Chapitre 17

Détection d'une carte réseau (3) Cas de la carte 3Com 501

Intéressons-nous à un pilote en particulier pour voir à quoi cela ressemble. Nous choisissons celui de la carte 3Com 501, qui a été l'une des premières à être implémentées sous Linux.

17.1 Description du contrôleur de la carte

17.1.1 Commentaire général

Le pilote de périphérique de la carte 3Com 501 fait l'objet des deux fichiers `linux/drivers/net/3c501.h` et `linux/drivers/net/3c501.c`. Un commentaire général se trouve au début du second fichier :

Code Linux 2.6.10

```
1   /* 3c501.c : un pilote Ethernet pour Linux de 3Com 3c501. */
2   /*
3       Ecrit en 1992,1993,1994 par Donald Becker
4
5       Copyright 1993 United States Government, represente par le
6       Directeur de la National Security Agency. Ce logiciel peut etre utilise et
7       distribue conformement aux termes de la GNU General Public License,
8       incorporee ici pour reference.
9
10      Ceci est le pilote de peripherique pour 3Com Etherlink 3c501.
11      N'achetez pas cette carte, meme par plaisanterie. Ses performances sont horribles
12      et elle s'arrete souvent.
13
14      L'auteur originel peut etre contacte par becker@scyld.com ou C/O
15          Scyld Computing Corporation
16          410 Severn Ave., Suite 210
17          Annapolis MD 21403
18
19      Rectifie (encore !) le verrouillage manquant de l'interruption sur le decalage TX/RX.
20                  Alan Cox <Alan.Cox@linux.org>
21
22      Appels a init_etherdev supprimes puisqu'ils ne sont plus necessaires et
23      un peu de menage sur la modularisation. Le pilote ne permet encore que
24      l'adresse par defaut pour les cartes chargees comme module, mais ceci est
25      reellement moins stupide que quiconque utilisant une carte 3c501. :)
26                      19950208 (invid@msen.com)
27
28      Trappes ajoutees pour les interruption frappant a la fenetre lorsque nous effacons et
29      chargeons en TX la carte. Obtenons maintenant 150K/seconde pour FTP avec une carte 3c501.
30      Je joue encore a l'optimisation TX-TX pour voir si nous pouvons atteindre 180-200K/seconde
31      qui semble etre le maximum theorique.
32                  19950402 Alan Cox <Alan.Cox@linux.org>
33
34      Nettoye pour 2.3.x puisque nous brisons maintenant la SMP.
35                  20000208 Alan Cox <alan@redhat.com>
36
37      Verification pour 2.5. Aucun changement significatif
38                  20021009 Alan Cox <alan@redhat.com>
39
40      Rectifie le cas du coin rempli avec des zeros
41                  20030104 Alan Cox <alan@redhat.com>
42
43
44     L'evidence des doutes sur la "forme preferee" de ce code a conduit a
45     un format ouvert non encombre de brevet. Puisque la cle cryptographique de signature
46     fait partie du processus de creation d'un executable, les informations
47     incluant les cles necessaires pour generer un executable fonctionnellement equivalent
48     sont considerees comme faisant parties du code source.
49
50  */
51
52
53  /**
54   * DOC : Notes sur la carte 3c501
```

```
55   *
56   *  Quelques notes sur cette chose si vous avez a l'etudier.  [Alan]
57   *
58   *  De la documentation est disponible chez 3Com. Du a l'age de la carte,
59   *  la reponse standard lorsque vous demandez celle-ci ira de 'soyez serieux'
60   *  a 'donnez-la a un musee'. La documentation est incomplete et pour une bonne part
61   *  d'un interet purement historique.
62   *
63   *  Le systeme de base est un unique tampon qui peut etre utilise pour recevoir ou
64   *  transmettre un paquet. Un troisieme mode de commande existe lorsque vous activez
65   *  la carte.
66   *
67   *  Si elle est en train d'emettre, elle ne peut pas recevoir et vice versa.
68   *  En effet le temps de retourner la carte dans un etat utile apres une operation
69   *  est tres grand.
70   *
71   *  Le pilote marche en gardant la carte en mode reception en attendant qu'un
72   *  paquet arrive. Lorsqu'il en arrive un, il est copie hors du tampon
73   *  et remis au noyau. La carte est rechargee et on y va.
74   *
75   *  Lorsqu'on emet, lp->txing est positionne et la carte est reinitialisee (du
76   *  mode reception) [en perdant eventuellement un paquet juste recu] au mode
77   *  commande. Un paquet est charge et le mode d'emission est lance. La routine
78   *  d'interruption execute un code different pour les interruptions d'emission et
79   *  peut s'occuper de revenir au mode reception ou emission (oui vous devez
80   *  l'aider pour ceci aussi).
81   *
82   * DOC : Problemes
83   *
84   *  Il y a de nombreux retours sur erreur non documentes depuis la carte,
85   *  aussi avez-vous fondamentalement a lui donner un coup de pied et a prier si ca se
86   *  passe mal. Beacoup d'erreurs apparaissent seulement en charge extreme ou si vous
87   *  faites quelque chose que la carte n'aime pas (par exemple toucher a un registre
       au mauvais moment).
88   *
89   *  Le pilote est moins efficace que ce qu'il pourrait etre. Il passe en
90   *  mode reception meme s'il y a encore a emettre. Si cela vous ennuie, achetez
91   *  une vraie carte Ethernet.
92   *
93   *  La combinaison d'un redemarrage en reception lent et pas de vrai filtre multidiffusion
94   *  fait que la carte est inutilisable avec un noyau compile pour la multidiffusion IP
95   *  dans un vrai environnement de multidiffusion. Ceci est du a la carte,
96   *  mais meme sans programme de multidiffusion executant une multidiffusion IP, le
97   *  noyau est dans le groupe 224.0.0.1 et vous entendrez toutes les multidiffusions.
98   *  Une conference s'executant sur cette carte Ethernet et vous renoncerez.
99   *
100  */
```

