BAN CƠ YẾU CHÍNH PHỦ

HỌC VIỆN KỸ THUẬT MẬT MÃ

PHÁT TRIỂN TRÌNH ĐIỀU KHIỂN THIẾT BỊ MẠNG

Ngành: Công nghệ thông tin

Chuyên ngành: Kỹ thuật phần mềm nhúng và di động

Mã số: 52.48.02.01

Nội dung

- Giao diện mạng
- Kết nối đến nhân HĐH
- Cấu trúc net_device
- Mở, đóng kết nối
- Gửi tin
- Nhận tin
- Bộ đệm socket
- Phân giải địa chỉ MAC

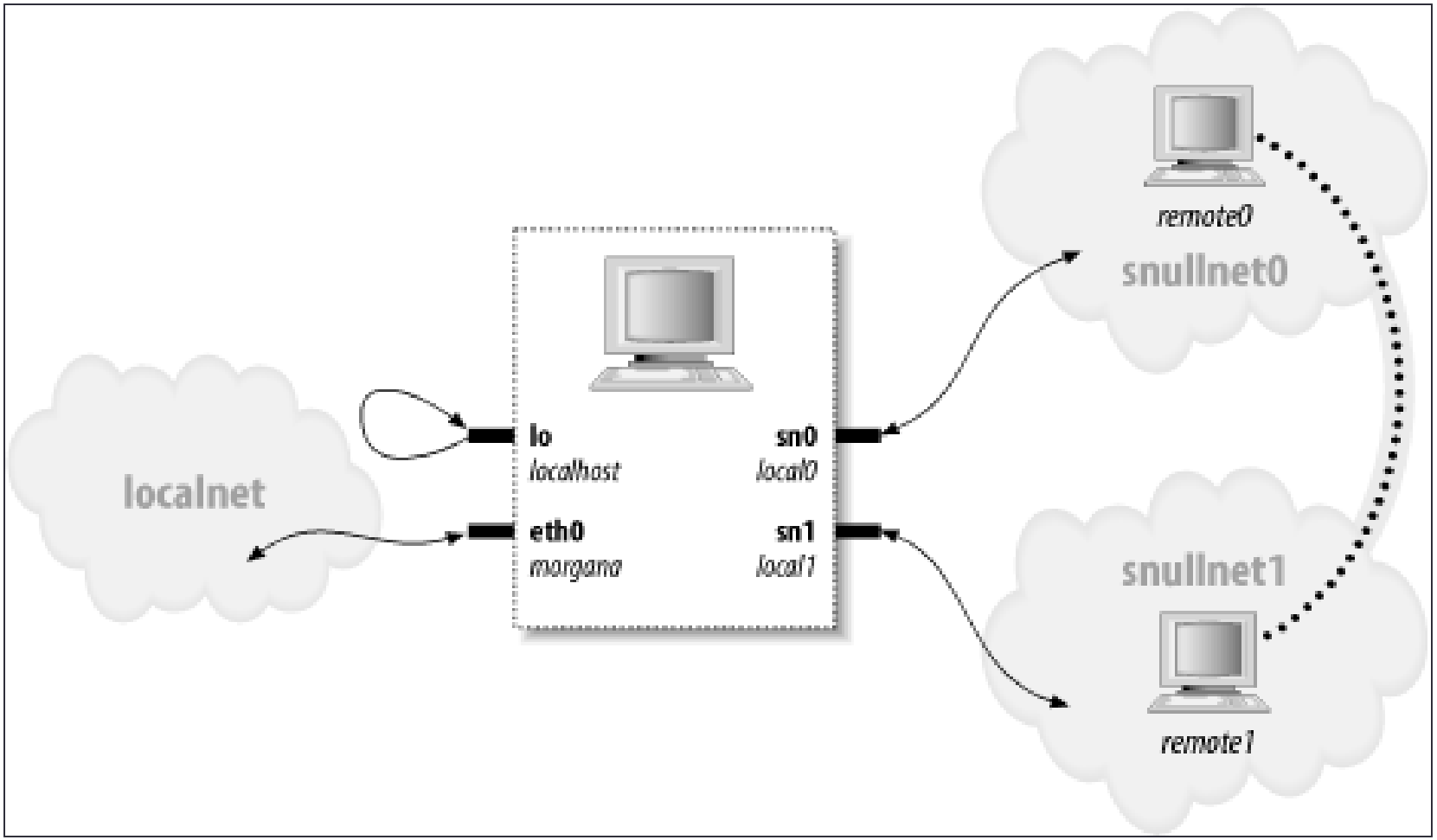
Giao diện mạng

- Vai trò của giao diện mạng trong hệ thống tương tự như thiết bị khối
- Một thiết bị khối đăng ký các đĩa và phương thức của nó với nhân, sau đó "truyền" và "nhận" các khối theo yêu cầu
- Tương tự, giao diện mạng phải tự đăng ký trong các cấu trúc dữ liệu nhân cụ thể để được gọi khi các gói tin được trao đổi với thế giới bên ngoài

