1. 網路設備驅動:
   * 一般來說，所謂網路就等於MAC加PHY(Network = MAC + PHY)，目前常見有兩種作法第一種是把MAC加到SOC芯片中，第二種作法是加在SOC外部，但通常不會有PHY加到SOC中。
   * 除了MAC之外還需要加上PHY，通常MAC會透過MII/RMII或GMII/RGMII的介面來相連(如下表一整理)。MAC層支持雙工或半雙工模式下運行。MAC可編程，並具有直接存儲器接口的專用DMA。
   * 對於PHY而言，IEEE定義了0~15這些暫存器的功能，16~31這16個寄存器由廠商自行實現。也就是說，所以照理說世界上所有的PHY芯片，其中0~15這16個寄存器是一模一樣的。僅靠這16個寄存器是完全可以驅動起PHY芯片的，至少能保證基本的網絡數據通信，因此Linux內核有通用PHY驅動。按道理來講，不管你使用的是哪個廠家的PHY芯片，都能使用這些通用驅動。

(表一:網路相關介面功能整理)

|  |  |  |
| --- | --- | --- |
| **名稱** | **全稱** | **描述** |
| MAC地址 | MAC Address | 每個網路設備的唯一標識。它是硬件層面的地址。 |
| PHY | Physical Layer | 網路技術中的基礎層級，負責管理實際的硬體傳輸和訊號處理。 |
| MII | Media Independent Interface | IEEE-802.3 定義的乙太網標準接口，MII 接口用於乙太網MAC 連接PHY 芯片。 |
| RMII | Reduced Media Independent Interface | 精簡的介質獨立接口，也就是MII 接口的精簡版本，使用的接線較少。 |
| GMII | Gigabit Media Independent Interface | MII的擴展版，旨在支持千兆位以太網（Gigabit Ethernet）。GMII的主要目的是提供更高的數據速率和更強的功能。 |
| RGMII | Reduced Gigabit Media Independent Interface | GMII的精簡版，旨在減少信號線數量和Pin腳數量，同時保留千兆位以太網的高性能。 |

1. IEEE對PHY前16個暫存器定義(如下表二)。

其中暫存器2和暫存器3為PHY的ID暫存器全稱是組織唯一標識符（Organizationally Unique Identifier，OUI），OUI 一共32位，分為三部分：22位的ID + 6位廠商型號ID + 4位廠商版本ID。

|  |  |  |
| --- | --- | --- |
| **暫存器編號** | **暫存器名稱** | **描述** |
| 0 | 控制暫存器 | 用於設置PHY的基本控制功能，包括重置、速度選擇、自動協商啟用等。 |
| 1 | 狀態暫存器 | 顯示PHY的狀態信息，如連接狀態、自動協商完成狀態等。 |
| 2 | PHY識別碼1 | 製造商OUI（組織唯一標識符）的高位部分。 |
| 3 | PHY識別碼2 | 製造商OUI的低位部分，以及製造商的型號和修訂號。 |
| 4 | 自動協商廣告 | 廣告本設備支持的自動協商能力。 |
| 5 | 自動協商連接夥伴能力 | 顯示連接夥伴廣告的能力。 |
| 6 | 自動協商擴展 | 顯示更多的自動協商能力和狀態。 |
| 7 | 自動協商次級狀態 | 擴展狀態和錯誤信息。 |
| 8 | 廣播控制暫存器 | 控制廣播的功能。 |
| 9 | 中斷暫存器 | 中斷功能的設置和狀態。 |
| 10 | 通道間隔暫存器 | 設置通道間隔的參數。 |
| 11 | 能量檢測暫存器 | 檢測能量的參數設置。 |
| 12 | 接收狀態暫存器 | 接收狀態的詳細信息。 |
| 13 | 失敗原因暫存器 | 顯示連接失敗的原因。 |
| 14 | 限制暫存器 | 設置和顯示PHY的限制參數。 |
| 15 | 擴展狀態暫存器 | 顯示更多的擴展狀態信息。 |

