# Chapitre 16

# Détection d'une carte réseau (2) Le cas d'Ethernet

Nous venons de voir comment charger un pilote de périphérique. Voyons ce qu'il en est dans le cas d'Ethernet.

## 16.1 Structures de données pour Ethernet

### 16.1.1 Constantes

Code Linux 2.6.10

Les constantes nécessaires à l'implémentation Linux d'Ethernet sont définies dans le fichier `linux/include/linux/if_ether.h` :

```
6     *              Definitions globales pour l'interface Ethernet IEEE 802.3.
7     *
8     * Version :    @(#)if_ether.h  1.0.1a  02/08/94
9     *
10    * Auteurs :    Fred N. van Kempen, <waltje@uWalt.NL.Mugnet.ORG>
11    *              Donald Becker, <becker@super.org>
12    *              Alan Cox, <alan@redhat.com>
13    *              Steve Whitehouse, <gw7rrm@eeshack3.swan.ac.uk>
[...]
21  #ifndef _LINUX_IF_ETHER_H
22  #define _LINUX_IF_ETHER_H
23
24  /*
25   *      Constantes magiques Ethernet IEEE 802.3. Les tailles des trames omettent
26   *      le preambule et FCS/CRC (frame check sequence).
27   */
28
29  #define ETH_ALEN        6               /* Octets dans une adresse ethernet      */
30  #define ETH_HLEN        14              /* Nombre total d'octets dans l'en-tete. */
31  #define ETH_ZLEN        60              /* Min. octets dans la trame sans FCS    */
32  #define ETH_DATA_LEN    1500            /* Max. octets dans les donnees          */
33  #define ETH_FRAME_LEN   1514            /* Max. octets dans la trame sans FCS    */
```

Autrement dit :

- Une adresse MAC n'ayant pas de structure particulière, à part le nombre de bits ou d'octets de celle-ci, Linux définit une constante symbolique `ETH_ALEN` (pour *Address LENgth*) spécifiant le nombre d'octets d'une telle adresse.
- La longueur `HLEN` (pour *Header LENgth*) d'un en-tête Ethernet (sans préambule at sans CRC) est de 14 octets.
- La longueur minimum d'une trame (sans préambule et sans CRC) acceptable est de 60 octets pour éviter de prendre en compte les débuts de trames qui subissent une collision.
- La longueur maximum des données, ou MTU Ethernet, est de 1 500 octets, comme nous l'avons déjà vu.
- La longueur maximum de la trame (sans préambule et sans CRC) est donc de 1 514 octets.

### 16.1.2 Les types de protocole

Nous avons vu que l'en-tête d'une trame Ethernet comprend un champ `type` qui permet de déterminer à quelle suite de protocoles doit être livré la trame. Les types de protocole permis sont repérés par des constantes symboliques définies dans le fichier `linux/include/linux/if_ether.h` :