Gán địa chỉ IP

- Để tránh phải làm việc với nhiều số, chúng ta gán các tên đến các số IP tương ứng, bao gồm:
  - snullnet0 là mạng kết nối với giao diện sn0 và snullnet1 là mạng kết nối với giao diện sn1. Địa chỉ của các mạng này chỉ khác nhau ở bit có trọng số thấp nhất của byte thứ 3. Các mạng này phải có 24 bit địa chỉ mặt nạ mạng.
  - local0 là địa chỉ IP gán cho giao diện sn0, địa chỉ này thuộc mạng snullnet0. Địa chỉ kết hợp với giao diện sn1 là local1. local0 và local1 phải khác nhau ở bit trọng số thấp nhất của byte thứ 3 và byte 4 (địa chỉ host).
  - remote0 là một host trong mạng snullnet0 và byte 4 của nó phải giống với byte 4 của local1. Bất kỳ gói tin nào gửi đến remote0 cũng đến local1 sau khi địa chỉ mạng đã được sửa đổi bởi mã giao diện. Host remote1 thuộc mạng snullnet1 và byte 4 của nó giống byte 4 của local0

Minh họa host và giao diện tương ứng



Minh họa host và giao diện tương ứng

- Ví dụ dưới đây là một số giá trị hợp lệ của số mạng. Sau khi đưa các dòng này vào **/etc/networks** ta có thể gọi các mạng bằng tên.
  - snullnet0 192.168.0.0
  - snullnet1 192.168.1.0
- Trong mỗi mạng trên, các số host hợp lệ được đặt trong **/etc/hosts** và được minh họa như sau:
  - 192.168.0.1 local0
  - 192.168.0.2 remote0
  - 192.168.1.2 local1
  - 192.168.1.1 remote1

Truyền tải dữ liệu vật lý của các gói tin

- Theo như truyền tải dữ liệu thì các giao diện snull thuộc về lớp Ethernet
- snull mô phỏng Ethernet vì phần lớn các mạng hiện có, ít nhất là các phân đoạn mà máy trạm kết nối đến, đều dựa trên công nghệ Ethernet
- Nhân cũng cung cấp một số thao tác hỗ trợ tổng quát cho các thiết bị Ethernet và không có lý do gì để không sử dụng nó
- Ưu điểm của việc trở thành một thiết bị Ethernet mạnh đến mức ngay cả giao diện plip

Kết nối đến nhân HĐH

- Đăng ký thiết bị
- Khởi tạo thiết bị
- Gỡ bỏ mô-đun

Đăng ký thiết bị

- Mỗi giao diện được mô tả bởi một mục cấu trúc net_device, được định nghĩa trong <linux / netdevice.h>. Trình điều khiển snull giữ con trỏ đến cấu trúc (cho sn0 và sn1) trong một mảng đơn giản:
  - struct net_device *snull_devs[2];
- Cấu trúc net_device, giống như nhiều cấu trúc hạt nhân khác, chứa một kobject và được tính số tham chiếu và xuất qua sysfs. Cũng như các cấu trúc khác, nó phải được cấp phát động.
- Hàm trong nhân được cung cấp để thực hiện việc cấp phát này là alloc_netdev:
  - struct net_device *alloc_netdev(int sizeof_priv,
  - const char *name,
  - void (*setup)(struct net_device *));

Đăng ký thiết bị

- Trong đó, sizeof_priv là kích thước vùng dữ liệu riêng của trình điều khiển
  - Với các thiết bị mạng, vùng này được cấp phát cùng với cấu trúc **net_device**
  - name là tên của giao diện mạng và setup là con trỏ đến hàm khởi tạo để thiết lập phần còn lạo của cấu trúc net_device
- Snull cấp phát hai cấu trúc thiết bị của nó như sau:

```
snull_devs[0] = alloc_netdev(sizeof(struct snull_priv),
"sn%d", snull_init);
snull_devs[1] = alloc_netdev(sizeof(struct snull_priv),
"sn%d", snull_init);
if (snull_devs[0] =  = NULL || snull_devs[1] =  = NULL)
    goto out;
```

Khởi tạo thiết bị

- Việc khởi tạo này phải được hoàn thành trước khi gọi register_netdev
- Cấu trúc net_device lớn và phức tạp nhưng nhân đã xử lý một số mặc định cho Ethernet thông qua hàm ether_setup (được gọi bởi alloc_etherdev)
- Vì snull sử dụng alloc_netdev, nên nó có hàm khởi tạo riêng.

