Chapitre 14

Les structures de données Linux pour le pilote de périphérique réseau

La première action à entreprendre avant de pouvoir utiliser le réseau est de détecter la carte réseau (éventuellement les cartes réseau) de l'ordinateur, ce qui est un problème de gestion de périphérique, comme détecter un disque dur, une carte graphique ou une clé USB.

Unix distingue traditionnellement deux types seulement de fichiers de périphériques: les fichiers caractère et les fichiers bloc. Linux préfère introduire un troisième type de fichiers de périphériques pour les réseaux. Il serait possible de considérer un périphérique réseau comme un périphérique bloc mais avec une certaine gymnastique inutile.

Les structures de données fondamentales concernant les pilotes de périphériques réseau sous Linux sont au nombre de deux:

- La carte réseau est représentée par un descripteur, de type `struct net_device`. Comme pour tout périphérique, la carte réseau sera vue par le système d'exploitation comme un fichier.
- L'en-tête de la trame, appelé **en-tête matériel** ou **en-tête physique**, est placée dans un cache afin d'éviter de le recomposer sans cesse.

Nous terminerons par l'ordre des octets utilisé sur les réseaux.

14.1 Descripteur d'interface de périphérique réseau

14.1.1 Philosophie Linux des fichiers de périphérique réseau

Comme nous l'avons dit dans l'introduction, Linux préfère définir un troisième type de fichiers de périphériques pour les réseaux. Ce type présente quelques différences par rapport aux fichiers de périphériques caractère ou bloc :

- Le rôle d'un périphérique réseau dans le système est semblable à celui d'un périphérique bloc monté. Un périphérique bloc enregistre ses caractéristiques dans le tableau `blk_array[]` puis reçoit et transmet des blocs sur requête, au moyen de la fonction `request()`. De même un périphérique réseau doit s'enregistrer avant que des paquets soient échangés avec le monde extérieur. Nous verrons au chapitre 15 comment on enregistre un périphérique réseau.
- Sous Linux, un pilote de carte réseau peut être lié statiquement au noyau ou chargé comme module, comme pour tout pilote de périphérique. Cependant, contrairement aux périphériques caractère et bloc, le pilote de carte réseau ne se déclare pas de la même façon suivant qu'il est lié statiquement ou modularisé.
- Un périphérique réseau n'est pas repéré par un nombre majeur et un nombre mineur, comme le sont les périphériques caractère et bloc, mais par un **index** comme nous le verrons.
- Chaque pilote correspond à un descripteur de type `struct net_device` inséré dans une liste chaînée globale de périphériques réseau, appelée `dev_base` et déclarée dans le fichier `linux/net/core/dev.c` :

Code Linux 2.6.10

```
185 struct net_device *dev_base;
```

- Un disque possède un fichier spécial dans le répertoire `/dev` mais les périphériques réseau ne possèdent pas de telle entrée. Il n'existe donc, par exemple, aucun périphérique réseau tel que `/dev/eth0` ou `/dev/atm1`.
- Les opérations habituelles sur les fichiers (en particulier les appels système de lecture et d'écriture simples `read()` et `write()`) n'ont pas de sens pour les périphériques réseau (à part les sockets locales). Ceux-ci utilisent un ensemble différent d'opérations, comme indiqué au début du fichier `linux/drivers/net/net_init.c` :

Code Linux 2.6.10

```
55 /* Les peripheriques reseau existant a l'heure actuelle uniquement dans l'espace de
56    noms des sockets, ces entrees ne sont pas utilisees. Les seules qui ont un sens sont :
57     open        demarre la carte Ethernet
58     close       arrete la carte Ethernet
59     ioctl       Pour obtenir des statistiques et peut-etre positionner le port de
      l'interface (AUI, BNC, etc.)
60    On peut aussi imaginer obtenir des paquets bruts en utilisant
61     read & write
62    mais il vaut probalement mieux les manipuler par une socket de paquet brut.
63
64    Etant donne que presque toutes ces fonctions sont manipulees dans le schema
65    socket actuel, mettre les peripheriques carte Etherner dans /dev/ semble sans interet.
66
67    [Effacer tous les supports pour les peripheriques reseau de /dev. Lorsque quelqu'un
68     envoie des flux, nous les recuperons par magie, mais sinon ils ne sont pas
69         necessaires et constituent une perte d'espace]
70 */
```

Les descripteurs de périphérique réseau ne représentent pas tous une carte physique. Il existe également des périphériques comme le périphérique réseau en boucle `loopback` qui offrent une fonctionnalité de réseau logique.

14.1.2 Définition du type

Un périphérique réseau apparaît à Linux sous la forme d'un descripteur d'interface constituée d'un certain nombre d'attributs et de méthodes (ou fonctions de service). Il s'agit d'une classe au sens de la programmation orientée objet, mais implémentée en C.

Le descripteur de l'interface d'un périphérique réseau est une entité du type `struct net_device`, défini dans le fichier en-tête `linux/include/linux/netdevice.h`. Commençons par en reproduire la définition que nous commenterons ensuite :