Code Linux 2.6.10

```
35  /*
36   *      Ce sont les ID des protocoles Ethernet definis.
37   */
38
39  #define ETH_P_LOOP      0x0060          /* Paquet Loopback Ethernet                  */
40  #define ETH_P_PUP       0x0200          /* Paquet Xerox PUP                          */
41  #define ETH_P_PUPAT     0x0201          /* Paquet Trans Adresse Xerox PUP            */
42  #define ETH_P_IP        0x0800          /* Paquet du protocole Internet              */
```

```
43  #define ETH_P_X25       0x0805          /* CCITT X.25                                */
44  #define ETH_P_ARP       0x0806          /* Paquet de Resolution d'Adresse            */
45  #define ETH_P_BPQ       0x08FF          /* Paquet Ethernet G8BPQ AX.25
                                               [ ID NON OFFICIELLEMENT ENREGISTRE]       */
46  #define ETH_P_IEEEPUP   0x0a00          /* Paquet PUP Xerox IEEE802.3                */
47  #define ETH_P_IEEEPUPAT 0x0a01          /* Paquet Trans d'Adresse Xerox IEEE802.3 PUP */
48  #define ETH_P_DEC       0x6000          /* DEC Assigned proto                        */
49  #define ETH_P_DNA_DL    0x6001          /* DEC DNA Dump/Load                         */
50  #define ETH_P_DNA_RC    0x6002          /* DEC DNA Remote Console                    */
51  #define ETH_P_DNA_RT    0x6003          /* Routage DEC DNA                           */
52  #define ETH_P_LAT       0x6004          /* DEC LAT                                   */
53  #define ETH_P_DIAG      0x6005          /* Diagnostic DEC                            */
54  #define ETH_P_CUST      0x6006          /* DEC Customer use                          */
55  #define ETH_P_SCA       0x6007          /* DEC Systems Comms Arch                    */
56  #define ETH_P_RARP      0x8035          /* Paquet de Resolution d'adresse inverse    */
57  #define ETH_P_ATALK     0x809B          /* Appletalk DDP                             */
58  #define ETH_P_AARP      0x80F3          /* Appletalk AARP                            */
59  #define ETH_P_8021Q     0x8100          /* En-tete etendu VLAN 802.1Q                */
60  #define ETH_P_IPX       0x8137          /* IPX sur DIX                               */
61  #define ETH_P_IPV6      0x86DD          /* IPv6 sur bluebook                         */
62  #define ETH_P_WCCP      0x883E          /* Protocole de coordination de cache Web
63                                           * defini dans draft-wilson-wrec-wccp-v2-00.txt */
64  #define ETH_P_PPP_DISC  0x8863          /* Messages de decouverte PPPoE              */
65  #define ETH_P_PPP_SES   0x8864          /* Messages de session PPPoE                 */
66  #define ETH_P_MPLS_UC   0x8847          /* Trafic individuel MPLS                    */
67  #define ETH_P_MPLS_MC   0x8848          /* Trafic multidiffusion MPLS                */
68  #define ETH_P_ATMMPOA   0x884c          /* MultiProtocol sur ATM                     */
69  #define ETH_P_ATMFATE   0x8884          /* Transport fonde sur des trames ATM
70                                           * sur Ethernet
71                                           */
72  #define ETH_P_EDP2      0x88A2          /* Coraid EDP2                               */
73
74  /*
75   *      Types non DIX. Ne devrait pas interferer avec les types 1500.
76   */
77
78  #define ETH_P_802_3     0x0001          /* Type muet pour les trames 802.3           */
79  #define ETH_P_AX25      0x0002          /* Id de protocole muet pour AX.25           */
80  #define ETH_P_ALL       0x0003          /* Chaque paquet (soyez soigneux !!!)        */
81  #define ETH_P_802_2     0x0004          /* Trames 802.2                              */
82  #define ETH_P_SNAP      0x0005          /* En interne seulement                      */
83  #define ETH_P_DDCMP     0x0006          /* DEC DDCMP : en interne seulement          */
84  #define ETH_P_WAN_PPP   0x0007          /* Type muet pour les trames WAN PPP         */
85  #define ETH_P_PPP_MP    0x0008          /* Type muet pour les trames PPP MP          */
86  #define ETH_P_LOCALTALK 0x0009          /* Pseudo-type Localtalk                     */
87  #define ETH_P_PPPTALK   0x0010          /* Type muet pour Atalk sur PPP              */
88  #define ETH_P_TR_802_2  0x0011          /* Trames 802.2                              */
89  #define ETH_P_MOBITEX   0x0015          /* Mobitex (kaz@cafe.net)                    */
90  #define ETH_P_CONTROL   0x0016          /* Trames de controle specifiques a la carte */
91  #define ETH_P_IRDA      0x0017          /* Linux-IrDA                                */
92  #define ETH_P_ECONET    0x0018          /* Acorn Econet                              */
93  #define ETH_P_HDLC      0x0019          /* Trames HDLC                               */
```

dont nous ne retiendrons que les types `ETH_P_LOOP`, `ETH_P_IP` et `ETH_P_ARP`.