Khởi tạo thiết bị

- Vì snull sử dụng alloc_netdev, nên nó có hàm khởi tạo riêng. Cốt lõi của hàm này (snull_init) được cài đặt như sau:

```
ether_setup(dev); /* assign some of the fields */
dev->open            = snull_open;
dev->stop            = snull_release;
dev->set_config      = snull_config;
dev->hard_start_xmit = snull_tx;
dev->do_ioctl        = snull_ioctl;
dev->get_stats       = snull_stats;
dev->rebuild_header  = snull_rebuild_header;
dev->hard_header     = snull_header;
dev->tx_timeout      = snull_tx_timeout;
dev->watchdog_timeo = timeout;
/* keep the default flags, just add NOARP */
dev->flags           |= IFF_NOARP;
dev->features        |= NETIF_F_NO_CSUM;
dev->hard_header_cache = NULL;      /* Disable caching */
```

Khởi tạo thiết bị

- Không giống như fops-> private_data, con trỏ priv được cấp phát cùng với cấu trúc net_device
- Truy cập trực tiếp vào priv cũng không được khuyến khích vì lý do hiệu suất và tính linh hoạt
- Khi một trình điều khiển cần có quyền truy cập vào con trỏ đến vùng dữ liệu riêng tư nên sử dụng hàm netdev_priv.
  - struct snull_priv *priv = netdev_priv(dev);

Khởi tạo thiết bị

- Mô-đun snull khai báo một cấu trúc dữ liệu snull_priv để sử dụng cho priv:

```
struct snull_priv {
    struct net_device_stats stats;
    int status;
    struct snull_packet *ppool;
    struct snull_packet *rx_queue;  /* List of incoming packets */
    int rx_int_enabled;
    int tx_packetlen;
    u8 *tx_packetdata;
    struct sk_buff *skb;
    spinlock_t lock;
};
```

Khởi tạo thiết bị

- Cấu trúc này chứa một thể hiện của struct net_device_stats – là nơi lưu trữ thông tin về giao diện
- Đoạn mã sau trong snull_init sử dụng để cấp phát và khởi tạo dev->priv:

```
priv = netdev_priv(dev);
memset(priv, 0, sizeof(struct snull_priv));
spin_lock_init(&priv->lock);
snull_rx_ints(dev, 1);
```

Gỡ bỏ mô-đun

- hủy đăng ký các giao diện,thực hiện các công việc dọn dẹp tài nguyên nội nếu yêu cầu và giải phóng cấu trúc net_device:

```
void snull_cleanup(void)
{
    int i;
    for (i = 0; i < 2;  i++) {
        if (snull_devs[i]) {
            unregister_netdev(snull_devs[i]);
            snull_teardown_pool(snull_devs[i]);
            free_netdev(snull_devs[i]); }
    }
    return;
}
```

Cấu trúc net_device

- Thông tin toàn cục
- Thông tin phần cứng
- Thông tin giao diện
- Các phương thức thao tác thiết bị
- Các trường tiện ích

Thông tin toàn cục

- char name[IFNAMSIZ];
  - Là tên của thiết bị. Nếu tên chứa định dạng %d, hàm register_netdev thay thế nó bằng một số để tạo thành tên duy nhất; số này bắt đầu từ 0.
- unsigned long state;
  - Chứa thông tin về trạng thái thiết bị, biểu diễn thông qua các cờ.
- struct net_device *next;
  - Con trỏ, trỏ đến thiết bị tiếp theo trong danh sách liên kết toàn cục. Trình điều khiển không thể truy xuất vào trường này.
- int (*init)(struct net_device *dev);
  - Con trỏ hàm khởi tạo. Nếu con trỏ này được thiết lập, nó sẽ được gọi bởi register_netdev để hoàn thành việc khởi tạo cấu trúc net_device