Code Linux 2.6.10

```
6    *              Definitions pour les interfaces de routines.
7    *
8    * Version :    @(#)dev.h       1.0.10  08/12/93
9    *
10   * Auteurs :    Ross Biro, <bir7@leland.Stanford.Edu>
11   *              Fred N. van Kempen, <waltje@uWalt.NL.Mugnet.ORG>
12   *              Corey Minyard <wf-rch!minyard@relay.EU.net>
13   *              Donald J. Becker, <becker@cesdis.gsfc.nasa.gov>
14   *              Alan Cox, <Alan.Cox@linux.org>
15   *              Bjorn Ekwall. <bj0rn@blox.se>
16   *              Pekka Riikonen <priikone@poseidon.pspt.fi>
[...]
23   *              Deplace a /usr/include/linux pour NET3
24   */
25  #ifndef _LINUX_NETDEVICE_H
26  #define _LINUX_NETDEVICE_H
[...]
254 /*
255  *      La structure PERIPHERIQUE.
256  *      En fait cette structure entiere est une grosse erreur. Elle melange des donnees
257  *      d'E/S avec des donnees strictement de "haut niveau", et elle doit connaitre
258  *      presque toutes les structures de donnees utilisees dans le module INET.
259  *
260  *      A RECTIFIER : nettoyer struct net_device de facon a ce que les infos sur
261  *      le protocole reseau soient deplacees.
262  */
263
264 struct net_device
265 {
266
267         /*
268          * Ceci est le premier champ de la partie "visible" de cette structure
269          * (c'est-a-dire vue par les utilisateurs dans le fichier "Space.c"). C'est le nom
270          * de l'interface.
271          */
272         char                    name[IFNAMSIZ];
273
274         /*
275          *      Champs specifiques aux E/S
276          *      A RECTIFIER : Fusionner ceci et struct ifmap en une seule
277          */
278         unsigned long           mem_end;        /* Fin de la memoire partagee    */
279         unsigned long           mem_start;      /* Debut de la memoire partagee  */
280         unsigned long           base_addr;      /* adresse d'E/S du peripherique */
281         unsigned int            irq;            /* numero d'IRQ du peripherique  */
282
283         /*
284          *      Certains materiels necessitent egalement ces champs, mais ils ne font
285          *      pas partie de l'ensemble usuel specifie dans Space.c.
286          */
287
288         unsigned char           if_port;        /* AUI, TP,... selectionnable */
289         unsigned char           dma;            /* canal DMA                  */
```

```
290
291         unsigned long           state;
292
293         struct net_device       *next;
294
295         /* La fonction d'initialisation du peripherique. Appelee seulement une fois. */
296         int                     (*init)(struct net_device *dev);
297
298         /* ------- Les champs preinitialises dans Space.c se terminent ici --------- */
299
300         struct net_device       *next_sched;
301
302         /* Index d'interface. Identificateur unique de peripherique                  */
303         int                     ifindex;
304         int                     iflink;
305
306
307         struct net_device_stats* (*get_stats)(struct net_device *dev);
308         struct iw_statistics*   (*get_wireless_stats)(struct net_device *dev);
309
310         /* Liste de fonctions pour gerer les extensions sans fil (au lieu de).
311          * Voir <net/iw_handler.h> pour les details. Jean II */
312         const struct iw_handler_def *   wireless_handlers;
313         /* Donnees de l'instance gerees par le coeur des Extensions Sans Fil. */
314         struct iw_public_data * wireless_data;
315
316         struct ethtool_ops *ethtool_ops;
317
318         /*
319          * Ceci marque la fin de la partie "visible" de la structure. Tous
320          * les champs ci-apres sont internes au systeme, et peuvent etre changes a
321          * volonte (lire : peuvent etre enleves a volonte).
322          */
323
324         /* Ceux-ci peuvent etre necessaire pour du code futur de debranchement du reseau
               a distance. */
325         unsigned long           trans_start;    /* Heure (en jiffies) de la derniere Tx */
326         unsigned long           last_rx;        /* Heure de la derniere Rx              */
327
328         unsigned short          flags;  /* Drapeaux de l'interface (a la BSD)   */
329         unsigned short          gflags;
330         unsigned short          priv_flags; /* Comme 'flags' mais dans l'espace utilisateur
                                                   invisible. */
331         unsigned short          unused_alignment_fixer; /* Puisque nous avons besoin de
                                                               priv_flags,
332                                                          * et que nous voulons etre alignes
                                                             sur 32 bits.
333                                                          */
334
335         unsigned                mtu;    /* valeur MTU de l'interface    */
336         unsigned short          type;   /* type materiel de l'interface */
337         unsigned short          hard_header_len;        /* taille de l'en-tete materiel */
338         void                    *priv;  /* pointeur sur des donnees privees             */
339
340         struct net_device       *master; /* Pointeur sur le peripherique maitre du groupe
341                                           * dont ce peripherique est membre.
342                                           */
343
344         /* Info sur l'adresse de l'interface. */
345         unsigned char           broadcast[MAX_ADDR_LEN];        /* adresse materielle de
                                                                       diffusion generale */
346         unsigned char           dev_addr[MAX_ADDR_LEN]; /* adresse materielle         */
```

```
347         unsigned char           addr_len;       /* longueur de l'adresse physique       */
348
349         struct dev_mc_list      *mc_list;       /* Adresses mac de multidiffusion       */
350         int                     mc_count;       /* Nombre de multidiffusions installees */
351         int                     promiscuity;
352         int                     allmulti;
353
354         int                     watchdog_timeo;
355         struct timer_list       watchdog_timer;
356
357         /* Pointeurs sur des protocoles specifiques */
358
359         void                    *atalk_ptr;     /* Lien AppleTalk               */
360         void                    *ip_ptr;        /* Donnees specifiques a IPv4   */
361         void                    *dn_ptr;        /* Donnees specifiques a DECnet */
362         void                    *ip6_ptr;       /* Donnees specifiques a IPv6   */
363         void                    *ec_ptr;        /* Donnees specifiques a Econet */
364         void                    *ax25_ptr;      /* Donnees specifiques a AX.25  */
365
366         struct list_head        poll_list;      /* Lien a la liste d'election   */
367         int                     quota;
368         int                     weight;
369
370         struct Qdisc            *qdisc;
371         struct Qdisc            *qdisc_sleeping;
372         struct Qdisc            *qdisc_ingress;
373         struct list_head        qdisc_list;
374         unsigned long           tx_queue_len;   /* Nombre max de trames permis par file
                                                       d'attente */
375
376         /* Synchronisateur de chemin ingress */
377         spinlock_t              ingress_lock;
378         /* Synchronisateur de hard_start_xmit */
379         spinlock_t              xmit_lock;
380         /* id du microprocesseur entre a hard_start_xmit ou -1
381            s'il n'y en a pas.
382          */
383         int                     xmit_lock_owner;
384         /* verrou de file d'attente du peripherique */
385         spinlock_t              queue_lock;
386         /* Nombre de references a ce peripherique */
387         atomic_t                refcnt;
388         /* enregistrement/retrait retarde */
389         struct list_head        todo_list;
390         /* chaine de hachage des noms de peripherique */
391         struct hlist_node       name_hlist;
392         /* chaine de hachage des index de peripherique */
393         struct hlist_node       index_hlist;
394
395         /* etat de la machine d'enregistrement/retrait */
396         enum { NETREG_UNINITIALIZED=0,
397                NETREG_REGISTERING,      /* register_netdevice appele          */
398                NETREG_REGISTERED,       /* enregistrement complete a realiser */
399                NETREG_UNREGISTERING,    /* unregister_netdevice appele        */
400                NETREG_UNREGISTERED,     /* retrait complete a realiser        */
401                NETREG_RELEASED,         /* free_netdev appele                 */
402         } reg_state;
403
404         /* Caracteristiques du peripherique reseau */
405         int                     features;
406 #define NETIF_F_SG              1       /* Scatter/gather IO. */
407 #define NETIF_F_IP_CSUM         2       /* Doit controler la somme seulement pour TCP/UDP
```

```
                                        sur IPv4. */
408 #define NETIF_F_NO_CSUM         4       /* Ne demande pas de controle de la somme. Par
                                        exemple loopack. */
409 #define NETIF_F_HW_CSUM         8       /* Doit verifier la sommme pour tous les paquets. */
410 #define NETIF_F_HIGHDMA         32      /* Doit utiliser DMA pour la memoire haute. */
411 #define NETIF_F_FRAGLIST        64      /* Scatter/gather IO. */
412 #define NETIF_F_HW_VLAN_TX      128     /* Acceleration materielle pour une transmission
                                        VLAN */
413 #define NETIF_F_HW_VLAN_RX      256     /* Acceleration materielle pour une reception
                                        VLAN */
414 #define NETIF_F_HW_VLAN_FILTER  512     /* Filtrage des receptions sur VLAN */
415 #define NETIF_F_VLAN_CHALLENGED 1024    /* Le peripherique ne peut pas manipuler les
                                        paquets VLAN */
416 #define NETIF_F_TSO             2048    /* Peut se charger de la segmentation TCP/IP */
417 #define NETIF_F_LLTX            4096    /* LockLess TX */
418
419         /* Appele apres qu'un peripherique soit detache du reseau. */
420         void                    (*uninit)(struct net_device *dev);
421         /* Appele apres que la derniere reference utilisateur ait disparue. */
422         void                    (*destructor)(struct net_device *dev);
423
424         /* Pointeurs vers les routines de service de l'interface.      */
425         int                     (*open)(struct net_device *dev);
426         int                     (*stop)(struct net_device *dev);
427         int                     (*hard_start_xmit) (struct sk_buff *skb,
428                                                     struct net_device *dev);
429 #define HAVE_NETDEV_POLL
430         int                     (*poll) (struct net_device *dev, int *quota);
431         int                     (*hard_header) (struct sk_buff *skb,
432                                                 struct net_device *dev,
433                                                 unsigned short type,
434                                                 void *daddr,
435                                                 void *saddr,
436                                                 unsigned len);
437         int                     (*rebuild_header)(struct sk_buff *skb);
438 #define HAVE_MULTICAST
439         void                    (*set_multicast_list)(struct net_device *dev);
440 #define HAVE_SET_MAC_ADDR
441         int                     (*set_mac_address)(struct net_device *dev,
442                                                    void *addr);
443 #define HAVE_PRIVATE_IOCTL
444         int                     (*do_ioctl)(struct net_device *dev,
445                                             struct ifreq *ifr, int cmd);
446 #define HAVE_SET_CONFIG
447         int                     (*set_config)(struct net_device *dev,
448                                               struct ifmap *map);
449 #define HAVE_HEADER_CACHE
450         int                     (*hard_header_cache)(struct neighbour *neigh,
451                                                      struct hh_cache *hh);
452         void                    (*header_cache_update)(struct hh_cache *hh,
453                                                        struct net_device *dev,
454                                                        unsigned char *  haddr);
455 #define HAVE_CHANGE_MTU
456         int                     (*change_mtu)(struct net_device *dev, int new_mtu);
457
458 #define HAVE_TX_TIMEOUT
459         void                    (*tx_timeout) (struct net_device *dev);
460
461         void                    (*vlan_rx_register)(struct net_device *dev,
462                                                     struct vlan_group *grp);
463         void                    (*vlan_rx_add_vid)(struct net_device *dev,
464                                                    unsigned short vid);
```

```
465         void                    (*vlan_rx_kill_vid)(struct net_device *dev,
466                                                     unsigned short vid);
467
468         int                     (*hard_header_parse)(struct sk_buff *skb,
469                                                      unsigned char *haddr);
470         int                     (*neigh_setup)(struct net_device *dev, struct neigh_parms *);
471         int                     (*accept_fastpath)(struct net_device *, struct dst_entry*);
472 #ifdef CONFIG_NETPOLL
473         int                     netpoll_rx;
474 #endif
475 #ifdef CONFIG_NET_POLL_CONTROLLER
476         void                    (*poll_controller)(struct net_device *dev);
477 #endif
478
479         /* Chose relative aux ponts */
480         struct net_bridge_port  *br_port;
481
482 #ifdef CONFIG_NET_DIVERT
483         /* Ceci sera initialise par la routine init de chaque type d'interface */
484         struct divert_blk       *divert;
485 #endif /* CONFIG_NET_DIVERT */
486
487         /* entree classe/reseau/nom */
488         struct class_device     class_dev;
489         /* Combien de [caracteres de] remplissage ont ete ajoutes par alloc_netdev() */
490         int padded;
491 };
```