### 16.1.3 Structure d'en-tête de trame

Linux spécifie la structure d'un en-tête de trame Ethernet, constituée uniquement des deux adresses source et de destination et du type, sous le nom `ethhdr` à la suite dans le même fichier `linux/include/linux/if_ether.h`:

Code Linux 2.6.10

```
95  /*
```

```
96   *      Ceci est l'en-tete de trame Ethernet.
97   */
98
99  struct ethhdr
100 {
101         unsigned char   h_dest[ETH_ALEN];       /* adr eth de destination     */
102         unsigned char   h_source[ETH_ALEN];     /* adr eth source             */
103         unsigned short  h_proto;                /* champ ID de type de paquet */
104 } __attribute__((packed));
```

## 16.2 Détection et initialisation des cartes Ethernet

Nous avons vu que, lors du démarrage du système Linux, huit cartes Ethernet sont sondées et leurs pilotes de périphérique installées. Étudions ceci plus en détail maintenant.

### 16.2.1 Détection des périphériques Ethernet

#### 16.2.1.1 Fonction principale

Code Linux 2.6.10

La fonction `ethif_probe2()` est définie dans le fichier `linux/drivers/net/Space.c`:

```
315 /*
316  * Detection unifiee des peripheriques ethernet, segmente par architecture et
317  * par interface de bus. Ceci pilote uniquement les peripheriques legaux pour l'instant.
318  */
319
320 static void __init ethif_probe2(int unit)
321 {
322         unsigned long base_addr = netdev_boot_base("eth", unit);
323
324         if (base_addr == 1)
325                 return;
326
327         (void)( probe_list2(unit, m68k_probes, base_addr == 0) &&
328                 probe_list2(unit, mips_probes, base_addr == 0) &&
329                 probe_list2(unit, eisa_probes, base_addr == 0) &&
330                 probe_list2(unit, mca_probes, base_addr == 0) &&
331                 probe_list2(unit, isa_probes, base_addr == 0) &&
332                 probe_list2(unit, parport_probes, base_addr == 0));
333 }
```

Autrement dit :

- On déclare une adresse de base, que l'on instantie grâce à la fonction `netdev_boot_base()` étudiée ci-dessous. S'il existe déjà un périphérique portant ce nom ("eth0" pour l'unité égale à zéro), on a terminé.
- On essaie de détecter les cartes pour les microprocesseurs 68000 de Motorola et MIPS, puis pour différents systèmes de bus (EISA, MCA et ISA) et enfin pour le port parallèle, grâce à la fonction `probe_list()` étudiée ci-dessous.

Rappelons que `unit` prenait les valeurs 0 à 7 dans la fonction `net_olddevs_init()`.

#### 16.2.1.2 Instantiation de l'adresse de base

Code Linux 2.6.10

La fonction `netdev_boot_base()` est définie dans le fichier `linux/net/core/dev.c`:

```
410 /**
411  *      netdev_boot_base        - obtient une adresse au moment du demarrage
```

```
412  *      @prefix : prefixe du peripherique reseau
413  *      @unit : id du peripherique reseau
414  *
415  *      Initialisation au moment du demarrage de l'adresse de base du peripherique.
416  *      Les initialisations trouvees sont positionnees pour le peripherique a utiliser
417  *      plus tard lors du sondage du peripherique.
418  *      Renvoie 0 si aucune initialisation n'est trouvee.
419  */
420 unsigned long netdev_boot_base(const char *prefix, int unit)
421 {
422         const struct netdev_boot_setup *s = dev_boot_setup;
423         char name[IFNAMSIZ];
424         int i;
425
426         sprintf(name, "%s%d", prefix, unit);
427
428         /*
429          * Si le peripherique est deja enregistre alors renvoyer une base de 1
430          * pour indiquer de ne pas sonder cette interface
431          */
432         if (__dev_get_by_name(name))
433                 return 1;
434
435         for (i = 0; i < NETDEV_BOOT_SETUP_MAX; i++)
436                 if (!strcmp(name, s[i].name))
437                         return s[i].map.base_addr;
438         return 0;
439 }
```

Autrement dit :

– On déclare un tableau de structures `netdev_boot_setup`, que l'on instantie avec le tableau `dev_boot_setup[]`.