Thông tin phần cứng

- unsigned long rmem_end;
- unsigned long rmem_start;
- unsigned long mem_end;
- unsigned long mem_start;
  - Thông tin về bộ nhớ thiết bị. Các trường này giữ địa chỉ bắt đầu và kết thúc của vùng nhớ chia sẻ của thiết bị. mem sử dụng cho vùng nhớ truyền dữ liệu và rmem sử dụng cho vùng nhớ nhận dữ liệu. rmem vùng nhớ chỉ được tham chiếu bởi trình điều khiển. end - start là tổng dung lượng bộ nhớ có hiệu lực của thiết bị.
- unsigned long base_addr;
  - Địa chỉ vào/ra cơ sở của giao diện mạng.
- unsigned char irq;
  - Số hiệu ngắt được gán. Giá trị của dev->irq được in ra bởi ifconfig khi các giao diện mạng được liệt kê.
- unsigned char if_port;
  - Cổng đang sử dụng trong thiết bị nhiều cổng.
- unsigned char dma;
  - Kênh DMA được cấp phát bởi thiết bị

Thông tin giao diện

- void ltalk_setup(struct net_device *dev);
  - Thiết lập các trường cho thiết bị LocalTalk
- void fc_setup(struct net_device *dev);
  - Khởi tạo các trường cho thiết bị cáp quang.
- void fddi_setup(struct net_device *dev);
  - Cấu hình cho giao diện sử dụng cáp quang (FDDI).
- void hippi_setup(struct net_device *dev);
  - Chuẩn bị các trường cho giao diện song song hiệu năng cao (High-Performance Parallel Interface - HIPPI).
- void tr_setup(struct net_device *dev);
  - Xử lý cài đặt cho giao diện mạng vòng sử dụng thẻ bài (token ring).
- Các trường thông tin giao diện sau đây thường được gán bằng tay:
- unsigned short hard_header_len;
  - Độ dài tiêu đề phần cứng – là số byte trong gói tin được truyền đứng trước phần tiêu đề IP.

Thông tin giao diện

- unsigned mtu;
  - Đơn vị vận chuyển cực đại. Giá trị này có thể được thay đổi bằng ifconfig.
- unsigned long tx_queue_len;
  - Số frame tối đa trong hàng đợi truyền dữ liệu của thiết bị.
- unsigned short type;
  - Kiểu phần cứng của giao diện. Trường này được sử dụng bởi ARP để xác định loại địa chỉ phần cứng mà giao diện mạng hỗ trợ.
- unsigned char addr_len;
- unsigned char broadcast[MAX_ADDR_LEN];
- unsigned char dev_addr[MAX_ADDR_LEN];

Các phương thức thao tác thiết bị

- int (*open)(struct net_device *dev);
  - Sử dụng để mở một giao diện mạng.
- int (*stop)(struct net_device *dev);
  - Dừng hoạt động của giao diện mạng.
- int (*hard_start_xmit) (struct sk_buff *skb, struct net_device *dev);
  - Khởi tạo việc truyền gói tin.
- int (*hard_header) (struct sk_buff *skb, struct net_device *dev, unsigned short type, void *daddr, void *saddr, unsigned len);
  - Sử dụng để xây dựng tiêu đề phần cứng dựa trên các địa chỉ phần cứng đích và nguồn.

Mở, đóng kết nối

- Trình điều khiển phải thực hiện nhiều nhiệm vụ giống như trình điều khiển ký tự và khối
  - open yêu cầu tài nguyên hệ thống nào nó cần và yêu cầu giao diện xuất hiện; stop tắt giao diện và giải phóng tài nguyên hệ thống
  - Tuy nhiên, trình điều khiển mạng cũng phải thực hiện một số bước bổ sung vào thời gian mở
- Đầu tiên, địa chỉ phần cứng (MAC) cần được sao chép từ thiết bị phần cứng sang dev-> dev_addr trước khi giao diện có thể giao tiếp với bên ngoài
  - Địa chỉ phần cứng sau đó có thể được sao chép vào thiết bị tại thời điểm mở
  - Giao diện phần mềm snull gán nó từ bên trong open; nó chỉ giả mạo số phần cứng bằng chuỗi ASCII có độ dài ETH_ALEN - độ dài của địa chỉ phần cứng Ethernet

Mở, đóng kết nối

- Mã nguồn phương thức open cho snull như sau:

```
int snull_open(struct net_device *dev)
{
    /* request_region(  ), request_irq(  ), ....  (like fops->open) */

memcpy(dev->dev_addr, "\0SNUL0", ETH_ALEN);
    if (dev =  = snull_devs[1])
        dev->dev_addr[ETH_ALEN-1]++; /* \0SNUL1 */
    netif_start_queue(dev);
    return 0;
}
```