17.1.2 Les registres

17.1.2.1 Adresses des registres

Les adresses des registres de la carte 3Com 501 sont repérées, par rapport à l'adresse de base, par des constantes symboliques définies dans le fichier `linux/drivers/net/3c501.h`:

Code Linux 2.6.10

```
41 #define RX_STATUS (ioaddr + 0x06)
42 #define RX_CMD    RX_STATUS
43 #define TX_STATUS (ioaddr + 0x07)
44 #define TX_CMD    TX_STATUS
45 #define GP_LOW    (ioaddr + 0x08)
46 #define GP_HIGH   (ioaddr + 0x09)
47 #define RX_BUF_CLR (ioaddr + 0x0A)
```

```
48 #define RX_LOW    (ioaddr + 0x0A)
49 #define RX_HIGH   (ioaddr + 0x0B)
50 #define SAPROM    (ioaddr + 0x0C)
51 #define AX_STATUS (ioaddr + 0x0E)
52 #define AX_CMD    AX_STATUS
53 #define DATAPORT  (ioaddr + 0x0F)
[...]
56 #define EL1_DATAPTR     0x08
57 #define EL1_RXPTR       0x0A
58 #define EL1_SAPROM      0x0C
59 #define EL1_DATAPORT    0x0f
```

Remarquons que `ioaddr` n'est pas défini globalement, mais dans chaque contexte qui utilise les adresses des registres, ce qui est possible puisqu'il s'agit de macros et non de fonctions. Nous verrons le cas, par exemple, pour la fonction `el_receive()` que nous étudierons plus loin.

17.1.2.2 Les registres TCR et TSR

Code Linux 2.6.10

Les bits des registres TCR et `TSR` sont repérés grâce aux constantes symboliques définies dans le fichier `linux/drivers/net/3C501.h`:

```
54 #define TX_RDY          0x08      /* Dans TX_STATUS */
[...]
82 /*
83  *      Registre TX_STATUS.
84  */
85
86 #define TX_COLLISION 0x02
87 #define TX_16COLLISIONS 0x04
88 #define TX_READY 0x08
```

