DISCLAIMER

MAC ADDRESS

A MAC (Media Access Control) address is a unique identifier assigned to a network interface card (NIC) at the hardware level. It is a 48-bit (6-byte) address that is globally unique and used for identifying devices on a local network.

The MAC address is stored in the network interface's hardware itself. It is typically programmed into the NIC during the manufacturing process and is usually stored in non-volatile memory, such as an EEPROM (Electrically Erasable Programmable Read-Only Memory), which retains the address even when the device is powered off.

The MAC address is associated with the specific network interface and remains unchanged unless it is modified through explicit configuration or MAC address spoofing. Every network interface, whether it's an Ethernet card, wireless adapter, or other types of network interfaces, has its own unique MAC address.

When devices communicate on a local network, they use MAC addresses to address and route network packets at the data link layer of the networking protocol stack. MAC addresses are used by Ethernet and Wi-Fi protocols to ensure that network packets are delivered to the correct device within the local network segment.

It's worth noting that the MAC address is separate from the IP address, which is used for network communication at the network layer (Layer 3) of the protocol stack. The MAC address is specific to the local network and is not routable over the internet, while IP addresses are used for global network communication.

In the **Linux network subsystem,** the **MAC** address for a net_device is typically obtained through the following steps and kernel APIs:

1. Initialization: During the initialization of a network device, the net_device structure is allocated and initialized. This structure contains a field called dev_addr, which represents the MAC address of the device.
2. Device driver configuration: The device driver for the network device is responsible for configuring the MAC address. It can set the MAC address explicitly or use default values.

3. Setting the MAC address: The device driver can set the MAC address using the **dev_set_mac_address()** function. This function is typically called during the initialization process or when the MAC address needs to be changed.
4. Persistent MAC address: If the network device has a persistent MAC address stored in non-volatile memory, the device driver can retrieve it and set it in the dev_addr field using the memcpy() function or similar methods.
5. Random MAC address generation: If a persistent MAC address is not available, the kernel can generate a random MAC address. The **eth_hw_addr_random()** function is used to generate a random MAC address and set it in the **dev_addr** field.
6. MAC address spoofing: In certain cases, the user or an application may want to spoof the MAC address of a network device. This can be achieved by using the **dev_set_mac_address()** function to set a custom MAC address in the dev_addr field.

To retrieve the MAC address from a net_device, you can use the following methods and kernel APIs:

1. Within the kernel: If you are writing kernel code, you can access the MAC address directly from the net_device structure. The dev_addr field contains the MAC address in binary form.
2. IOCTL: The SIOCGIFHWADDR ioctl command can be used to retrieve the MAC address of a network device from user space. This ioctl command is typically used with the ioctl() function to get the MAC address as a string.
3. Netlink sockets: Netlink sockets provide a communication channel between user space and the kernel. You can use the NETLINK_ROUTE family of netlink sockets and send a RTM_GETLINK message to retrieve the MAC address of a network device.

These are some of the common methods and kernel APIs used to obtain the MAC address of a net_device in the Linux network subsystem. The specific implementation details may vary depending on the kernel version and the device driver being used.

To read the MAC address from a NIC device in Linux, you can use the following kernel APIs and functions:

1. struct net_device: The net_device structure represents a network device and contains the MAC address information. It is defined in the linux/netdevice.h header file.
2. dev_get_by_name(): This function retrieves a net_device structure based on the device name. It takes the device name as a parameter and returns a pointer to the net_device structure if found, or NULL otherwise. It is defined in the linux/netdevice.h header file.
3. netdev_info() or netdev_info_once(): These functions are used to print information about a network device, including its MAC address. You can pass the net_device structure to these functions to print the MAC address to the system logs. They are defined in the linux/netdevice.h header file.