14.1.3 Partie visible

Les membres de la structure `net_device` peuvent être répartis en deux groupes : la **partie visible** par un utilisateur et la partie invisible par un utilisateur, mais évidemment visible par le noyau. La partie visible de la structure est constituée des membres qui peuvent être assignés statiquement ; toutes les structures définies dans le fichier `drivers/net/Space.c` sont initialisés de cette façon. Les autres membres ne sont pas initialisés au moment de la compilation : certains le sont au moment de l'initialisation et d'autres par le périphérique lui-même à tout moment (par exemple les statistiques sur le nombre de paquets reçus).

14.1.3.1 Nom du périphérique réseau

Sous Linux, tout périphérique réseau dispose d'un nom sans équivoque. Le tableau ci-dessous montre quelques-uns des noms actuellement utilisés :

Nom	Type de périphérique réseau
eth	Ethernet
tr	Token Ring
atm	ATM
sl	SLIP
ppp	PPP
plip	PLIP (*Parallel Line Interface Protocol*)
tunl	Tunnel IPIP
isdn	ISDN (NUMERIS en France)
dummy	Périphérique factice
lo	Périphérique en boucle (*loopback*)

Les périphériques de même type sont généralement numérotés à la file de 0 à n, par ordre

croissant, par exemple `isdn0`, `isdn1`, etc. Certains périphériques, comme le périphérique en boucle (`lo`), n'apparaissent qu'une fois et comportent par conséquent des noms fixes.

Du fait que Linux est très étroitement lié à Ethernet et que quelques fonctions du noyau en simplifient l'utilisation, certaines cartes non Ethernet se servent également de la catégorie `eth`*n* grâce à des adaptateurs simulant Ethernet.

Le schéma de désignation des périphériques réseau offre quelques avantages. Lors de la conception d'applications et l'établissement de fichiers de commandes de configuration, il est plus facile de solliciter les périphériques réseau sans avoir à connaître quel est leur fabricant et quels sont leurs paramètres (numéro d'interruption, numéro de port).

Le nom d'un périphérique réseau constitue le champ `name` du descripteur.

Le nombre maximum de caractères pour ce nom est repéré par la constante symbolique `IFNAMESIZ` (pour *Interface PHysique NAME SIZe*), définie dans le fichier en-tête `linux/include/linux/if.h`:

Code Linux 2.6.10

```
26 #define IFNAMSIZ        16
```

14.1.3.2 Champs spécifiques au matériel

Les attributs liés aux entrées-sorties correspondent aux champs :

- `mem_end` et `mem_start` de fin et de début de la mémoire partagée entre adaptateur et noyau. La zone de mémoire `[mem_start, mem_end]` désigne le tampon destiné aux paquets à envoyer ou à réceptionner.

  Ces valeurs peuvent être spécifiées sur la ligne de commande du noyau lors du démarrage du système et peuvent être retrouvées grâce à la commande `ifconfig`. On s'arrange, par convention, pour que `mem_end - mem_start` soit égal à la quantité de mémoire vive disponible.

- `base_addr` correspondant au numéro de port d'entrée-sortie du microprocesseur relié au périphérique. En général une carte réseau utilise plusieurs ports d'entrée-sortie du microprocesseur mais seul le premier est important puisque les autres sont toujours les suivants, leur nombre, quant à lui, dépendant de la carte.

  Cette valeur est en général définie au moment de la détection de la carte. La commande `ifconfig` peut également être utilisée pour afficher ou modifier cette valeur.

- `irq` correspondant au numéro d'interruption du périphérique.

  Cette valeur est en général définie au moment de la recherche du périphérique dans la routine de détection du pilote ou par une indication explicite lors du chargement du module ou du démarrage du noyau. La valeur `dev->irq` est affichée par `ifconfig` lorsque les interfaces sont listées. Elle peut être modifiée par cette commande.

- `if_port` permet de repérer le port du périphérique si celui-ci en possède plusieurs. C'est souvent le cas avec des cartes qui possèdent à la fois une connexion coaxiale (`IF_PORT_10BASE2`) et une connexion en paire torsadée (`IF_PORT_10BASET`).

  La liste des paramètres possibles est définie dans le fichier `linux/include/linux/netdevice.h`:

Code Linux 2.6.10

```
142 /* Options de selection du support. */
143 enum {
144         IF_PORT_UNKNOWN = 0,
145         IF_PORT_10BASE2,
146         IF_PORT_10BASET,
147         IF_PORT_AUI,
148         IF_PORT_100BASET,
```

```
149         IF_PORT_100BASETX,
150         IF_PORT_100BASEFX
151 };
```

– `dma` spécifie le canal d'accès direct à la mémoire si la carte réseau prend en charge le mode de transfert DMA.

14.1.3.3 État du périphérique

Le champ `state` indique l'état du périphérique. Il a été réintroduit depuis la version du noyau 2.3.43, remplaçant les champs précédents `start` (la carte réseau est ouverte), `interrupt` (le pilote traite directement une interruption) et `tbusy` (tous les tampons de paquets sont occupés).

Les champs précédemment décrits sont à présent remplacés par sept drapeaux dont les bits correspondants sont repérés par les constantes symboliques suivantes, définies dans le fichier `linux/include/linux/netdevice.h`:

Code Linux 2.6.10

```
226 /* Ces bits de drapeau sont reserves au niveau file d'attente de reseau generique,
227  * ils ne peuvent pas etre explicitement references par du code
228  * autre.
229  */
230
231 enum netdev_state_t
232 {
233         __LINK_STATE_XOFF=0,
234         __LINK_STATE_START,
235         __LINK_STATE_PRESENT,
236         __LINK_STATE_SCHED,
237         __LINK_STATE_NOCARRIER,
238         __LINK_STATE_RX_SCHED,
239         __LINK_STATE_LINKWATCH_PENDING
240 };
```

La signification des deux premiers drapeaux est la suivante:

– `__LINK_STATE_START` indique que la carte a été ouverte avec la fonction `dev_open()` que nous étudierons plus loin, c'est-à-dire qu'elle est activée et peut être utilisée. Un tel état ne signifie toutefois pas automatiquement que des paquets peuvent être émis: il est possible que tous les tampons de la carte soient occupés (voir drapeau suivant). Seul l'accès en lecture est possible sur ce drapeau, puisqu'il ne doit être modifié que par des méthodes destinées à la gestion des périphériques. La méthode `netif_running(dev)` sert à vérifier ce drapeau.

– `__LINK_STATE_XOFF` indique que la carte réseau peut accepter un tampon de socket à émettre. Ici aussi, le drapeau ne doit être accessible qu'en lecture. La méthode `netif_queue_stopped(dev)` sert à vérifier ce drapeau.

  Ce drapeau remplace le champ `dev->tbusy` des versions antérieures. Dans les versions antérieures, il existait trois situations particulières pour accéder au champ `tbusy`. Après son remplacement, les fonctions suivantes ont été introduites, améliorant ainsi nettement la clarté et le style de programmation:

  – *Clôture de la transmission*: lorsque les tampons de paquet d'une carte réseau sont occupés, la remise ultérieure de paquets à la carte était arrêtée avec `dev->busy = 1`. Désormais, il existe pour ce faire la fonction (en ligne) `netif_stop_queue(dev)`, dont le but est de positionner à 1 le bit `__LINK_STATE_XOFF`. Aucun paquet n'est ainsi retiré de la file d'attente et remis à la carte.