17.1.2.3 Le registre RCR

Code Linux 2.6.10

Les valeurs du registre RCR sont repérées grâce aux constantes symboliques définies dans le fichier `linux/drivers/net/3C501.h`:

```
72 /*
73  *      Mode de reception normal ecrit dans RX_STATUS. Nous devons lever une interruption
74  *      sur les petits paquets pour eviter les verrouillages en reception bidons.
75  */
76
77 #define RX_NORM 0xA8            /* 0x68 == toutes les adresses, 0xA8 seulement pour moi. */
78 #define RX_PROM 0x68            /* Promiscuite senior, hum mode promiscuite.             */
79 #define RX_MULT 0xE8            /* Accepte les paquets multidiffusion.                   */
```

17.1.2.4 Le registre RSR

Code Linux 2.6.10

Les valeurs du registre RSR sont repérées grâce aux constantes symboliques définies dans le fichier `linux/drivers/net/3C501.h`:

```
90 #define RX_RUNT   0x08
91 #define RX_MISSED 0x01       /* A manque un paquet a cause d'une lesion cerebrale de 3c501. */
92 #define RX_GOOD   0x30       /* Bon paquet 0x20 ou depassement simple 0x10.                 */
```

17.1.2.5 Le registre RCA

Code Linux 2.6.10

Les valeurs du registre RCA sont repérées grâce aux constantes symboliques définies dans le fichier `linux/drivers/net/3C501.h`:

```
61 /*
```

```
62  *      Ecrit sur le registre de commande ax.
63  */
64
65 #define AX_OFF   0x00                  /* Irq inhibe, acces au tampon active    */
66 #define AX_SYS   0x40                  /* Charge le tampon                      */
67 #define AX_XMIT  0x44                  /* Emet un paquet                    */
68 #define AX_RX    0x48                  /* Recoit un paquet                      */
69 #define AX_LOOP  0x0C                  /* Mode loopback                         */
70 #define AX_RESET 0x80
[...]
80 #define TX_NORM  0x0A                  /* Leve une interruption pour chaque chose
                                             qui peut accrocher le microprocesseur */
```

17.2 Détection et initialisation de la carte 3Com

Nous avons vu au chapitre 15 que les pilotes de carte réseau font appel à une fonction de détection et d'initialisation. Nous avons vu également qu'il est fait appel à la fonction `init()` dans le corps de la fonction `register_netdevice()` ainsi qu'à la fonction de destruction `destructor()` spécifiques à la carte réseau. La carte 3Com 501 ne possède pas de fonction de destruction spécifique. Nous avons vu, à propos de `isa_probes[]` et de `dev_3c501`, que la fonction de détection et d'initialisation s'appelle `el1_probe()` dans le cas de la carte 3Com501 (que le pilote de celle-ci soit lié statiquement au noyau ou modularisé).

17.2.1 Fonction principale

La fonction `el1_probe()` est définie dans le fichier `linux/drivers/net/3c501.c`: Code Linux 2.6.10

```
139 /*
140  *      Le code de detection [de la chaudiere].
141  */
142
143 static int io=0x280;
144 static int irq=5;
145 static int mem_start;
146
147 /**
148  * el1_probe :          -       detecte une 3c501
149  * @dev : la structure de peripherique passee pour la detection.
150  *
151  * Ceci peut etre appele depuis deux endroits. La couche reseau sondera en utilisant
152  * une structure de peripherique passee avec les informations de sondage completees. Pour un
153  * peripherique modularise, nous utilisons #init_module pour renseigner notre propre structure
154  * et pour la detecter.
155  *
156  * Renvoie 0 en cas de succes, ENXIO s'il n'est pas demande de sonder et ENODEV s'il est
157  * demande de sonder et qu'on echoue a trouver quelque chose.
158  */
159
160 struct net_device * __init el1_probe(int unit)
161 {
162         struct net_device *dev = alloc_etherdev(sizeof(struct net_local));
163         static unsigned ports[] = { 0x280, 0x300, 0};
164         unsigned *port;
165         int err = 0;
166
167         if (!dev)
168                 return ERR_PTR(-ENOMEM);
169
170         if (unit >= 0) {
```

```
171                 sprintf(dev->name, "eth%d", unit);
172                 netdev_boot_setup_check(dev);
173                 io = dev->base_addr;
174                 irq = dev->irq;
175                 mem_start = dev->mem_start & 7;
176         }
177
178         SET_MODULE_OWNER(dev);
179
180         if (io > 0x1ff) {       /* Verifie un seul emplacement specifie. */
181                 err = el1_probe1(dev, io);
182         } else if (io != 0) {
183                 err = -ENXIO;           /* Ne pas sonder du tout. */
184         } else {
185                 for (port = ports; *port && el1_probe1(dev, *port); port++)
186                         ;
187                 if (!*port)
188                         err = -ENODEV;
189         }
190         if (err)
191                 goto out;
192         err = register_netdev(dev);
193         if (err)
194                 goto out1;
195         return dev;
196 out1:
197         release_region(dev->base_addr, EL1_IO_EXTENT);
198 out:
199         free_netdev(dev);
200         return ERR_PTR(err);
201 }
```