Le type `struct netdev_boot_setup` est défini dans le fichier `linux/include/linux/-netdevice.h` : Code Linux 2.6.10

```
243 /*
244  * Cette structure contient les indications des peripheriques configures au moment du
245  * demarrage. Elles sont utilisees pour le sondage des peripheriques.
246  */
247 struct netdev_boot_setup {
248         char name[IFNAMSIZ];
249         struct ifmap map;
250 };
251 #define NETDEV_BOOT_SETUP_MAX 8
```

Le type `struct ifmap` est défini dans le fichier `linux/include/linux/if.h` : Code Linux 2.6.10

```
84  /*
85   *      Structure des parametres du peripherique. Je n'ai fait que concevoir un
86   *      beau schema utilisant seulement les modules chargeables avec des arguments
87   *      pour les options des peripheriques et alors vinrent les gens du PCMCIA 8)
88   *
89   *      Ah bien. Le cote get() de ceci est bon pour WDSETUP, et il sera manipulable
90   *      pour deboguer des choses. Le cote set() est bien pour l'instant et etant
91   *      tres petit il pourrait etre pire de le garder pour une configuration propre.
92   */
93
94  struct ifmap
95  {
96          unsigned long mem_start;
97          unsigned long mem_end;
```

```
98          unsigned short base_addr;
99          unsigned char irq;
100         unsigned char dma;
101         unsigned char port;
102         /* 3 octets disperses */
103 };
```

Code Linux 2.6.10

Le tableau `dev_boot_setup[]` est défini dans le fichier `linux/net/core/dev.c`:

```
346 /*****************************************************************************
347
348           Routines pour le parametrage des peripheriques au moment du demarrage
349
350 *****************************************************************************/
351
352 /* Table de configuration au moment du demarrage */
353 static struct netdev_boot_setup dev_boot_setup[NETDEV_BOOT_SETUP_MAX];
```

- On déclare un nom de périphérique réseau et un index.
- On initialise le nom avec le préfxe et l'unité passés en argument.
  On obtiendra donc "eth0" pour le premier.
- Si un périphérique portant ce nom est déjà enregistré, on renvoie 1.
- Si ce nom apparaît dans le tableau `dev_boot_setup[]`, on renvoie l'adresse de base spécifiée dans ce tableau.
- Sinon on renvoie 0.

#### 16.2.1.3 Détection suivant une liste

Code Linux 2.6.10

La fonction `probe_list2()` est définie dans le fichier `linux/drivers/net/Space.c`:

```
111 static int __init probe_list2(int unit, struct devprobe2 *p, int autoprobe)
112 {
113         struct net_device *dev;
114         for (; p->probe; p++) {
115                 if (autoprobe && p->status)
116                         continue;
117                 dev = p->probe(unit);
118                 if (!IS_ERR(dev))
119                         return 0;
120                 if (autoprobe)
121                         p->status = PTR_ERR(dev);
122         }
123         return -ENODEV;
124 }
```

Autrement dit:

- On parcourt la liste passée en argument à la recherche d'un périphérique:
  - Si la valeur `autoprobe` de l'adresse de base passée en argument est non nulle ainsi que le statut, on ne fait rien pour cet élément.
    Ce n'est pas le cas des six listes que nous avons à considérer puisque l'adresse de base était toujours zéro.
  - On essaie de détecter le périphérique associé à l'élément de cette liste et à l'unité passée en argument.
  - Si on n'a pas trouvé ce périphérique, on renvoie 0. Sinon, si l'adresse de base passée en argument était non nulle, on renseigne le champ statut de cet élément.
- Si aucune des fonctions de détection n'a réussi, on renvoie l'opposé du code d'erreur `ENODEV`.