Mở, đóng kết nối

- Hàm stop thường được gọi là hàm close hoặc release được cài đặt như sau:

```
int snull_release(struct net_device *dev)
{
    /* release ports, irq and such -- like fops->close */

    netif_stop_queue(dev); /* can't transmit any more */
    return 0;
}
```

- Hàm void netif_stop_queue(struct net_device *dev) trái ngược với hàm netif_start_queue. Nó đánh dấu thiết bị là vô hiệu để truyền dữ liệu. Hàm này phải được gọi khi đóng giao diện

Gửi tin

- Truyền dẫn đề cập đến hành động gửi một gói qua một liên kết mạng
- Bất cứ khi nào nhân cần truyền gói dữ liệu, nó sẽ gọi phương thức hard_start_transmit của trình điều khiển để đưa dữ liệu vào hàng đợi đi
- Mỗi gói được xử lý bởi nhân sẽ được chứa trong cấu trúc bộ đệm socket (cấu trúc sk_buff – được định nghĩa trong <linux/skbuff.h>)
- Ngay cả khi giao diện không liên quan gì đến socket, mỗi gói mạng cũng thuộc về một socket trong các lớp mạng mức cao hơn; bộ đệm vào/ra của bất kỳ socket nào là danh sách các cấu trúc sk_buff
- Cấu trúc sk_buff tương tự được sử dụng để lưu trữ dữ liệu mạng trong tất cả các hệ thống con cho mạng Linux

Gửi tin

```
int snull_tx(struct sk_buff *skb, struct net_device *dev)
{
    int len;
    char *data, shortpkt[ETH_ZLEN];
    struct snull_priv *priv = netdev_priv(dev);
    data = skb->data;
    len = skb->len;
    if (len < ETH_ZLEN) {
        memset(shortpkt, 0, ETH_ZLEN);
        memcpy(shortpkt, skb->data, skb->len);
        len = ETH_ZLEN;
        data = shortpkt;
    }
    dev->trans_start = jiffies; /* save the timestamp */
```

Gửi tin

- Con trỏ đến cấu trúc sk_buff thường sử dụng là skb. Tiếp theo, ta sẽ phân tích con trỏ, cấu trúc này cả mô tả và mã nguồn.
  - Bộ đệm socket là một cấu trúc phức tạp và nhân cung cấp một số chức năng để thao tác trên nó.
    - ✓Bộ đệm socket được chuyển đến hard_start_xmit chứa gói vật lý sẽ xuất hiện trên phương tiện, hoàn thiện với các tiêu đề tương ứng với tầng truyền dẫn.
  - Giao diện không cần sửa đổi dữ liệu được truyền. skb->data trỏ đến gói được truyền và skb-> len là độ dài của nó tính theo byte

Gửi tin

- Đoạn mã truyền gói tin của snull

```
int snull_tx(struct sk_buff *skb, struct net_device *dev)
{
    int len;
    char *data, shortpkt[ETH_ZLEN];
    struct snull_priv *priv = netdev_priv(dev);
    data = skb->data;
    len = skb->len;
    if (len < ETH_ZLEN) {
        memset(shortpkt, 0, ETH_ZLEN);
        memcpy(shortpkt, skb->data, skb->len);
        len = ETH_ZLEN;
        data = shortpkt;
    }
    dev->trans_start = jiffies; /* save the timestamp */

    // Remember the skb, so we can free it at interrupt time
    priv->skb = skb;
    /* actual deliver of data is device-specific, and not
shown here */
    snull_hw_tx(data, len, dev);
    return 0; /* Our simple device can not fail */
}
```

Điều khiển nhận đồng thời

- Hàm hard_start_xmit được bảo vệ từ các lời gọi đồng thời bởi hàm spinlock(xmit_lock) trong cấu trúc net_device. Tuy nhiên, ngay sau khi hàm trả về, nó có thể được gọi lại
- Khi trình điều khiển của đã dừng hàng đợi, nó phải sắp xếp để khởi động lại hàng đợi tại một thời điểm nào đó trong tương lai, khi đó mới có thể chấp nhận các gói để truyền. Để thực hiện việc này, ta gọi hàm sau:

  void netif_wake_queue(struct net_device *dev);

- Hàm này giống như hàm netif_start_queue, ngoại trừ việc nó tác động đến hệ thống mạng để có thể đầu truyền lại các gói tin.