Autrement dit :

- On déclare une adresse de descripteur de périphérique réseau, que l'on essaie d'instantier pour une carte Ethernet. Si on n'y parvient pas, on s'arrête en pointant l'opposé du code d'erreur `ENOMEM`.

Code Linux 2.6.10

La macro générale `ERR_PTR()` est définie dans le fichier `linux/include/linux/err.h`:

```
8  /*
9   * Les pointeurs noyau ont de l'information redondante, aussi pouvons-nous utiliser un
10  * schema ou nous pouvons renvoyer soit un code d'erreur, soit un pointeur
11  * dentry avec la meme valeur renvoyee.
12  *
13  * Ceci doit etre une chose pour l'architecture, de permettre des decisions
14  * pointeur et erreur differentes.
15  */
16 static inline void *ERR_PTR(long error)
17 {
18         return (void *) error;
19 }
```

- On définit le tableau des numéros de port d'entrée-sortie du microprocesseur relié à la carte pouvant être utilisés pour une carte 3Com 501.
- On déclare un numéro de port d'entrée-sortie et un code d'erreur.
- Si l'unité passée en argument est positive ou nulle (rappelons qu'elle était de -1 dans le cas du module et entre 0 et 7 dans le cas des huit cartes Ethernet de départ) :
  - On renseigne le champ nom du descripteur de périphérique.

  - On vérifie les paramètres de démarrage, grâce à la fonction `netdev_boot_setup_check()` étudiée ci-dessous.
  - On détermine l'adresse d'entrée-sortie de base, le numéro d'interruption et le début de mémoire vive à utiliser pour les tampons à partir du descripteur de périphérique.

- On se rend maître du module dans lequel se trouve le pilote.

  La macro `SET_MODULE_OWNER()` est définie dans le fichier `linux/include/linux/netdevice.h`: Code Linux 2.6.10

```
503 #define SET_MODULE_OWNER(dev) do { } while (0)
504 /* Positionne la reference du peripherique physique sysfs pour le peripherique logique
       de reseau.
505  * Positionner avant l'enregistrement causera un symlink pendant l'initialisation.
506  */
```

- Si l'adresse de base est strictement supérieure à `1FFh`, on essaie de détecter la présence physique de la carte au seul emplacement spécifié, grâce à la fonction `el1_probe1()` étudiée ci-après.
- Si l'adresse de base est comprise entre 1 et `1FFh`, ceci signifie qu'il ne faut pas sonder. On libère donc le descripteur de périphérique et on renvoie l'opposé du code d'erreur `ENXIO`.

  La fonction `free_netdev()` sera étudiée ci-dessous.
- Si l'adresse de base est nulle, on essaie de détecter la présence physique de la carte pour les valeurs du tableau `ports[]`. En cas de succès, on renvoie 0. Si cela échoue pour toutes ces valeurs, on libère le descripteur de périphérique et on renvoie l'opposé du code d'erreur `ENODEV`.
- On essaie d'enregistrer le périphérique. Si on n'y parvient pas, on libère les numéros de port d'entrée-sortie du microprocesseur occupés par cette carte, on libère le descripteur de périphérique et on renvoie le code d'erreur fourni par la fonction `register_netdev()`.

  La constante symbolique `EL1_IO_EXTENT`, spécifiant le nombre de ports d'entrée-sortie consécutifs nécessaire pour une carte 3Com 501, est définie dans le fichier `linux/drivers/net/3c501.h`: Code Linux 2.6.0

```
18 #define EL1_IO_EXTENT   16
```

- Sinon on renvoie l'adresse du descripteur de périphérique.