Le type `struct devprobe2` est défini dans le fichier `linux/drivers/net/Space.c`: Code Linux 2.6.10

```
106 struct devprobe2 {
107         struct net_device *(*probe)(int unit);
108         int status;     /* non-nul si l'auto-detection a echouee */
109 };
```

Les six listes (en fait tableaux) sont définies dans le fichier `linux/drivers/net/Space.c`, par exemple pour `isa_probes[]` : Code Linux 2.6.10

```
165 /*
166  * ISA detecte les adresses < 0x400 (y compris celles qui recherchent aussi
167  * les cartes EISA/PCI/MCA en plus des cartes ISA).
168  */
169 static struct devprobe2 isa_probes[] __initdata = {
170 #ifdef CONFIG_HP100             /* ISA, EISA & PCI */
171         {hp100_probe, 0},
172 #endif
173 #ifdef CONFIG_3C515
174         {tc515_probe, 0},
175 #endif
176 #ifdef CONFIG_ULTRA
177         {ultra_probe, 0},
178 #endif
179 #ifdef CONFIG_WD80x3
180         {wd_probe, 0},
181 #endif
182 #ifdef CONFIG_EL2               /* 3c503 */
183         {el2_probe, 0},
[...]
231 #ifdef CONFIG_EL1               /* 3c501 */
232         {el1_probe, 0},
233 #endif
[...]
258         {NULL, 0},
259 };
```

### 16.2.2 Initialisation d'une carte Ethernet

Lors de la conception du pilote d'une carte particulière, on fait heureusement, pour la plus grande part, appel à une initialisation par défaut, implémentée dans le noyau Linux pour les grands types de cartes (située dans le fichier `linux/drivers/net/net_init.c`), ce qui facilite le travail de conception d'un pilote de périphérique:

- `ether_setup()` pour les cartes Ethernet, que nous allons étudier ici ;
- `ltalk_setup()` pour les cartes *LocalTalk* ;
- `fc_setup()` pour les périphériques utilisant de la fibre optique ;
- `fddi_setup()` pour les réseaux FDDI (*Fiber Distributed Data Interface*) ;
- `hippi_setup()` pour les périphériques à haut débit HIPPI ;
- `tr_configure()` pour configurer les interfaces réseau *token ring*.

La mise en place de valeurs par défaut dans le cas de Ethernet est effectuée grâce à la fonction `ether_setup()`. Celle-ci est définie dans le fichier `linux/net/ethernet/eth.c`: Code Linux 2.6.10

```
266 /*
267  * Renseigne les champs de la structure de peripherique avec les valeurs generiques
             d'ethernet.
268  */
269 void ether_setup(struct net_device *dev)
```

```
270 {
271         dev->change_mtu         = eth_change_mtu;
272         dev->hard_header        = eth_header;
273         dev->rebuild_header     = eth_rebuild_header;
274         dev->set_mac_address    = eth_mac_addr;
275         dev->hard_header_cache  = eth_header_cache;
276         dev->header_cache_update= eth_header_cache_update;
277         dev->hard_header_parse  = eth_header_parse;
278
279         dev->type               = ARPHRD_ETHER;
280         dev->hard_header_len    = ETH_HLEN;
281         dev->mtu                = 1500; /* eth_mtu */
282         dev->addr_len           = ETH_ALEN;
283         dev->tx_queue_len       = 1000; /* Ethernet exige de grandes files d'attente */
284         dev->flags              = IFF_BROADCAST|IFF_MULTICAST;
285
286         memset(dev->broadcast,0xFF, ETH_ALEN);
287
288 }
```

Autrement dit :

- elle assigne des valeurs par défaut au descripteur de périphérique réseau pour certaines fonctions de gestions ;
- elle indique le type matériel de l'interface (utilisé par ARP pour déterminer quelle sorte d'adresse physique est supportée par le périphérique réseau) ; il s'agit ici évidemment du type associé à Ethernet ;
- elle spécifie la longueur de l'en-tête d'une trame, à savoir 14 octets pour Ethernet ;
- elle spécifie la longueur maximale d'une trame, ou MTU, à savoir 1 500 octets pour Ethernet ; remarquons qu'on aurait pu utiliser la constante symbolique `ETH_DATA_LEN` ;
- elle spécifie la longueur effective d'une adresse, à savoir six octets dans le cas d'Ethernet ;
- elle spécifie la taille maximale de la file d'attente en transmission, à savoir 1000 trames pour Ethernet ;
- elle spécifie l'adresse de diffusion générale, à savoir six octets de valeur `FFh`, c'est-à-dire que des 1.

