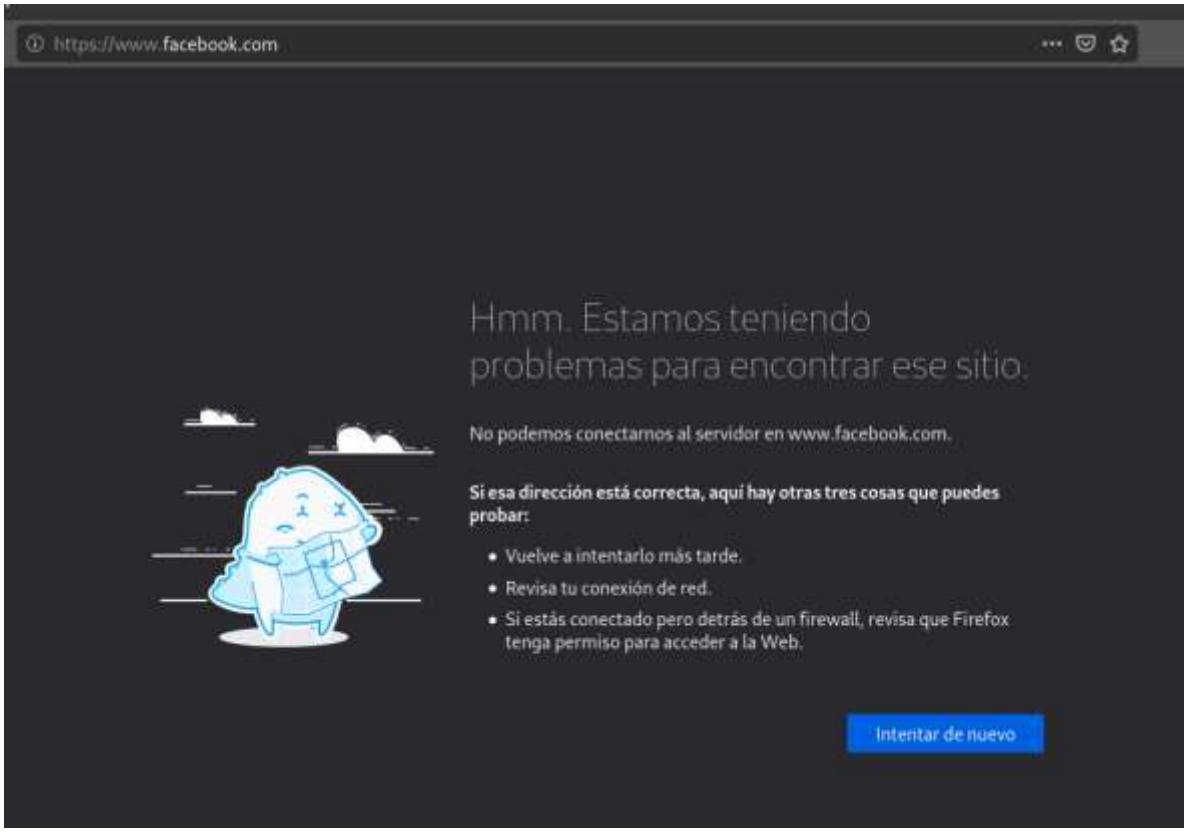

Figura 6. Sin acceso a internet por apagar la tarjeta de red

Ahora bien, para prender nuestra tarjeta de red usamos el comando ***ifconfig nombreInterfaz up*** como se ve en la figura 7.

```
root@BDMV:/media/root/Mas espacio/Redes 1# ifconfig enp0s3 up
root@BDMV:/media/root/Mas espacio/Redes 1# ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 10.0.2.15  netmask 255.255.255.0  broadcast 10.0.2.255
        inet6 fe80::a00:27ff:fe69:7a5f  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:69:7a:5f  txqueuelen 1000  (Ethernet)
        RX packets 15459  bytes 20413183 (19.4 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 7009  bytes 575056 (561.5 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 723  bytes 432179 (422.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 723  bytes 432179 (422.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

root@BDMV:/media/root/Mas espacio/Redes 1#
```