- *Autorisation de la transmission* : après avoir émis un paquet provenant du tampon (circulaire) de paquet, la carte réseau peut à nouveau accepter des paquets provenant du noyau. Pour ce faire, il existe la méthode `netif_start_queue(dev)` qui supprime le drapeau `__LINK_STATE_XOFF`.
- *Début de la transmission* : la méthode `netif_start_queue()` n'autorise à nouveau que la remise de tampons de socket à la carte réseau. La méthode `netif_wake_queue(dev)`, quant à elle, autorise à nouveau la remise de paquets et déclenche l'interruption logicielle `NET_TX` qui met en œuvre la remise des paquets à la carte réseau.

Le champ `interrupt` n'a aucun équivalent dans les versions ultérieures à 2.3.43.

14.1.3.4 Liste chaînée des périphériques réseau

Nous avons vu que les périphériques réseau doivent être enregistrés sous la forme d'une entité du type `struct net_device` dans une liste chaînée de nom `dev_base`. Le champ `next` permet la constitution de cette liste chaînée.

Le parcours de cette liste sera effectué par une boucle commençant par :

```
for (dev = dev_base; dev; dev = dev->next)
```

14.1.3.5 Fonction d'initialisation du périphérique

La fonction `init()` est chargée de la vérification de la présence physique d'un adaptateur réseau et de l'initialisation de sa structure `struct net_device` avec les informations de pilote appropriées.

L'argument de la fonction `init()` est un pointeur sur le descripteur de périphérique à initialiser. La valeur de renvoi est 0 ou un code d'erreur négatif (par exemple `-ENODEV` si aucun adaptateur n'a été découvert).

14.1.3.6 Liste chaînée des périphériques en attente

Le champ `next_sched` sert (utilisé deux fois uniquement dans tout le code) pour la liste chaînée des périphériques réseau en attente de réception, à travers le type structuré `struct sofnet_data` que nous étudierons plus tard.

14.1.3.7 Index d'interfaçage physique

Nous avons déjà dit qu'un périphérique réseau n'est pas repéré, contrairement aux périphériques caractère et bloc, par un nombre majeur et un nombre mineur. Cependant à chaque périphérique réseau enregistré est associé un numéro unique qui constitue son champ `ifindex`.

Lors de la génération d'un nouveau périphérique réseau, la fonction `dev_get_index()` lui assigne un index nouveau et non utilisé. En comparaison de la recherche par le nom, l'index permet une découverte plus rapide d'un périphérique dans la liste de tous les périphériques.

Le champ `iflink` indique l'index du périphérique réseau à l'aide duquel un paquet est émis. Normalement, il s'agit de l'index `ifindex`, mais pour les périphériques de tunnelisation, comme `ipip`, l'index de périphérique réseau est consigné dans `iflink` à l'aide du paquet duquel le paquet encapsulé est finalement émis. Ce champ ne nous intéressera pas dans cet ouvrage.

14.1.3.8 Statistiques

Un certain nombre d'informations (telles que le nombre total de paquets reçus par le périphérique) sont détenues dans une entité du type structuré `net_device_stats`. Celui-ci est défini

dans le fichier `linux/include/linux/netdevice.h`: Code Linux 2.6.10

```
103 /*
104  *      Statistiques sur le peripherique reseau. Analogues aux stats ether 2.0 mais
105  *      avec des compteurs d'octets.
106  */
107
108 struct net_device_stats
109 {
110         unsigned long   rx_packets;             /* paquets recus au total               */
111         unsigned long   tx_packets;             /* paquets transmis au total            */
112         unsigned long   rx_bytes;               /* octets recus au total                */
113         unsigned long   tx_bytes;               /* octets transmis au total             */
114         unsigned long   rx_errors;              /* mauvais paquets recus                */
115         unsigned long   tx_errors;              /* problemes de transmission de paquet  */
116         unsigned long   rx_dropped;             /* pas d'espace dans les tampons linux  */
117         unsigned long   tx_dropped;             /* pas d'espace disponible dans linux   */
118         unsigned long   multicast;              /* paquets multidiffusion recus         */
119         unsigned long   collisions;
120
121         /* Details sur les erreurs en reception : */
122         unsigned long   rx_length_errors;
123         unsigned long   rx_over_errors;         /* tampon de reception en anneau sature */
124         unsigned long   rx_crc_errors;          /* paquet recu avec erreur sur crc      */
125         unsigned long   rx_frame_errors;        /* erreur d'alignement dans trame recue */
126         unsigned long   rx_fifo_errors;         /* fifo de reception saturee            */
127         unsigned long   rx_missed_errors;       /* paquet manque en reception           */
128
129         /* Details sur les erreurs en transmission */
130         unsigned long   tx_aborted_errors;
131         unsigned long   tx_carrier_errors;
132         unsigned long   tx_fifo_errors;
133         unsigned long   tx_heartbeat_errors;
134         unsigned long   tx_window_errors;
135
136         /* pour cslip etc */
137         unsigned long   rx_compressed;
138         unsigned long   tx_compressed;
139 };
```

La méthode `get_stats()`, spécifique à chaque carte réseau, permet d'obtenir ces statistiques et ces informations sur la carte réseau et ses activités.

14.1.3.9 Périphériques sans fil

Les trois champs `get_wirelees_stats()`, `wireless_handlers` et `wireless_data` concernent les périphériques sans fil. Ils ne nous intéresserons pas dans cet ouvrage.

14.1.3.10 Opérations liées à Ethernet

Le champ `ethtool_ops` concerne les opérations liées à Ethernet, égal à `NULL` dans le cas d'un périphérique non Ethernet. Nous reviendrons sur le type `struct ethtool_ops` lors de l'étude de l'implémentation d'Ethernet au chapitre 16.

14.1.4 Champs généraux

14.1.4.1 Estampilles temporelles

Le champ `trans_start` spécifie le moment précis (en *jiffies*) du début de la dernière transmission. Si au bout d'un certain temps, le pilote n'a reçu aucune confirmation de l'émission du paquet

(interruption de confirmation), il prend les mesures appropriées. Le minuteur `watchdog_timer` existe dans ce but depuis la version 2.4 du noyau.

Le champ `last_rx` devrait contenir le moment précis (en *jiffies*) d'arrivée du dernier paquet. En fait il n'est pas utilisé.

14.1.4.2 Partie privée

Le champ `priv` permet d'adjoindre une structure spécifique à la carte réseau. Il s'agira en général d'une entité du type `struct net_local` qui dépend, lui aussi, du type de carte utilisée.

Ce type est, par exemple, défini dans le fichier `linux/drivers/net/3c501.h` dans le cas de la carte 3Com 501 :

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 };
```

14.1.4.3 Carte réseau maître

Certains ordinateurs possèdent plusieurs cartes réseau, par exemple les serveurs Web. Le champ `master` spécifie alors la carte maître d'un groupe de cartes réseau. Ce champ ne nous intéressera pas dans ce livre.

14.1.4.4 Liste d'élection

Le champ `poll_list` permet de se référer à la file d'attente, en réception et en émission, de la carte réseau. En effet, la carte possède souvent un tampon permettant de contenir quelques paquets (mais un seul paquet pour la carte 3Com 501), toutefois cela est insuffisant. On doit donc s'aider de la mémoire vive de l'ordinateur.

Les champs `quota` et `weight` permettent de gérer celle-ci.

Le type général `struct list_head` est défini dans le fichier en-tête `linux/include/linux/list.h` :

Code Linux 2.6.10

```
18 /*
19  * Implementation d'une liste doublement chainee simple.
20  *
21  * Certaines des fonctions internes ("__xxx") sont utiles lorsqu'on
22  * manipule les listes en entier plutot que des entrees parce que nous
23  * connaissons deja les entrees next/prev et que nous pouvons
24  * generer du meilleur code en les utilisant directement au lieu d'utiliser
25  * les routines a une entree generique.
26  */
27
28 struct list_head {
29         struct list_head *next, *prev;
30 };
```

14.1.4.5 Stratégie de mise en file d'attente

Les champs `qdisc`, `qdisc_sleeping`, `qdisc_list`, `qdisc_ingress` et `tx_queue_len` permettent la gestion des files d'attente.

Le périphérique réseau ne comporte généralement qu'une seule file d'attente et fonctionne suivant le principe du FIFO. Toutefois, il est possible de définir plusieurs files d'attente et de les utiliser grâce à une stratégie (le champ `qdisc`) particulière.

Le type `struct Qdisc` reflète la stratégie de contrôle des files d'attente du périphérique réseau en cours. Il est défini dans le fichier `linux/net/sch_generic.h` : Code Linux 2.6.10