Les fonctions de gestion Ethernet par défaut devront être étudiées au fur et à mesure des besoins. En fait nous n'en aurons pas besoin dans cet ouvrage.

### 16.2.3 Allocation d'un descripteur de périphérique réseau

#### 16.2.3.1 Cas de Ethernet

Code Linux 2.6.10

La fonction `alloc_etherdev()` est définie dans le fichier `linux/net/ethernet/eth.c` :

```
291 /**
292  * alloc_etherdev - Alloue et initialise un peripherique ethernet
293  * @sizeof_priv : Taille de la structure privee additionnelle du pilote a allouer
294  *      pour ce peripherique ethernet
295  *
296  * Renseigne les champs de la structure du peripherique avec les valeurs generiques
297  * ethernet. Ne fait rien d'autre de fondamental que d'enregistrer le peripherique.
298  *
299  * Construit un nouveau peripherique reseau, le complete avec une zone de donnees
300  * privees de taille @sizeof_priv. Un alignement de 32 octets (pas bits) est force
301  * pour cette zone de donnees privees.
302  */
303
```

```
304 struct net_device *alloc_etherdev(int sizeof_priv)
305 {
306         return alloc_netdev(sizeof_priv, "eth%d", ether_setup);
307 }
```

qui renvoie à la fonction générale `alloc_netdev()` avec les deux paramètres propres à Ethernet, à savoir le nom et la fonction d'initialisation `ether_setup()`.

#### 16.2.3.2 Cas général

La fonction générale est définie dans le fichier `linux/drivers/net/net_init.c`: Code Linux 2.6.10

```
73  struct net_device *alloc_netdev(int sizeof_priv, const char *mask,
74                                  void (*setup)(struct net_device *))
75  {
76          void *p;
77          struct net_device *dev;
78          int alloc_size;
79
80          /* On s'assure d'un alignement sur 32 octets de la zone privee */
81
82          alloc_size = (sizeof(struct net_device) + NETDEV_ALIGN_CONST)
83                          & ~NETDEV_ALIGN_CONST;
84          alloc_size += sizeof_priv + NETDEV_ALIGN_CONST;
85
86          p = kmalloc (alloc_size, GFP_KERNEL);
87          if (!p) {
88                  printk(KERN_ERR "alloc_dev: Unable to allocate device.\n");
89                  return NULL;
90          }
91
92          memset(p, 0, alloc_size);
93
94          dev = (struct net_device *)(((long)p + NETDEV_ALIGN_CONST)
95                                  & ~NETDEV_ALIGN_CONST);
96          dev->padded = (char *)dev - (char *)p;
97
98          if (sizeof_priv)
99                  dev->priv = netdev_priv(dev);
100
101         setup(dev);
102         strcpy(dev->name, mask);
103
104         return dev;
105 }
```

Autrement dit:

- On déclare une adresse, un descripteur de périphérique et une taille.
- On initialise cette taille en tenant compte de la taille d'un descripteur de périphérique, de la zone des données privées et de l'alignement sur 32 octets.

  La constante symbolique `NETDEV_ALIGN_CONST` est définie dans le fichier `linux/include/linux/netdevice.h`: Code Linux 2.6.10

  ```
  493 #define NETDEV_ALIGN            32
  494 #define NETDEV_ALIGN_CONST      (NETDEV_ALIGN - 1)
  ```

- On essaie de réserver de la place en mémoire vive pour le descripteur de périphérique. Si on n'y parvient pas, on affiche un message noyau et on renvoie `NULL`.
- On initialise cette zone de mémoire à zéro.

– On initialise l'adresse du descripteur de périphérique.
– On renseigne le champ de remplissage de ce descripteur de périphérique.
– Si la taille de la zone privée passée en argument est non nulle, on renseigne le champ concernant celle-ci du descripteur de périphérique.