Điều khiển nhận đồng thời

- Nếu phải vô hiệu hóa truyền gói từ bất cứ nơi nào khác ngoài hàm hard_start_xmit (ví dụ đáp ứng yêu cầu cấu hình lại), có thể sử dụng hàm sau:
  - void netif_tx_disable(struct net_device *dev);
- Hàm này hoạt động giống như hàm netif_stop_queue nhưng nó đảm bảo khi trả về thì hàm hard_start_xmit đang không thực hiện trên một CPU khác và có thể khởi động lại hàng đợi với hàm netif_wake_queue

Timeout

- Nếu thời gian hệ thống hiện tại vượt quá thời gian lưu trong TRans_start ít nhất là khoảng thời gian chờ, tầng mạng sẽ gọi hàm tx_timeout của trình điều khiển.
  - Công việc của hàm này là đảm bảo hoàn thành đúng cách bất kỳ việc truyền nào đang diễn ra. Đặc biệt, điều quan trọng là trình điều khiển không bị mất dấu vết của bất kỳ bộ đệm socket nào tương ứng với mạng.
- snull có khả năng giả lập việc khóa phát tin và được điều khiển bởi hai tham số thời gian tải:
  - static int lockup = 0;
  - module_param(lockup, int, 0);
  - static int timeout = SNULL_TIMEOUT;
  - module_param(timeout, int, 0);

Timeout

- Nếu trình điều khiển được tải với tham số lockup=n, một khóa phát tin được giả lập ngay khi n gói tin được phát ra và trường watchdog_timeo được thiết lập với giá trị với một giá trị timeout. Khi giả lập khóa phát, snull cũng gọi hàm netif_stop_queue để tránh việc cố truyền phát tiếp
- Đoạn mã xử lý hết thời gian truyền của snull như sau:

```
void snull_tx_timeout (struct net_device *dev){
    struct snull_priv *priv = netdev_priv(dev);
    PDEBUG("Transmit timeout at %ld, latency %ld\n", jiffies,
            jiffies - dev->trans_start);
       /* Simulate a transmission interrupt to get things moving */
    priv->status = SNULL_TX_INTR;
    snull_interrupt(0, dev, NULL);
    priv->stats.tx_errors++;
    netif_wake_queue(dev);
    return;
```

Phân rã và hợp nhất

- Quá trình tạo một gói để truyền trên mạng bao gồm lắp ghép nhiều mảnh. Dữ liệu của gói thường được sao chép từ không gian người dùng và các tiêu đề được sử dụng bởi các tầng mạng khác nhau cũng phải được thêm vào.
  - Việc lắp ghép này có thể yêu cầu một số lượng lớn dữ liệu sao chép
- Nhân không chuyển các gói phân tán đến phương thức hard_start_xmit trừ khi bit NETIF_F_SG đã được thiết lập. Nếu đã đặt cờ đó, cần xem trường "shared info" trong skb để kiểm tra liệu gói được tạo thành từ một mảnh duy nhất hay nhiều và để tìm các mảnh phân tán nếu cần.
- Một macro đặc biệt sử dụng để truy cập thông tin này là skb_shinfo.

```
if (skb_shinfo(skb)->nr_frags == 0) {
    /* Just use skb->data and skb->len as usual */
}
```

Phân rã và hợp nhất

- Trường nr_frags cho biết có bao nhiêu đoạn đã được sử dụng để xây dựng gói.
  - Nếu giá trị của trường này bằng 0, gói tồn tại trong một mảnh duy nhất và có thể được truy cập thông qua trường dữ liệu như bình thường.
  - Tuy nhiên, nếu khác 0, trình điều khiển sắp xếp để chuyển từng đoạn riêng lẻ.
- Trường data của cấu trúc skb trỏ đến đoạn đầu tiên. Độ dài của đoạn phải được tính bằng cách trừ skb->len (chứa độ dài của gói đầy đủ) cho skb->data_len.
  - Các mảnh còn lại sẽ được tìm thấy trong một mảng frags; mỗi mục trong frags là một cấu trúc skb_frag_struct:

```
struct skb_frag_struct {
    struct page *page;
    _ _u16 page_offset;
    _ _u16 size;
};
```