```
27 struct Qdisc
28 {
29         int                     (*enqueue)(struct sk_buff *skb, struct Qdisc *dev);
30         struct sk_buff *        (*dequeue)(struct Qdisc *dev);
31         unsigned                flags;
32 #define TCQ_F_BUILTIN   1
33 #define TCQ_F_THROTTLED 2
34 #define TCQ_F_INGRESS   4
35         int                     padded;
36         struct Qdisc_ops        *ops;
37         u32                     handle;
38         u32                     parent;
39         atomic_t                refcnt;
40         struct sk_buff_head     q;
41         struct net_device       *dev;
42         struct list_head        list;
43
44         struct gnet_stats_basic bstats;
45         struct gnet_stats_queue qstats;
46         struct gnet_stats_rate_est      rate_est;
47         spinlock_t              *stats_lock;
48         struct rcu_head         q_rcu;
49         int                     (*reshape_fail)(struct sk_buff *skb,
50                                         struct Qdisc *q);
51
52         /* Ce champ est desavoue mais il est encore utilise par CBQ
53          * et il restera en vie jusqu'a ce qu'une meilleure solution soit inventee.
54          */
55         struct Qdisc            *__parent;
56 };
```

La signification des champs est la suivante :

- La méthode `enqueue()` permet d'ajouter un élément à la file d'attente.
- La méthode `dequeue()` permet de retirer un élément de la file d'attente.
- Le champ `flags` est un champ de bits dont trois seulement sont significatifs, repérés par les constantes symboliques citées.
- Le champ `padded` spécifie le rembourrage utilisé.
- Nous allons revenir ci-dessous sur l'ensemble des opérations, le champ `ops`, permises sur une instance de cette structure.
- La poignée `handle` permet de repérer l'instance.
- Le parent `parent` permet de se repérer dans une liste chaînée.
- Le compteur de référence `refcnt` permet de ne pas détruire l'instance si quelqu'un y fait encore référence.
- Le champ `q` permet de pointer sur le premier élément de la file d'attente.
- Le champ `dev` spécifie la carte réseau associée à la file d'attente.

– Les champ `bstats`, `qstats` et `rate_est` permettent de détenir des informations.

Code Linux 2.6.10

Les types `struct gnet_stats_basic`, `struct gnet_stats_queue` et `struct gnet_stats_rate_est` sont définis dans le fichier en-tête `linux/include/linux/gen_stats.h`:

```
16 /**
17  * struct gnet_stats_basic - statistiques sur les octets/paquets
18  * @bytes : nombre d'octets vus
19  * @packets : nombre de paquets vus
20  */
21 struct gnet_stats_basic
22 {
23         __u64   bytes;
24         __u32   packets;
25 };
26
27 /**
28  * struct gnet_stats_rate_est - estimateur de taux
29  * @bps : taux d'octets en cours
30  * @pps : taux de paquet en cours
31  */
32 struct gnet_stats_rate_est
33 {
34         __u32   bps;
35         __u32   pps;
36 };
37
38 /**
39  * struct gnet_stats_queue - statistiques de file d'attente
40  * @qlen : longueur de la file d'attente
41  * @backlog : taille maximale d'elements que peut contenir la file d'attente
42  * @drops : nombre de paquets ecartes
43  * @requeues : nombre de remises dans la file d'attente
44  * @overlimits : nombre de mises en file d'attente depassant la limite
45  */
46 struct gnet_stats_queue
47 {
48         __u32   qlen;
49         __u32   backlog;
50         __u32   drops;
51         __u32   requeues;
52         __u32   overlimits;
53 };
```

– Le champ `stats_lock` permet de verrouiller les informations statistiques.

– Le champ `q_rcu` sert au mécanisme de mise à jour en copie et écriture d'exlusion mutuelle (*Read-Copy Update* en anglais).

Code Linux 2.6.10

Le type général `struct rcu_head` est défini dans le fichier en-tête `linux/include/linux/rcupdate.h`:

```
45 /**
46  * struct rcu_head - structure de rappel a utiliser avec la RCU
47  * @next : requete de mise a jour suivante dans une liste
48  * @func : fonction de mise a jour en cours a appeler apres la periode de grace.
49  */
50 struct rcu_head {
51         struct rcu_head *next;
52         void (*func)(struct rcu_head *head);
53 };
```

– La méthode `reshape_fail()` permet d'intervenir en cas d'échec.

– Le champ `__parent` n'est plus utilisé, comme indiqué.

Le type `struct Qdisc_ops` des opérations permises sur une instance du type défini ci-dessus est également défini dans le fichier `linux/net/sch_generic.h`:

Code Linux 2.6.10

```
58 struct Qdisc_class_ops
59 {
60         /* Manipulation du qdisc fils */
61         int                     (*graft)(struct Qdisc *, unsigned long cl,
62                                         struct Qdisc *, struct Qdisc **);
63         struct Qdisc *          (*leaf)(struct Qdisc *, unsigned long cl);
64
65         /* Routines de manipulation de la classe */
66         unsigned long           (*get)(struct Qdisc *, u32 classid);
67         void                    (*put)(struct Qdisc *, unsigned long);
68         int                     (*change)(struct Qdisc *, u32, u32,
69                                         struct rtattr **, unsigned long *);
70         int                     (*delete)(struct Qdisc *, unsigned long);
71         void                    (*walk)(struct Qdisc *, struct qdisc_walker * arg);
72
73         /* Manipulation du filtre */
74         struct tcf_proto **     (*tcf_chain)(struct Qdisc *, unsigned long);
75         unsigned long           (*bind_tcf)(struct Qdisc *, unsigned long,
76                                         u32 classid);
77         void                    (*unbind_tcf)(struct Qdisc *, unsigned long);
78
79         /* Specifique a rtnetlink */
80         int                     (*dump)(struct Qdisc *, unsigned long,
81                                         struct sk_buff *skb, struct tcmsg*);
82         int                     (*dump_stats)(struct Qdisc *, unsigned long,
83                                         struct gnet_dump *);
84 };
85
86  struct Qdisc_ops
87  {
88          struct Qdisc_ops        *next;
89          struct Qdisc_class_ops  *cl_ops;
90          char                    id[IFNAMSIZ];
91          int                     priv_size;
92
93          int                     (*enqueue)(struct sk_buff *, struct Qdisc *);
94          struct sk_buff *        (*dequeue)(struct Qdisc *);
95          int                     (*requeue)(struct sk_buff *, struct Qdisc *);
96          unsigned int            (*drop)(struct Qdisc *);
97
98          int                     (*init)(struct Qdisc *, struct rtattr *arg);
99          void                    (*reset)(struct Qdisc *);
100         void                    (*destroy)(struct Qdisc *);
101         int                     (*change)(struct Qdisc *, struct rtattr *arg);
102
103         int                     (*dump)(struct Qdisc *, struct sk_buff *);
104         int                     (*dump_stats)(struct Qdisc *, struct gnet_dump *);
105
106         struct module           *owner;
107 };
```

14.1.4.6 Synchronisation de l'émission

Les champs `xmit_lock`, `xmit_lock_owner` et `queue_lock` permettent de synchroniser l'émission des trames:

– Le verrou rotatif `xmit_lock` est utilisé pour éviter des appels multiples simultanés à la fonction `hard_start_xmit()` du pilote de périphérique.

- Le champ `xmit_lock_owner` est le numéro du microprocesseur qui a obtenu `xmit_lock`, utile dans le cas où il y a plusieurs microprocesseurs. Si aucun microprocesseur n'émet momentanément, ce champ prend la valeur -1.
- Le verrou rotatif `queue_lock` est utilisé pour éviter des appels multiples simultanés à la file d'attente.

14.1.4.7 Installation/désinstallation

Le champ `refcnt` permet de maintenir le nombre de renvois à cette carte, ce qui évite de la désinstaller si celui-ci est non nul.

La liste `todo_list` permet de savoir si la carte est installée ou non.

Les chaînes de hachage `name_hlist` et `index_hlist` permettent de repérer plus facilement l'instance.

Code Linux 2.6.10

Le type général `struct hlist_node` est défini dans le fichier `linux/include/linux/list.h`:

```
503 struct hlist_node {
504         struct hlist_node *next, **pprev;
505 };
```

L'état `reg_state` (pour *REGister STATE*) permet de savoir où l'on en est du point de vue de l'installation de la carte: non initialisée, en train d'être enregistrée, enregistrée, en train d'être désintallée, désinstallée ou libérée.

14.1.5 Membres concernant la sous-couche MAC

14.1.5.1 Drapeaux de la sous-couche MAC

Les drapeaux `flags` et `gflags` sont des champs de bits, décrivant d'une part les propriétés de la carte réseau et transmettant l'état temporaire (par exemple `IFF_UP`). Les bits sont repérés par des constantes symboliques définies dans le fichier `linux/include/linux/if.h`:

Code Linux 2.6.10

```
29 /* Drapeaux standard de l'interface (netdevice->flags). */
30 #define IFF_UP          0x1             /* l'interface est active                */
31 #define IFF_BROADCAST   0x2             /* adresse de diffusion generale valide  */
32 #define IFF_DEBUG       0x4             /* en phase de debogage                  */
33 #define IFF_LOOPBACK    0x8             /* est un reseau loopback                */
34 #define IFF_POINTOPOINT 0x10            /* l'interface a un lien p-p             */
35 #define IFF_NOTRAILERS  0x20            /* eviter l'utilisation de remorques     */
36 #define IFF_RUNNING     0x40            /* ressources allouees                   */
37 #define IFF_NOARP       0x80            /* pas de protocole ARP                  */
38 #define IFF_PROMISC     0x100           /* recoit tous les paquets               */
39 #define IFF_ALLMULTI    0x200           /* recoit tous les paquets multidiffusion */
40
41 #define IFF_MASTER      0x400           /* maitre d'un equilibreur de chargement  */
42 #define IFF_SLAVE       0x800           /* esclave d'un equilibreur de chargement  */
43
44 #define IFF_MULTICAST   0x1000          /* Supporte la multidiffusion            */
45
46 #define IFF_VOLATILE
       (IFF_LOOPBACK|IFF_POINTOPOINT|IFF_BROADCAST|IFF_MASTER|IFF_SLAVE|IFF_RUNNING)
47
48 #define IFF_PORTSEL     0x2000          /* peut positionner le type de support    */
49 #define IFF_AUTOMEDIA   0x4000          /* auto-selection du support active       */
50 #define IFF_DYNAMIC     0x8000          /* peripherique telephonique avec adresse
                                           changeante                             */
```

avec un préfixe `IFF` pour *InterFace Flag*. La signification lorsque le bit est à 1 est la suivante:

- `IFF_UP`: le noyau positionne ce bit (qui peut être lu, mais pas écrit par le périphérique) lorsque l'interface est active et prête à transférer des paquets.

- `IFF_BROADCAST`: ce bit indique que la carte réseau permet la diffusion générale et que l'adresse de diffusion générale est valide.
- `IFF_DEBUG`: ce bit indique qu'on est en mode de débogage. Il peut être positionné ou remis à zéro grâce à la commande utilisateur `ioctl()`.
- `IFF_LOOPBACK`: ce bit ne doit être positionné que pour l'interface en boucle.
- `IFF_POINTOPOINT`: ce bit signale que l'interface est connectée à une liaison point-à-point. Il est positionné par la commande utilisateur `ifconfig`.
- `IFF_NOTRAILERS`: ce bit n'est pas utilisé par Linux mais il existe pour des raisons de compatibilité avec BSD.
- `IFF_RUNNING`: ce bit indique que l'interface est active et est en train de travailler. Il est présent pour des raisons de compatibilité avec BSD mais Linux en fait peu d'usage.
- `IFF_NOARP`: ce bit signale que l'interface ne peut pas effectuer d'ARP, ne serait-ce que, par exemple, parce que les interfaces point-à-point n'ont pas besoin d'effectuer d'ARP.
- `IFF_PROMISC`: ce bit est positionné pour activer le mode promiscuité.
- `IFF_MULTICAST`: ce bit est positionné lorsque l'interface peut effectuer des multidiffusions.
  Il est positionné par défaut lors de l'initialisation. Il doit donc être remis à zéro pour les cartes qui ne permettent pas la diffusion restreinte (ce qui est le cas de la carte 3Com 501, par exemple).
- `IFF_ALLMULTI`: ce bit indique à l'interface qu'il faut recevoir tous les paquets de diffusion restreinte. Il ne peut être positionné, par le noyau, que si `IFF_MULTICAST` l'est également. Ce bit ne peut qu'être lu par la carte.
- `IFF_MASTER` et `IFF_SLAVE`: ces bits ne sont utilisés que par le programme. Le pilote de périphérique ne s'en sert pas.
- `IFF_PORTSEL` et `IFF_AUTOMEDIA`: ces bits signalent que le périphérique est capable de passer d'un type de support à un autre, par exemple de paire torsadée à câble coaxial. Si `IFF_AUTOMEDIA` est positionné, le périphérique sélectionne le support automatiquement.
- `IFF_DYNAMIC`: ce bit indique que l'adresse de la carte peut être changée. Ce mode est utilisé par les périphériques de connexion par appel téléphonique.

Lorsqu'un programme change `IFF_UP`, la méthode de périphérique `open()` ou `stop()` est appelée. Lorsqu'un drapeau est modifié, la méthode `set_multicast_list()` est appelée. Si le pilote de périphérique doit effectuer une action suite à la modification d'un drapeau, cette action doit donc être placée dans `set_multicast_list()`. Par exemple, lorsque `IFF_PROMISC` est positionné ou remis à zéro, `set_multicast_list()` doit le notifier au filtre matériel du périphérique.

Ces commutateurs peuvent être positionnés à l'aide de la commande `ifconfig`.

Les positions des drapeaux de `priv_flags` sont également définies dans le fichier `linux/include/linux/if.h`:

Code Linux 2.6.0

```
52 /* Drapeaux prives (de l'utilisateur) pour l'interface (netdevice->priv_flags). */
53 #define IFF_802_1Q_VLAN 0x1             /* peripherique VLAN 802.1Q.    */
54 #define IFF_EBRIDGE     0x2             /* peripherique pont Ethernet.  */
55
56 #define IF_GET_IFACE    0x0001          /* pour les requetes uniquement */
57 #define IF_GET_PROTO    0x0002
58
59 /* Pour les definitions voir hdlc.h */
60 #define IF_IFACE_V35    0x1000          /* interface serie V.35         */
61 #define IF_IFACE_V24    0x1001          /* interface serie V.24         */
```

```
62 #define IF_IFACE_X21    0x1002          /* interface serie X.21         */
63 #define IF_IFACE_T1     0x1003          /* interface serie telco T1     */
64 #define IF_IFACE_E1     0x1004          /* interface serie telco E1     */
65 #define IF_IFACE_SYNC_SERIAL 0x1005     /* ne peut pas etre positionne par logiciel */
66 #define IF_IFACE_X21D   0x1006          /* X.21 Dual Clocking (FarSite) */
67
68 /* Pour les definitions voir hdlc.h */
69 #define IF_PROTO_HDLC   0x2000          /* protocole HDLC brut          */
70 #define IF_PROTO_PPP    0x2001          /* protocole PPP                */
71 #define IF_PROTO_CISCO  0x2002          /* protocole HDLC Cisco         */
72 #define IF_PROTO_FR     0x2003          /* protocole Frame Relay        */
73 #define IF_PROTO_FR_ADD_PVC 0x2004      /*    Creer   FR PVC            */
74 #define IF_PROTO_FR_DEL_PVC 0x2005      /*    Retirer FR PVC            */
75 #define IF_PROTO_X25    0x2006          /* X.25                         */
76 #define IF_PROTO_HDLC_ETH 0x2007        /* HDLC brut, emulation d'Ethernet */
77 #define IF_PROTO_FR_ADD_ETH_PVC 0x2008  /*  Creer   FR pont-Ethernet PVC */
78 #define IF_PROTO_FR_DEL_ETH_PVC 0x2009  /*  Retirer FR pont-Ethernet PVC */
79 #define IF_PROTO_FR_PVC 0x200A          /* pour lire le statut PVC       */
80 #define IF_PROTO_FR_ETH_PVC 0x200B
81 #define IF_PROTO_RAW    0x200C          /* RAW Socket                   */
```

14.1.5.2 Unité de transfert d'information maximale

Chaque type de réseau possède une **unité de transfert d'information maximale**, ou **MTU** (pour l'anglais *Maximum Transfert Unit*), dans laquelle doit s'insérer chaque trame. Le champ `mtu` précise cette valeur pour le réseau auquel la carte est reliée.

Par exemple Ethernet a une MTU de 1 500 octets.

14.1.5.3 Type matériel

Le type matériel de l'interface est utilisé par le protocole de résolution d'adresse **ARP** pour déterminer quelle sorte d'adresse physique est supportée par le périphérique réseau. La valeur pour les cartes Ethernet est `ARPHDR_ETHER`.

Les valeurs des types matériels ont été codifiées dans [RFC 1700] concernant le protocole ARP. Linux définit des constantes supplémentaires qui ne sont pas codifiées dans cette RFC.

Les types pris en charge par Linux sont repérés par l'une des constantes symboliques définies dans le fichier en-tête `linux/include/linux/if_arp.h`:

Code Linux 2.6.10