Code Linux 2.6.10

La fonction en ligne `netdev_priv()` est définie dans le fichier `linux/include/linux/-netdevice.h`:

```
496 static inline void *netdev_priv(struct net_device *dev)
497 {
498         return (char *)dev + ((sizeof(struct net_device)
499                                         + NETDEV_ALIGN_CONST)
500                                 & ~NETDEV_ALIGN_CONST);
501 }
```

– On fait appel à la fonction d'initialisation passée en argument, qui dépend du type de carte réseau.
– On renseigne le champ nom du descripteur de périphérique réseau avec le nom passé en argument.
– On renvoie l'adresse du descripteur de périphérique ainsi créé.

## 16.3 Opérations liées à Ethernet

Nous avons vu au chapitre 14 que le descripteur de périphérique réseau prévoie un champ faisant référence à un ensemble d'opérations Ethernet (champ égal à `NULL` dans le cas d'un périphérique non Ethernet).

Code Linux 2.6.10

Le type `struct ethtool_ops` de ce champ est défini dans le fichier `linux/include/linux/-ethtool.h`:

```
264 /**
265  * &ethtool_ops - Change et rappelle les positionnements du peripherique reseau
266  * get_settings : recupere les positionnements specifiques au peripherique
267  * set_settings : positionne ce qui est specifique au peripherique
268  * get_drvinfo : recupere les informations sur le peripherique
269  * get_regs : recupere les registres du peripherique
270  * get_wol : dit si Wake-on-Lan est active
271  * set_wol: active ou desactive Wake-on-Lan
272  * get_msglevel : recupere la couche message du peripherique
273  * set_msglevel : positionne la couche message du peripherique
274  * nway_reset : redemarre l'autonegociation
275  * get_link : recupere le statut de lien
276  * get_eeprom : lit les donnees depuis l'EEPROM du peripherique
277  * set_eeprom : ecrit des donnees sur l'EEPROM du peripherique
278  * get_coalesce : recupere les parametres de combinaison de l'interruption
279  * set_coalesce : positionne les parametres de combinaison de l'interruption
280  * get_ringparam : recupere les tailles de l'anneau
281  * set_ringparam : positionne les tailles de l'anneau
282  * get_pauseparam : recupere les parametres de pause
283  * set_pauseparam : positionne les parametres de pause
284  * get_rx_csum : dit si le controle de la somme de controle en reception est activee ou non
285  * set_rx_csum : active ou desactive le controle de la somme de controle en reception
286  * get_tx_csum : dit si le controle de la somme de controle en emission est active ou non
287  * set_tx_csum : active ou desactive le controle de la somme de controle en emission
288  * get_sg : dit si la dispersion-rassemblement est activee
289  * set_sg : active ou desactive la dispersion-rassemblement
290  * get_tso : dit si la segmentation TCP est activee
291  * set_tso : active ou desactive la segmentation TCP
292  * self_test : effectue les auto-tests specifies
```

```
293  * get_strings : renvoie un ensemble de chaines de caracteres qui decrit les objets demandes
294  * phys_id : identifie le peripherique
295  * get_stats : renvoie les statistiques concernant le peripherique
296  *
297  * Description :
298  *
299  * get_settings :
300  *      @get_settings passe une &ethtool_cmd a renseigner. Elle renvoie
301  *      un errno negatif ou zero.
302  *
303  * set_settings :
304  *      @set_settings passe une &ethtool_cmd et doit essayer de positionner
305  *      tout ce que supporte ce peripherique. Elle renvoie une valeur d'erreur
306  *      si quelque chose se passe mal (0 sinon).
307  *
308  * get_eeprom :
309  *      Doit renseigner le champ magic. Ne necessite pas de verifier si len est nulle
310  *      ou l'emballage. Renseigne l'argument des donnees avec les valeurs eeprom
311  *      de offset a offset + len. Met a jour len sur la quantite lue.
312  *      Renvoie une erreur ou zero.
313  *
314  * set_eeprom :
315  *      Doit valider le champ magic. Ne necessite pas de verifier si len est nulle