1. Linux Kernel中網路架構:

* Linux內核使用net\_device結構體表示一個具體的網絡設備，net\_device是整個網絡驅動的靈魂。網絡驅動的核心就是初始化net\_device結構體中的各個成員變量，然後將初始化完成後的net\_device註冊到Linux內核中。net\_device結構體定義在include/linux/netdevice.h中。
* 其中比較重要的是netdev\_ops是網絡設備的操作集函數，包含了一系列的網絡設備操作回調函數，類似於字元設備中的file\_operations，稍後會講解netdev\_ops結構體。

1. Linux Kernel網路設備驅動常見API及功能(如下表三)。

* 編寫網絡驅動時首先要申請net\_device，使用alloc\_netdev函數來申請net\_device。
* 當我們註銷網絡驅動時，需要釋放掉前面已經申請到的net\_device，釋放函數為free\_netdev。
* net\_device申請並初始化完成以後，就需要向內核註冊net\_device，要用到的函數是register\_netdev。
* 若是向內核註銷net\_device使用函數unregister\_netdev。

(表三: 常見Linux網路設備API及其功能)

|  |  |
| --- | --- |
| **API函式** | **功能描述** |
| alloc\_netdev | 申請net\_device |
| free\_netdev | 釋放已經申請到的net\_device |
| register\_netdev | 向內核註冊已申請並初始化完成的net\_device |
| unregister\_netdev | 向內核註銷net\_device |

* net\_device有個非常重要的成員變數：netdev\_ops，為net\_device\_ops結構體指針類型，這就是網絡設備的操作集。net\_device\_ops結構體定義在include/linux/netdevice.h文件中，net\_device\_ops結構體裡面都是一些以“ndo\_”開頭的函數，這些函數需要網絡驅動編寫人員去實現，不需要全部都實現，可以根據實際驅動情況實現其中的一部分即可。

1. NAPI處理機制:

* 我們知道常見接收數據有兩種方法：輪詢或中斷。在Linux裡面的網絡數據接收也有輪詢和中斷兩種方式。但是我們也知道輪詢跟中斷各有其優缺點，因此Linux系統在這兩個處理方式的基礎上，特別開發出網路接收數據的處理方法就是NAPI。
* NAPI 是一種高效的網路處理技術。NAPI 的核心思想是不完全依賴中斷來讀取網絡數據，而是使用中斷來喚醒數據接收服務程序，並在接收服務程序中使用POLL的方法來輪詢處理數據。

1. 常見的NAPI函式及其功能(如下表四)。

* 首先要初始化一個napi\_struct實例，使用netif\_napi\_add函數，此函數定義在net/core/dev.c中。
* 如果要刪除NAPI，使用netif\_napi\_del函數即可。
* 初始化完NAPI以後，必須使能才能使用，使用函數napi\_enable。
* 關閉NAPI使用napi\_disable函數即可。
* 使用napi\_schedule\_prep函數檢查NAPI是否可以進行調度。
* 如果可以調度的話就進行調度，使用\_\_napi\_schedule函數完成NAPI調度。
* NAPI處理完成以後需要調用napi\_complete函數來標記NAPI處理完成。

(表四:NAPI常見函式及其功能)

|  |  |
| --- | --- |
| **API函數** | **功能描述** |
| netif\_napi\_add | 初始化一個napi\_struct實例，此函數定義在net/core/dev.c中。 |
| netif\_napi\_del | 刪除NAPI。 |
| napi\_enable | 使能NAPI，以便使用。 |
| napi\_disable | 關閉NAPI。 |
| napi\_schedule\_prep | 檢查NAPI是否可以進行調度。 |
| \_\_napi\_schedule | 完成NAPI調度。 |
| napi\_complete | 標記NAPI處理完成。 |

※結論:

Linux系統中，網路設備非常複雜，所以這裡所舉的內容都只是滄海一粟，實際開發中難度也相當高，但是不用驚慌，因為通常廠商會幫我們完成相關驅動編寫。我們通常只要調試及使用就好。