Here's an example code snippet that demonstrates how to read and print the MAC address of a NIC device:

Linux Kernel Module:

Example program :

```
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <linux/version.h>
#include <linux/netdevice.h>
#include <linux/kernel.h>

void read_mac_address(const char* device_name) {
    struct net_device* dev = dev_get_by_name(&init_net, device_name);

    if (dev) {
        netdev_info(dev, "MAC Address: %pM\n", dev->dev_addr);
        // Alternatively, you can use netdev_info_once() for one-time printing:
        // netdev_info_once(dev, "MAC Address: %pM\n", dev->dev_addr);
    } else {
        printk(KERN_ERR "Device '%s' not found\n", device_name);
    }
}


static int __init start(void)
{
        printk(KERN_DEBUG "Module to read MAC address \n");

        read_mac_address("wlp2s0");
        return 0;
}

static void __exit stop(void)
{
        printk(KERN_INFO "Good bye driver\n");
}


module_init(start);
module_exit(stop);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Sateesh Kumar G");


Compile and run :
$ qc read_MAC_address
Delete Makefile
Createing Makefile .....................
Make file created.......................
Calling make ...........................
make -C /lib/modules/5.15.87/build/ M=/home/skg/SSD240G/23-Linux/LDD/progs_ldd/01.Modules modules
make[1]: Entering directory '/media/skg/SATASSD240G/src/linux-5.15.87'
  CC [M]  /home/skg/SSD240G/23-Linux/LDD/progs_ldd/01.Modules/read_MAC_address.o
  MODPOST /home/skg/SSD240G/23-Linux/LDD/progs_ldd/01.Modules/Module.symvers
  CC [M]  /home/skg/SSD240G/23-Linux/LDD/progs_ldd/01.Modules/read_MAC_address.mod.o
  LD [M]  /home/skg/SSD240G/23-Linux/LDD/progs_ldd/01.Modules/read_MAC_address.ko
make[1]: Leaving directory '/media/skg/SATASSD240G/src/linux-5.15.87'
Now check .ko file
-------------------------------------------
$ sudo insmod read_MAC_address.ko
$ dmesg
Module to read MAC address
iwlwifi 0000:02:00.0 wlp2s0: MAC Address: b8:8a:60:b0:9f:14
```

In the above example, the read_mac_address() function takes the name of the network device as input. It uses the dev_get_by_name() function to retrieve the corresponding net_device structure. If the device is found, it prints the MAC address using the netdev_info() or netdev_info_once() functions, which format the MAC address as a string.

Please note that this code assumes that you are developing a kernel module or code that runs in the kernel context. If you are developing a user-space application, you would need to use different APIs, such as the ioctl() system call with the SIOCGIFHWADDR command, to read the MAC address.

Application Program: using ioctl call

Here's an example C program that can be used in user space to print the MAC address of a network interface:

Example program:

```
#include <stdio.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <net/if.h>
#include <stdlib.h>
#include <unistd.h>


void print_mac_address(const char* interface_name) {
    struct ifreq ifr;
    int sockfd;

    // Create a socket
    sockfd = socket(AF_INET, SOCK_DGRAM, 0);
    if (sockfd == -1) {
        perror("socket");
        return;
    }

    // Set interface name
    strncpy(ifr.ifr_name, interface_name, IFNAMSIZ - 1);

    // Get MAC address
    if (ioctl(sockfd, SIOCGIFHWADDR, &ifr) == -1) {
        perror("ioctl");
        close(sockfd);
        return;
    }

    // Print MAC address
    unsigned char* mac = (unsigned char*)ifr.ifr_hwaddr.sa_data;
    printf("MAC Address: %02X:%02X:%02X:%02X:%02X:%02X\n",
           mac[0], mac[1], mac[2], mac[3], mac[4], mac[5]);

    // Close socket
    close(sockfd);
}

int main() {
    const char* interface_name = "wlp2s0";  // Replace with your interface name
    print_mac_address(interface_name);

    return 0;
}
```

Compile and run :

```
$ gcc read_MAC_Application.c
$ ./a.out
MAC Address: B8:8A:60:B0:9F:14
$
```

In this program, the print_mac_address() function takes the name of the network interface as input. It creates a socket using socket(), sets the interface name in the ifr structure, and retrieves the MAC address using the ioctl() system call with the SIOCGIFHWADDR command.

The MAC address is then printed by accessing the ifr_hwaddr.sa_data field of the ifr structure. Finally, the socket is closed using close().

Note: Make sure to replace "eth0" with the name of the network interface you want to retrieve the MAC address from. You can find the interface name using tools like ifconfig or ip addr show.

Compile and run the program, and it should display the MAC address of the specified network interface.