```
28 /* Identificateurs MATERIELS de protocole ARP. */
29 #define ARPHRD_NETROM   0               /* de KA9Q : pseudo NET/ROM           */
30 #define ARPHRD_ETHER    1               /* Ethernet 10Mbps                    */
31 #define ARPHRD_EETHER   2               /* Ethernet Experimental              */
32 #define ARPHRD_AX25     3               /* AX.25 niveau 2                      */
33 #define ARPHRD_PRONET   4               /* PROnet token ring                  */
34 #define ARPHRD_CHAOS    5               /* Chaosnet                           */
35 #define ARPHRD_IEEE802  6               /* IEEE 802.2 Ethernet/TR/TB          */
36 #define ARPHRD_ARCNET   7               /* ARCnet                             */
37 #define ARPHRD_APPLETLK 8               /* APPLEtalk                          */
38 #define ARPHRD_DLCI     15              /* Frame Relay DLCI                   */
39 #define ARPHRD_ATM      19              /* ATM                                */
40 #define ARPHRD_METRICOM 23              /* Metricom STRIP (nouvel id IANA)    */
41 #define ARPHRD_IEEE1394 24              /* IEEE 1394 IPv4 - RFC 2734          */
42 #define ARPHRD_EUI64    27              /* EUI-64                             */
43 #define ARPHRD_INFINIBAND 32            /* InfiniBand                         */
44
45 /* Types muets pour du materiel non ARP */
46 #define ARPHRD_SLIP     256
47 #define ARPHRD_CSLIP    257
48 #define ARPHRD_SLIP6    258
```

```
49 #define ARPHRD_CSLIP6   259
50 #define ARPHRD_RSRVD    260             /* Notional KISS type                 */
51 #define ARPHRD_ADAPT    264
52 #define ARPHRD_ROSE     270
53 #define ARPHRD_X25      271             /* CCITT X.25                         */
54 #define ARPHRD_HWX25    272             /* Cartes avec X.25 dans le firmware  */
55 #define ARPHRD_PPP      512
56 #define ARPHRD_CISCO    513             /* HDLC Cisco                         */
57 #define ARPHRD_HDLC     ARPHRD_CISCO
58 #define ARPHRD_LAPB     516             /* LAPB                               */
59 #define ARPHRD_DDCMP    517             /* protocole DDCMP de Digital         */
60 #define ARPHRD_RAWHDLC  518             /* HDLC brut                          */
61
62 #define ARPHRD_TUNNEL   768             /* Tunnel IPIP                        */
63 #define ARPHRD_TUNNEL6  769             /* Tunnel IP6IP6                      */
64 #define ARPHRD_FRAD     770             /* Frame Relay Access Device          */
65 #define ARPHRD_SKIP     771             /* SKIP vif                           */
66 #define ARPHRD_LOOPBACK 772             /* Peripherique Loopback              */
67 #define ARPHRD_LOCALTLK 773             /* Peripherique Localtalk             */
68 #define ARPHRD_FDDI     774             /* Fiber Distributed Data Interface   */
69 #define ARPHRD_BIF      775             /* AP1000 BIF                         */
70 #define ARPHRD_SIT      776             /* Peripherique sit0 - IPv6-in-IPv4   */
71 #define ARPHRD_IPDDP    777             /* Tunellisation IP sur DDP           */
72 #define ARPHRD_IPGRE    778             /* GRE sur IP                         */
73 #define ARPHRD_PIMREG   779             /* PIMSM register interface           */
74 #define ARPHRD_HIPPI    780             /* High Performance Parallel Interface */
75 #define ARPHRD_ASH      781             /* Nexus 64Mbps Ash                   */
76 #define ARPHRD_ECONET   782             /* Acorn Econet                       */
77 #define ARPHRD_IRDA     783             /* Linux-IrDA                         */
78 /* ARP fonctionne differemment sur des supports FC differents ... donc  */
79 #define ARPHRD_FCPP     784             /* Point to point fibrechannel        */
80 #define ARPHRD_FCAL     785             /* Fibrechannel arbitrated loop       */
81 #define ARPHRD_FCPL     786             /* Fibrechannel public loop           */
82 #define ARPHRD_FCFABRIC 787             /* Fibrechannel fabric                */
83         /* 787->799 reserve pour les types de media fibrechannel */
84 #define ARPHRD_IEEE802_TR 800           /* Ident de type magique pour tr      */
85 #define ARPHRD_IEEE80211 801            /* IEEE 802.11                        */
86 #define ARPHRD_IEEE80211_PRISM 802      /* IEEE 802.11 + en-tete Prism2       */
87
88 #define ARPHRD_VOID       0xFFFF        /* Type vide, rien n'est connu        */
89 #define ARPHRD_NONE       0xFFFE        /* Longueur d'en-tete nulle           */
```

Seuls quelques types (`ARPHRD_ETHER`, `ARPHRD_IEEE802`, `ARPHRD_SLIP`, `ARPHRD_PPP`, `ARPHRD_LOOPBACK` et `ARPHRD_VOID`) nous intéresserons dans ce livre.

14.1.5.4 Longueur de l'en-tête matériel

Le champ `hard_header_len` indique la longueur de l'en-tête et du suffixe d'une trame, c'est-à-dire ce qui n'est pas occupé par les données, en ne comptant pas les emplacements occupés par les adresses de l'émetteur et du destinataire.

Par exemple, pour Ethernet on a `ETH_HLEN`, que nous reverrons dans le chapitre 16, égal à 14, correspondant à 8 octets de préambule, 2 octets de type et 4 octets de CRC.

14.1.5.5 Adresses de l'interface

L'adresse physique de la carte, le champ `dev_addr`, est seulement structurée par sa longueur maximale. La constante symbolique `MAX_ADDR_LEN` est définie dans le fichier `linux/include/linux/netdevice.h`:

Code Linux 2.6.10

```
74 #define MAX_ADDR_LEN    32              /* Plus grande longueur d'adresse physique */
```

ce qui suffit pour les 48 bits (ou six octets) d'une adresse physique MAC.

La longueur effective de l'adresse, le champ `addr_len`, prendra la valeur `ETH_LEN`, c'est-à-dire 6, dans le cas d'Ethernet.

L'adresse physique doit être lue sur le périphérique par une méthode qui dépend du type de périphérique.

Ces valeurs sont fixées, à l'aide de la commande `ifconfig`, lors de l'activation d'une carte réseau.

14.1.5.6 Attribut lié à la diffusion générale

L'adresse de diffusion générale, le champ `broadcast`, dépend du protocole. Rappelons que tous les bits de l'adresse de diffusion générale sont égaux à 1 dans le cas d'Ethernet, c'est-à-dire qu'elle est constituée de six octets de valeur `FFh`.

14.1.5.7 Attributs liés à la multidiffusion

Les adresses de multidiffusion appartiennent à une liste chaînée du type `struct dev_mc_list`. Celui-ci est défini dans le fichier `linux/include/linux/netdevice.h`:

Code Linux 2.6.10