316  *      ou l'emballage. Met a jour la quantite ecrite. Renvoie une erreur
317  *      ou zero.
318  */
319 struct ethtool_ops {
320         int     (*get_settings)(struct net_device *, struct ethtool_cmd *);
321         int     (*set_settings)(struct net_device *, struct ethtool_cmd *);
322         void    (*get_drvinfo)(struct net_device *, struct ethtool_drvinfo *);
323         int     (*get_regs_len)(struct net_device *);
324         void    (*get_regs)(struct net_device *, struct ethtool_regs *, void *);
325         void    (*get_wol)(struct net_device *, struct ethtool_wolinfo *);
326         int     (*set_wol)(struct net_device *, struct ethtool_wolinfo *);
327         u32     (*get_msglevel)(struct net_device *);
328         void    (*set_msglevel)(struct net_device *, u32);
329         int     (*nway_reset)(struct net_device *);
330         u32     (*get_link)(struct net_device *);
331         int     (*get_eeprom_len)(struct net_device *);
332         int     (*get_eeprom)(struct net_device *, struct ethtool_eeprom *, u8 *);
333         int     (*set_eeprom)(struct net_device *, struct ethtool_eeprom *, u8 *);
334         int     (*get_coalesce)(struct net_device *, struct ethtool_coalesce *);
335         int     (*set_coalesce)(struct net_device *, struct ethtool_coalesce *);
336         void    (*get_ringparam)(struct net_device *, struct ethtool_ringparam *);
337         int     (*set_ringparam)(struct net_device *, struct ethtool_ringparam *);
338         void    (*get_pauseparam)(struct net_device *, struct ethtool_pauseparam*);
339         int     (*set_pauseparam)(struct net_device *, struct ethtool_pauseparam*);
340         u32     (*get_rx_csum)(struct net_device *);
341         int     (*set_rx_csum)(struct net_device *, u32);
342         u32     (*get_tx_csum)(struct net_device *);
343         int     (*set_tx_csum)(struct net_device *, u32);
344         u32     (*get_sg)(struct net_device *);
345         int     (*set_sg)(struct net_device *, u32);
346         u32     (*get_tso)(struct net_device *);
347         int     (*set_tso)(struct net_device *, u32);
348         int     (*self_test_count)(struct net_device *);
349         void    (*self_test)(struct net_device *, struct ethtool_test *, u64 *);
350         void    (*get_strings)(struct net_device *, u32 stringset, u8 *);
351         int     (*phys_id)(struct net_device *, u32);
352         int     (*get_stats_count)(struct net_device *);
353         void    (*get_ethtool_stats)(struct net_device *, struct ethtool_stats *,
                                     u64 *);
```

```
354         int     (*begin)(struct net_device *);
355         void    (*complete)(struct net_device *);
356 };
```

Code Linux 2.6.10

Le type `ethtool_cmd` est défini dans le fichier `linux/include/linux/ethtool.h`:

```
16 /* Ceci doit marcher a la fois pour les espaces utilisateur 32 et 64 bits. */
17 struct ethtool_cmd {
18         u32     cmd;
19         u32     supported;      /* Caracteristiques que cette interface supporte */
20         u32     advertising;    /* Caracteristiques dont cette interface fait la publicite */
21         u16     speed;          /* La vitesse forcee : 10Mb, 100Mb, gigabit */
22         u8      duplex;         /* Duplex, half ou full */
23         u8      port;           /* Quel port connecteur */
24         u8      phy_address;
25         u8      transceiver;    /* Quel transcepteur utiliser */
26         u8      autoneg;        /* Active ou desactive l'auto-negociation */
27         u32     maxtxpkt;       /* Paquets transmis avant de lever l'interruption de
                                      transmission */
28         u32     maxrxpkt;       /* Paquets recus avant de lever l'interruption de reception */
29         u32     reserved[4];
30 };
```

Ces fonctions devraient être étudiées au fur et à mesure des besoins. En fait aucune ne sera utilisée dans cet ouvrage.