Application Program: using netlink call

C program that uses Netlink to access the MAC address of a network interface in Linux:

Example program using Netlink :

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <linux/netlink.h>
#include <linux/rtnetlink.h>
#include <linux/if.h>

#define MAX_PAYLOAD 4096

struct nl_req {
    struct nlmsghdr nl_hdr;
    struct ifinfomsg if_info;
};

int main() {
    int sockfd;
    struct sockaddr_nl sa;
    struct nl_req req;
    struct msghdr msg;
    struct iovec iov;
    char buffer[MAX_PAYLOAD];
    ssize_t len;

    // Create netlink socket
    sockfd = socket(AF_NETLINK, SOCK_RAW, NETLINK_ROUTE);
    if (sockfd < 0) {
        perror("socket");
        exit(EXIT_FAILURE);
    }

    memset(&sa, 0, sizeof(sa));
    sa.nl_family = AF_NETLINK;
    sa.nl_groups = RTMGRP_LINK;

    // Bind the socket
```

```
    if (bind(sockfd, (struct sockaddr *)&sa, sizeof(sa)) < 0) {
        perror("bind");
        exit(EXIT_FAILURE);
    }

    memset(&req, 0, sizeof(req));
    req.nl_hdr.nlmsg_len = NLMSG_LENGTH(sizeof(struct ifinfomsg));
    req.nl_hdr.nlmsg_type = RTM_GETLINK;
    req.nl_hdr.nlmsg_flags = NLM_F_REQUEST | NLM_F_DUMP;
    req.if_info.ifi_family = AF_UNSPEC;

    iov.iov_base = &req;
    iov.iov_len = req.nl_hdr.nlmsg_len;

    memset(&msg, 0, sizeof(msg));
    msg.msg_iov = &iov;
    msg.msg_iovlen = 1;

    // Send the request
    if (sendmsg(sockfd, &msg, 0) < 0) {
        perror("sendmsg");
        exit(EXIT_FAILURE);
    }

    // Receive the response
    memset(buffer, 0, sizeof(buffer));
    iov.iov_base = buffer;
    iov.iov_len = sizeof(buffer);
    msg.msg_iov = &iov;
    msg.msg_iovlen = 1;

    len = recvmsg(sockfd, &msg, 0);
    if (len < 0) {
        perror("recvmsg");
        exit(EXIT_FAILURE);
    }

    // Parse the response
    struct nlmsghdr *nl_hdr;
       for (nl_hdr = (struct nlmsghdr *)buffer; NLMSG_OK(nl_hdr, len); nl_hdr =
NLMSG_NEXT(nl_hdr, len)) {
        if (nl_hdr->nlmsg_type == NLMSG_DONE) {
            break;
        }

        if (nl_hdr->nlmsg_type == RTM_NEWLINK) {
            struct ifinfomsg *if_info = (struct ifinfomsg *)NLMSG_DATA(nl_hdr);
            struct rtattr *attr = IFLA_RTA(if_info);
            int attr_len = nl_hdr->nlmsg_len - NLMSG_LENGTH(sizeof(struct ifinfomsg));

            while (RTA_OK(attr, attr_len)) {
                if (attr->rta_type == IFLA_ADDRESS) {
                    char mac_addr[IFHWADDRLEN];
                    memcpy(mac_addr, RTA_DATA(attr), IFHWADDRLEN);
                    printf("MAC Address: ");
                    for (int i = 0; i < IFHWADDRLEN; i++) {
                        printf("%02X:", mac_addr[i]);
```


Email: info@motionzen.com

```
                    }
                    printf("\n");
                }
                attr = RTA_NEXT(attr, attr_len);
            }
        }
    }

    close(sockfd);

    return 0;
}
```

Compile and run program:

```
$ gcc read_MAC_using_netlink.c
$ ./a.out
MAC Address: 00:00:00:00:00:00:
MAC Address: 48:FFFFFFBA:4E:FFFFFF8B:FFFFFFDC:48:
```

This program creates a Netlink socket, sends a request to retrieve the network interface information, and then parses the response to extract the MAC address. The MAC address is printed in the format XX:XX:XX:XX:XX:XX.

AUTHOR

SATEESH KUMAR G