17.2.2 Établissement des paramètres au démarrage

La fonction `netdev_boot_setup_check()` est définie dans le fichier `linux/net/core/dev.c`: Code Linux 2.6.10

```
382 /**
383  *      netdev_boot_setup_check - verifie les parametres au moment du demarrage
384  *      @dev : le peripherique reseau
385  *
386  *      Verifie les parametres au moment du demarrage pour ce peripherique.
387  *      Les parametres trouves sont positionnes pour que le peripherique les utilise
388  *      plus tard lors de la detection du peripherique.
389  *      Renvoie 0 si aucun parametre trouve, 1 s'il y en a.
390  */
391 int netdev_boot_setup_check(struct net_device *dev)
392 {
393         struct netdev_boot_setup *s = dev_boot_setup;
394         int i;
395
396         for (i = 0; i < NETDEV_BOOT_SETUP_MAX; i++) {
397                 if (s[i].name[0] != '\0' && s[i].name[0] != ' ' &&
```

```
398                     !strncmp(dev->name, s[i].name, strlen(s[i].name))) {
399                         dev->irq        = s[i].map.irq;
400                         dev->base_addr  = s[i].map.base_addr;
401                         dev->mem_start  = s[i].map.mem_start;
402                         dev->mem_end    = s[i].map.mem_end;
403                         return 1;
404                 }
405         }
406         return 0;
407 }
```

Autrement dit :

- On déclare un tableau de structures de paramètres de démarrage, que l'on initialise avec `dev_boot_setup[]`.
- On cherche si le nom du périphérique réseau passé en argument apparaît dans le tableau des paramétrages. Si c'est le cas, on renseigne les champs du descripteur de périphérique passé en argument avec ces paramètres et on renvoie 1.
- Si ce nom n'apparaît pas, on renvoie 0.

17.2.3 Libération d'un descripteur de périphérique réseau

Code Linux 2.6.10

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

```
2960 /**
2961  *      free_netdev - libere un peripherique reseau
2962  *      @dev : peripherique
2963  *
2964  *      Cette fonction effectue la derniere etape de destruction d'une interface de
2965  *      peripherique allouee. La reference a l'objet peripherique est relachee.
2966  *      S'il s'agit de la derniere reference, il est libere.
2967  */
2968 void free_netdev(struct net_device *dev)
2969 {
2970 #ifdef CONFIG_SYSFS
2971         /*  Compatibilite avec la manipulation des erreurs dans le peripherique */
2972         if (dev->reg_state == NETREG_UNINITIALIZED) {
2973                 kfree((char *)dev - dev->padded);
2974                 return;
2975         }
2976
2977         BUG_ON(dev->reg_state != NETREG_UNREGISTERED);
2978         dev->reg_state = NETREG_RELEASED;
2979
2980         /* sera libere via le relachement de la classe */
2981         class_device_put(&dev->class_dev);
2982 #else
2983         kfree((char *)dev - dev->padded);
2984 #endif
2985 }
```