Figura 7. Tarjeta de red encendida

Como se ve, al prenderla con el comando anteriormente descrito, al ejecutar el comando ifconfig ya podemos ver de nuevo la información de nuestra tarjeta de red

2.4 INCLUYE LA CAPTURA DE PANTALLA DEL RESULTADO DE EJECUTAR EL PROGRAMA.

POR NOMBRE:

```
                    Interacciones con la NIC

Exito al abrir el socket: Success

Para Obtener datos a partir del nombre introduzca n
Para Obtener datos a partir del indice introduzca i
Opcion: n

Introduce la interfaz de red: enp0s3

El indice es: 2
MAC:  8: 0:27:69:7a:5f:
IP: 10: 0: 2:15:
Broadcast: 10: 0: 2:255:
MASK: 255:255:255: 0:
MTU: 1500
```

Figura 8. Resultado del programa por nombre

POR INDICE:

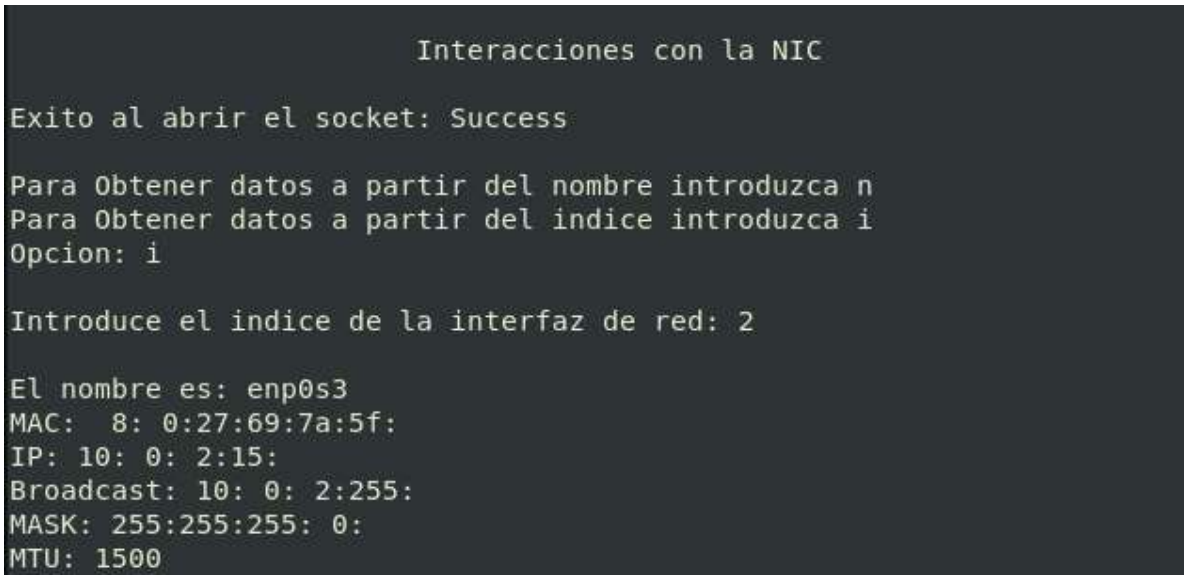

Figura 9. Resultado del programa por índice

Como vimos, se hace la ejecución del programa de las dos opciones, para obtener datos por medio del nombre o por medio del índice, arrojando los mismos datos que el comando ifconfig.

2.5 EXPLICA TEXTUALMENTE EL MANEJO DE CADA PETICION QUE USASTE PARA PODER IMPRIMIR EN TU PROGRAMA (IMPRIMIR EL NOMBRE DE LA ARJETA DE RED ATRAVEZ DEL INDICE, DIRECCION IP,LA MASCARA DE SUBRED, LA MAC, EL MTU Y LA DIRECCION DE BROAD CAST.