Nhận tin

- Nhận dữ liệu từ mạng khó hơn truyền dữ liệu, bởi vì sk_buff phải được cấp phát và chuyển cho các lớp mạng bên trên.
  - Có hai chế độ nhận gói có thể được thực hiện bởi trình điều khiển mạng: điều khiển ngắt và thăm dò. Hầu hết các trình điều khiển thực hiện kỹ thuật điều khiển ngắt.
- Việc cài đặt snull tách biệt với chi tiết “phần cứng".
  - hàm snull_rx được gọi từ bộ xử lý "ngắt" snull sau khi phần cứng đã nhận được gói và chứa nó trong bộ nhớ của máy tính.
  - snull_rx nhận được một con trỏ tới dữ liệu và độ dài của gói tin. Nhiệm vụ duy nhất của nó là gửi gói tin và một số thông tin bổ sung cho các lớp mạng phía trên.
  - Mã nguồn xử lý này độc lập với cách thu được con trỏ dữ liệu và độ dài

Nhận tin

```
void snull_rx(struct net_device *dev, struct snull_packet *pkt)
{
    struct sk_buff *skb;
    struct snull_priv *priv = netdev_priv(dev);
    /** The packet has been retrieved from the transmission
     * medium. Build an skb around it, so upper layers can handle it*/
    skb = dev_alloc_skb(pkt->datalen + 2);
    if (!skb) {
        if (printk_ratelimit(  ))
            printk(KERN_NOTICE "snull rx: low on mem - packet
dropped\n");
        priv->stats.rx_dropped++;
        goto out;
    }
```

Nhận tin

```
memcpy(skb_put(skb, pkt->datalen), pkt->data, pkt->datalen);

    /* Write metadata, and then pass to the receive level */
    skb->dev = dev;
    skb->protocol = eth_type_trans(skb, dev);
    skb->ip_summed = CHECKSUM_UNNECESSARY; /* don't check it
*/
    priv->stats.rx_packets++;
    priv->stats.rx_bytes += pkt->datalen;
    netif_rx(skb);
  out:
    return;
}
```

Nhận tin

- Lớp mạng cần phải có thông tin trước khi nó có thể phân tích ngữ nghĩa của gói. Để kết thúc này, các trường dev và protocol phải được gán trước khi bộ đệm được chuyển lên tầng trên.
- Đoạn mã hỗ trợ Ethernet xuất một hàm trợ giúp là eth_type_trans; hàm này tìm giá trị phù hợp để đưa vào protocol. Sau đó, chúng ta cần xác định cách thức kiểm tra được thực hiện hoặc đã được thực hiện trên gói (snull không cần thực hiện bất kỳ tổng kiểm tra nào). Các chính sách cho skb-> ip_summed là:
  - CHECKSUM_HW
  - Thiết bị đã thực hiện tổng kiểm tra trên phần cứng.
  - CHECKSUM_NONE
  - Tổng kiểm tra chưa được thực hiện và phải được thực hiện bằng phần mềm.
  - CHECKSUM_UNNECESSARY
  - Không thực hiện kiểm tra tổng thể

Bộ đệm Socket

- Sk_buff là cấu trúc cốt lõi của hệ thống con xử lý mạng trong nhân Linux.
- Chúng ta sẽ không mô tả toàn bộ cấu trúc mà chỉ tập trung vào các trường có thể được sử dụng từ bên trong trình điều khiển.
- Nếu muốn tìm hiểu thêm, ta có thể xem trong <linux / skbuff.h> - nơi cấu trúc được định nghĩa và các hàm được xây dựng nguyên mẫu

Các trường quan trọng trong bộ đệm Socket

- struct net_device *dev;
  - Con trỏ đến thiết bị gửi hoặc nhận bộ đệm.
- union { /* ... */ } h;
- union { /* ... */ } nh;
- union { /*... */} mac;
  - Các con trỏ, trỏ đến các tiêu đề trong gói tin theo các mức khác nhau. Trong đó, h là tiêu đề tầng giao vận, nh tương ứng với tiêu đề tầng mạng và mac tương ứng với tầng liên kết dữ liệu.
  - Nếu trình điều khiển cần truy xuất địa chỉ nguồn và địa chỉ đích của gói tin TCP, có thể sử dụng skb->h.th.
- unsigned char *head;
- unsigned char *data;
- unsigned char *tail;
- unsigned char *end;
  - Các con trỏ được sử dụng để xác định dữ liệu trong gói tin. head trỏ đến địa chỉ bắt đầu của không gian bộ nhớ đã cấp phát, data là địa chỉ bắt đầu của các byte dữ liệu hợp lệ, tail là địa chỉ cuối của byte dữ liệu hợp lệ và end trỏ đến địa chỉ lớn nhất mà tail có thể đạt được. Theo đó, ta có thể tính kích thước bộ đệm là skb->end - skb->head và kích thước dữ liệu sử dụng hiện tại là skb->tail - skb->data.