17.2.4 Vérification de la présence physique de la carte

Code Linux 2.6.10

La fonction `el1_probe1()` est définie dans le fichier `linux/drivers/net/3c501.c` :

```
203 /**
204  *      el1_probe1 :
205  *      @dev : la structure de peripherique a utiliser
206  *      @ioaddr : une adresse d'E/S a tester.
207  *
```

```
208  *      La detection elle-meme. Ceci est itere par #el1_probe afin de
209  *      verifier tous les emplacements de peripheriques applicables.
210  *
211  *      Renvoie 0 en cas de succes, en quel cas le peripherique est active,
212  *      EAGAIN si le IRQ est utilise par un autre peripherique et ENODEV si la
213  *      carte n'est pas detectee.
214  */
215
216 static int __init el1_probe1(struct net_device *dev, int ioaddr)
217 {
218         struct net_local *lp;
219         const char *mname;              /* Nom du vendeur */
220         unsigned char station_addr[6];
221         int autoirq = 0;
222         int i;
223
224         /*
225          *      Reserve les ressources d'E/S exclusivement pour ce peripherique
226          */
227
228         if (!request_region(ioaddr, EL1_IO_EXTENT, DRV_NAME))
229                 return -ENODEV;
230
231         /*
232          *      Lit les donnees PROM de l'adresse de la station a partir du port special.
233          */
234
235         for (i = 0; i < 6; i++)
236         {
237                 outw(i, ioaddr + EL1_DATAPTR);
238                 station_addr[i] = inb(ioaddr + EL1_SAPROM);
239         }
240         /*
241          *      Verifie les trois premiers octets de l'adresse de la station a la
242          *      recherche du prefixe 3Com ou du prefixe Sager NP943.
243          */
244
245         if (station_addr[0] == 0x02  &&  station_addr[1] == 0x60
246                 && station_addr[2] == 0x8c)
247         {
248                 mname = "3c501";
249         } else if (station_addr[0] == 0x00  &&  station_addr[1] == 0x80
250         && station_addr[2] == 0xC8)
251         {
252                 mname = "NP943";
253         }
254         else {
255                 release_region(ioaddr, EL1_IO_EXTENT);
256                 return -ENODEV;
257         }
258
259         /*
260          *      Nous auto-IRQ en fermant la ligne d'interruption et en la positionnant
261          *      a la valeur haute.
262          */
263
264         dev->irq = irq;
265
266         if (dev->irq < 2)
267         {
268                 unsigned long irq_mask;
269
```

```
270                 irq_mask = probe_irq_on();
271                 inb(RX_STATUS);         /* Efface les interruptions en souffrance. */
272                 inb(TX_STATUS);
273                 outb(AX_LOOP + 1, AX_CMD);
274
275                 outb(0x00, AX_CMD);
276
277                 mdelay(20);
278                 autoirq = probe_irq_off(irq_mask);
279
280                 if (autoirq == 0)
281                 {
282                         printk(KERN_WARNING "%s probe at %#x failed to detect IRQ line.\n",
283                                 mname, ioaddr);
284                         release_region(ioaddr, EL1_IO_EXTENT);
285                         return -EAGAIN;
286                 }
287         }
288
289         outb(AX_RESET+AX_LOOP, AX_CMD);                 /* Mode loopback. */
290         dev->base_addr = ioaddr;
291         memcpy(dev->dev_addr, station_addr, ETH_ALEN);
292
293         if (mem_start & 0xf)
294                 el_debug = mem_start & 0x7;
295         if (autoirq)
296                 dev->irq = autoirq;
297
298         printk(KERN_INFO "%s: %s EtherLink at %#lx, using %sIRQ %d.\n",
               dev->name, mname, dev->base_addr,
299                         autoirq ? "auto":"assigned ", dev->irq);
300
301 #ifdef CONFIG_IP_MULTICAST
302         printk(KERN_WARNING "WARNING: Use of the 3c501 in a multicast kernel is
               NOT recommended.\n");
303 #endif
304
305         if (el_debug)
306                 printk(KERN_DEBUG "%s", version);
307
308         memset(dev->priv, 0, sizeof(struct net_local));
309         lp = netdev_priv(dev);
310         spin_lock_init(&lp->lock);
311
312         /*
313          *      Les entrees specifiques a EL1 dans la structure de peripherique.
314          */
315
316         dev->open = &el_open;
317         dev->hard_start_xmit = &el_start_xmit;
318         dev->tx_timeout = &el_timeout;
319         dev->watchdog_timeo = HZ;
320         dev->stop = &el1_close;
321         dev->get_stats = &el1_get_stats;
322         dev->set_multicast_list = &set_multicast_list;
323         dev->ethtool_ops = &netdev_ethtool_ops;
324         return 0;
325 }
```

Autrement dit :

– On déclare une entité, du type `struct net_local`, pour les informations spécifiques à ce type de cartes.

Le type dépend du type de cartes. Il est défini dans le fichier `linux/drivers/net/-3c501.h` pour la carte qui nous intéresse: Code Linux 2.6.10

```
26 /*
27  *      Informations specifiques a la carte dans dev->priv.
28  */
29
30 struct net_local
31 {
32         struct net_device_stats stats;
33         int             tx_pkt_start;   /* La longueur du paquet emis en cours. */
34         int             collisions;     /* Collisions en emission dans ce paquet */
35         int             loading;        /* Tampon d'eclaboussure qui charge les
                                              collisions */
36         int             txing;          /* Vrai si la carte est en mode d'emission */
37         spinlock_t      lock;           /* Verrou de serialisation */
38 };
```

- On déclare des variables pour le nom du vendeur et l'adresse MAC de la station.
- Si la plage d'adresses d'entrée-sortie passée en argument chevauche une plage utilisée par un autre pilote, on renvoie l'opposé du code d'erreur `ENODEV`.