Para cada petición se hizo uso de la función ioctl, que nos sirve para manipular los valores de los parámetros, en este caso de un socket, proporciona una interfaz para controlar el comportamiento de dispositivos y de sus descriptores, como mencione, en este caso sockets. Mandando como parámetros el socket, el identificador y la estructura que en este caso será una estructura ifreq llamada nic

Primeramente, se estableció una estructura ifreq, que contiene todos los datos que queremos obtener de nuestra NIC.

Para obtener la información de nuestra NIC leemos el nombre de la interfaz de red y se almacena en ***nic.ifr_name,*** de esta forma indicamos de cual interfaz queremos la información.

- **INDEX:** para obtener el index, pasamos como parámetro el identificador **SIOCGIFINDEX** de esta forma obtenemos el índice.
- **MAC:** para la MAC usamos el identificador **SIOCGIFHWADDR** en la función **ioctl,** para terminar de obtener la MAC, copiamos el valor dentro de

nic.ifr_hwaddr.sa_data en una variable llamada MAC, la cual es un arreglo de 6 posiciones.

- **IP:** para la ip usamos el identificador **SIOCGIFADDR**, para terminar de obtener la IP, copiamos el valor de ***nic.ifr_addr.sa_data*** dentro de una variable llamada IP que es un arreglo
- **DIRECCIÓN DE BROADCAST:** para la dirección de broadcast usamos el identificador **SIOCGIFBRDADDR**, para terminar de obtener la dirección de broadcast, copiamos el valor de ***nic.ifr_broadaddr.sa_data*** dentro de una variable llamada BROAD que es un arreglo
- **MASCARA DE SUBRED:** para la máscara de subred usamos el identificador **SIOCGIFNETMASK**, para terminar de obtener la masara de subred, copiamos el valor de ***nic.ifr_netmask.sa_data*** dentro de una variable llamada MASK que es un arreglo
- **MTU:** para obtener el máximum transfer unit, hacemos uso del identificador **SIOCGIFMTU**, devolviéndonos el valor del MTU que se encuentra en ***nic.ifr_mtu***

Para obtener los datos a partir del índice, solo leemos el índice de la interfaz de red y la almacenamos en ***nic.ifindex*** de esta forma indicamos de que tarjeta de red obtener su información a partir del índice.

3. CONCLUSIONES INDIVIDUALES DE CADA PARTICIPANTE DEL EQUIPO

FISCHER SALAZAR CÉSAR EDUARDO

Esta práctica me parecido bastante interesante ya que desconocía completamente que se pudiera conocer información de nuestra nic mediante la implementación de un programa de socket crudo el cual nos brindaba dicha información mediante el nombre de nuestra tarjeta o el índice de estay que muy fácilmente pudiéramos corroborar mediante el comando ifconfig.

LÓPEZ GARCÍA JOSÉ EDUARDO

Por medio de la implementación de esta práctica se ha dado un entendimiento más amplio del empleo de sockets crudos, con los cuales se pudo tener interacción con los datos de nuestra tarjeta NIC, por medio de elementos primordiales dentro de la red, como es el nombre de la red, la IP, su submáscara y el broadcast que maneja. De igual forma, sirvió para comprender y tener en cuenta que el socket crudo es una herramienta esencial que permite la conexión a red e intercambio de datos.

MEZA VARGAS BRANDON DAVID

Con esta práctica hicimos uso de los sockets crudos, los cuales nos permitieron acceder a la información de nuestra nic gracias al programa implementado, personalmente no tenía idea de que esto se pudiera hacer, pero con esta práctica fui capaz de comprender al menos a grandes rasgos la utilidad que pueden tener los sockets crudos.

Me pareció muy buna la idea de ejecutar el comando ifconfig para compararlo con la salida de nuestro programa, siendo estas las mismas, de esta forma supimos que nuestro programa estaba correcto.

Una buena práctica para poner en marcha los conocimientos que hemos adquirido en clases de una forma interesante interactuando con nuestra NIC, pues no solo se pueden obtener y leer datos, si no que también podemos modificarlos, esto lo podemos ver en los manuales que Linux nos ofrece, en este caso el manual usado fue netdevice y el de función ioctl para comprender de mejor forma cómo funciona.