Các hàm quan trọng trong bộ đệm Socket

- unsigned int len;
- unsigned int data_len;
  - len là kích thước đầy đủ của dữ liệu trong gói tin và data_len là độ dài phần dữ liệu của gói tin được lưu trong các đoạn (fragment) riêng biệt. Trường data_len bằng 0 nếu phân rã/hợp nhất không được sử dụng.
- unsigned char ip_summed;
  - Chính sách kiểm tra tổng thể cho gói tin.
- unsigned char pkt_type;
  - Kiểu gói tin được sử dụng khi chuyển giao. Trường này được thiết lập nó theo các giá trị: PACKET_HOST (gói tin cho host), PACKET_OTHERHOST (gói tin không cho host), PACKET_BROADCAST, or PACKET_MULTICAST. Trình điều khiển Ethernet không sửa đổi pkt_type tường minh mà hàm eth_type_trans sẽ thực hiện.
- shinfo(struct sk_buff *skb);
- unsigned int shinfo(skb)->nr_frags;
- skb_frag_t shinfo(skb)->frags;
  - Đây là các trường lưu thông tin chia sẻ của skb. Vì lý do hiệu năng, thông tin chia sẻ skb được lưu trữ trong các cấu trúc khác nhau, xuất hiện ngay sau skb trong bộ nhớ

Các hàm quan trọng trong bộ đệm Socket

- struct sk_buff *alloc_skb(unsigned int len, int priority);
- struct sk_buff *dev_alloc_skb(unsigned int len);
  - Cấp phát một bộ đệm. Hàm alloc_skb cấp phát một bộ đệm và khởi tạo cả skb->data, skb->tail và skb->head. Hàm dev_alloc_skb là một shortcut – nó gọi hàm alloc_skb với chính sách GFP_ATOMIC và Đặt trước không gian giữa skb->head và skb->data. Không gian dữ liệu này được sử dụng tối ưu trong tầng mạng và trình điều khiển không nên can thiệp.
- void kfree_skb(struct sk_buff *skb);
- void dev_kfree_skb(struct sk_buff *skb);
- void dev_kfree_skb_irq(struct sk_buff *skb);
- void dev_kfree_skb_any(struct sk_buff *skb);
  - Các hàm giải phóng bộ đệm.
- unsigned char *skb_put(struct sk_buff *skb, int len);
- unsigned char *_ _skb_put(struct sk_buff *skb, int len);
  - Cập nhật trường tail và len của cấu trúc sk_buff. Các hàm này được sử dụng để thêm dữ liệu từ cuối bộ đệm.

Các hàm quan trọng trong bộ đệm Socket

- unsigned char *skb_push(struct sk_buff *skb, int len);
- unsigned char *_ _skb_push(struct sk_buff *skb, int len);
  - Các hàm sử dụng để giảm skb->data và tăng skb->len. Tương tự như hàm skb_put nhưng dữ liệu được thêm vào từ đầu của gói tin.
- int skb_tailroom(struct sk_buff *skb);
  - Trả về tổng dung lượng còn trống để thêm dữ liệu vào bộ đệm.
- int skb_headroom(struct sk_buff *skb);
  - Trả về tổng dung lượng còn trống đứng trước vùng dữ liệu.
- void skb_reserve(struct sk_buff *skb, int len);
  - Tăng cả data và tail. Hàm này sử dụng để đặt chỗ phần tiêu đề trước khi đưa dữ liệu vào bộ đệm.

Các hàm quan trọng trong bộ đệm Socket

- unsigned char *skb_pull(struct sk_buff *skb, int len);
  - Gỡ bỏ dữ liệu từ phần tiêu đề của gói tin.
- int skb_is_nonlinear(struct sk_buff *skb);
  - Trả về true nếu skb bị phân chia thành nhiều mảnh.
- int skb_headlen(struct sk_buff *skb);
  - Trả về độ dài của mảnh đầu tiên của skb.
- void *kmap_skb_frag(skb_frag_t *frag);
- void kunmap_skb_frag(void *vaddr);
  - Nếu muốn truy cập trực tiếp các mảnh của skb từ nhân, ta sử dụng các hàm này để liên kết và bỏ liên kết đến chúng

HỎI - ĐÁP