  Les détails de la macro générale `request_region()` ne nous intéresserons pas dans cet ouvrage.

  La constante symbolique `DRV_NAME` est définie, pour la carte qui nous intéresse, dans le fichier `linux/drivers/net/3c501.h`: Code Linux 2.6.10

```
102 #define DRV_NAME        "3c501"
```

- On lit l'adresse MAC de la station à partir de la mémoire PROM de la carte.
- On vérifie les trois premiers caractères pour déterminer le préfixe du vendeur (deux possibilités) et initialiser le nom du vendeur de la carte. S'il ne s'agit pas de l'un de ces deux vendeurs, on libère la plage de ports d'entrée-sortie et on renvoie le code d'erreur `ENODEV`.
- On essaie d'initialiser l'interruption matérielle de la carte. Si on n'y parvient pas, on libère la plage de ports d'entrée-sortie et on renvoie l'opposé du code d'erreur `EAGAIN`.
- On renseigne les champ adresse d'entrée-sortie de base et adresse MAC de la station du descripteur de périphérique passé en argument (lignes 290 et 291).
- On change éventuellement la valeur de la variable `el_debug`, celle-ci étant définie dans le fichier `linux/drivers/net/3c501.h`: Code Linux 2.6.10

```
20 #ifndef EL_DEBUG
21 #define EL_DEBUG  0     /* utiliser 0 pour production, 1 pour developpement, >2 pour
                              debogage */
22 #endif                 /* Tout ce qui est dessus de 5 est mort ! */
23 #define debug el_debug
24 static int el_debug = EL_DEBUG;
```

- Si on est parvenu à déterminer le numéro d'interruption, on renseigne le champ `irq` du descripteur de périphérique passé en argument.
- On affiche un message noyau indiquant que la carte a été détectée.
- Si les options de compilation du noyau comprennent la multidiffusion, un message noyau est affiché, indiquant que ce n'est pas une bonne idée d'utiliser cette carte pour la multidiffusion.
- Si la valeur de débogage est non nulle, on affiche un message noyau spécifiant la version.

  La constante `version` est définie dans le fichier `linux/drivers/net/3c501.h`: Code Linux 2.6.10

```
103 #define DRV_VERSION     "2002/10/09"
```

```
104
105
106 static const char version[] =
107         DRV_NAME ".c: " DRV_VERSION " Alan Cox (alan@redhat.com).\n";
```

- On initialise la structure privée à zéro.
- On verrouille cette structure.
- On renseigne les champs du descripteur de périphérique spécifiques à la carte : les fonctions `el_open()` d'ouverture (en fait d'activation) du périphérique, `el_start_xmit()` de transmission des paquets, `el1_close()` de fermeture (désactivation) du périphérique, `el_timeout()` d'action à entreprendre lorsque le délai est écoulé, `el1_get_stats()` de récupération des informations statistiques et `set_multicast_list()` de mise en place de la diffusion restreinte ; le délai est initialisé à `HZ`.

  Ces fonctions spécifiques seront étudiées au fur et à mesure des besoins.
- On renseigne le champ de l'ensemble des opérations liées à Ethernet du descripteur de périphérique et on renvoie 0.

  L'ensemble d'opérations `netdev_ethtool_ops` propre à cette carte est défini dans le fichier `linux/drivers/net/3c501.c` :

Code Linux 2.6.10

```
885 static struct ethtool_ops netdev_ethtool_ops = {
886         .get_drvinfo            = netdev_get_drvinfo,
887         .get_msglevel           = netdev_get_msglevel,
888         .set_msglevel           = netdev_set_msglevel,
889 };
```

Les fonctions génériques `netdev_get_drvinfo()`, `netdev_get_msglevel()` et `netdev_set_msglevel()` devront être étudiées au fur et à mesure des besoins. En fait nous n'en aurons pas besoin dans cet ouvrage.