```
179 /*
180  *      Nous marquons la multidiffusion avec ces structures.
181  */
182
183 struct dev_mc_list
184 {
185         struct dev_mc_list      *next;
186         __u8                    dmi_addr[MAX_ADDR_LEN];
187         unsigned char           dmi_addrlen;
188         int                     dmi_users;
189         int                     dmi_gusers;
190 };
```

Le nombre d'adresses dans cette liste est indiqué par le champ `mc_count`.

Si la carte réseau reçoit un paquet dont l'adresse de destination est contenue dans la liste `dev_mc_list`, elle doit le transmettre aux couches supérieures. Les adresses de cette liste sont transmises à la carte à l'aide de la méthode du pilote `set_multicast_list()`.

Le champ `allmulti` spécifie si la carte réseau accepte ou non les messages envoyés en multidiffusion. Il est manipulé à travers la fonction `dev_set_allmulti()`, définie dans le fichier `linux/net/core/dev.c`:

Code Linux 2.6.10

```
2176 /**
2177  *      dev_set_allmulti        - met a jour le compteur allmulti sur un peripherique
2178  *      @dev : peripherique
2179  *      @inc : modificateur
2180  *
2181  *      Active ou inhibe la reception de toutes les trames de multidiffusion pour un
2182  *      peripherique. Tant que le compteur du peripherique reste en dessous de zero,
2183  *      l'interface reste a l'ecoute de toutes les interfaces. Une fois qu'il atteint zero,
2184  *      le peripherique revient a l'operation normale de filtrage. Une valeur @inc negative
2185  *      est utilisee pour ecarter le compteur lorsqu'on libere une ressource necessaitant
        *      toutes les adresses de multidiffusion.
2186  */
2187
2188 void dev_set_allmulti(struct net_device *dev, int inc)
```

14.1.5.8 Acceptation de paquets destinés à d'autres hôtes

Le champ `promiscuity` spécifie si la carte doit accepter les trames destinées à d'autres ordinateurs, autrement dit si on se trouve dans le mode spécial de promiscuité.

Il est manipulé à travers la fonction `dev_set_promiscuity()`, définie dans le fichier `linux/net/core/dev.c`:

Code Linux 2.6.10

```
2151 /**
2152  *      dev_set_promiscuity     - met a jour le compteur de promiscuite sur un peripherique
2153  *      @dev : peripherique
2154  *      @inc : modificateur
2155  *
2156  *      Active ou inhibe la promiscuite pour un peripherique. Tant que le compteur du
2157  *      peripherique reste en-dessous de zero, l'interface admet la promiscuite.
2158  *      Des qu'il atteint zero, le peripherique revient a l'operation normale de
2159  *      filtrage. Une valeur negative de inc est utilisee pour inhiber le
             mode promiscuite du peripherique.
2160  */
2161 void dev_set_promiscuity(struct net_device *dev, int inc)
```

14.1.5.9 Attributs liés aux délais

La carte réseau utilise un certain nombre de temporisateurs, pour déterminer si le délai s'est écoulé depuis l'expédition d'une trame par exemple. Ils sont placés dans une liste, le champ `watchdog_timer`.

Le délai minimum (en *jiffies*) que l'on doit attendre avant que la couche réseau décide qu'une émission a échouée fait l'objet du champ `watchdog_timeo`.

La routine de traitement `dev_watchdog()` vérifie si, depuis la dernière émission d'un paquet, `watchdog_timeo` unités de temps se sont écoulées. Si tel est le cas, des problèmes sont apparus au moment de l'émission du dernier paquet et la carte réseau nécessite une vérification. On appelle pour ce faire la méthode de pilote `tx_timeout()`.

14.1.5.10 Nombre maximum de trames en émission

Le champ `tx_queue_len` spécifie le nombre maximum de trames qui peuvent être placées dans la file d'attente en émission du périphérique. Cette valeur est positionnée à 100 dans le cas d'Ethernet mais on peut la changer si nécessaire.

Il ne faut pas confondre `tx_queue_len` avec les tampons de la carte réseau proprement dite. Normalement une carte réseau dispose elle-même d'un tampon circulaire supplémentaire pour 16 ou 32 paquets (mais seulement 1 pour la carte 3Com 501).

14.1.6 Membres concernant la couche réseau

A priori la couche réseau ne devrait rien avoir à faire avec le pilote de périphérique. Cependant certaines cartes prennent en charge certains protocoles de la couche réseau.

14.1.6.1 Pointeurs sur des protocoles spécifiques

Ces protocoles nécessitent des données supplémentaires. C'est l'objet des six champs `atalk_ptr`, `ip_ptr`, `dn_ptr`, `ip6_ptr`, `ec_ptr` et `ax25_ptr` concernant respectivement Appletalk, IPv4, DECnet, IPv6, Econet et AX.25.

Si la carte réseau est entre autre configurée pour le protocole Internet, `ip_ptr` renvoie à une structure du type `in_device` dans laquelle sont gérées les informations et les paramètres de configuration de l'instance IP en question. La structure `in_device` gère par exemple la liste des

adresses IP de la carte réseau, la liste des groupes de multidiffusion IP ainsi que les paramètres du protocole ARP.

14.1.6.2 Comportement à l'égard de la somme de contrôle

Le champ de bits `features` permet de connaître les caractéristiques de la carte réseau: ne peut vérifier les sommes de contrôle que pour TCP/UDP sur IPv4, n'exige pas de somme de contrôle, peut vérifier la somme de contrôle de tous les paquets.

Code Linux 2.6.10

Un commentaire se trouve dans le fichier en-tête `linux/include/linux/skbuff.h`:

```
73  *      Le peripherique doit montrer ses possibilites dans dev->features, positionne
74  *      au moment du demarrage du peripherique.
75  *      NETIF_F_HW_CSUM - c'est un peripherique intelligent, il est capable de calculer
76  *                        la somme de controle dans tous les cas.
77  *      NETIF_F_NO_CSUM - loopback ou simple support de saut digne de confiance.
78  *      NETIF_F_IP_CSUM - le peripherique est bete. Il est seulement capable de
                              calculer la somme de controle
79  *                        pour TCP/UDP sur IPv4. Soupir. Les vendeurs aiment cette
80  *                        facon pour une raison inconnue. Quoique, voir le commentaire
81  *                        ci-dessus sur CHECKSUM_UNNECESSARY. 8)
82  *
83  *      Des questions ? Pas de question, bien.              --ANK
84  */
```

14.1.7 Méthodes du pilote de carte réseau

Il existe un certain nombre de fonctions permettant la gestion de la carte, ces fonctions dépendant du type de celle-ci, dont le nom explique le comportement. Certaines de ces méthodes peuvent être laissées à `NULL`.

Quelques-unes de ces méthodes dépendent du matériel de la carte réseau, aussi sont-elles initialisées par la fonction d'initialisation du pilote de la carte réseau. Les autres fonctions sont spécifiques au protocole MAC de la carte réseau employée et on peut les initialiser à l'aide de méthodes spécialisées, comme `eth_setup()` par exemple.

14.1.7.1 Méthodes dépendant du matériel

Les méthodes dépendant du matériel sont les suivantes:

- `uninit()` est appelée lors de la déconnexion d'une carte réseau (*via* la fonction `unregister_netdevice()`). Des fonctions spécifiques au pilote éventuellement requises lors du retrait d'un périphérique réseau peuvent être appelées. Cette méthode n'est présente que depuis la version 2.4 du noyau Linux et aucun pilote ne l'utilise actuellement.
- `destructor()` est appelée lorsque la dernière référence à un périphérique réseau a été éliminée, c'est-à-dire lorsque aucune instance de protocole ou d'autres composants ne renvoie plus au noyau Linux sur la structure `net_device`. Cette fonction permet donc de mettre en œuvre des tâches de nettoyage comme la libération de mémoire ou autre opération similaire. Également récente, aucun pilote ne l'utilise actuellement.
- `open()` ouvre (active) une carte réseau. Au moment de l'activation, les ressources système nécessaires (ports d'entrée-sortie, IRQ, DMA, etc.) sont demandées (et marquées comme occupées), le périphérique est démarré et le compteur d'utilisation du module est incrémenté. Seuls les périphériques préalablement enregistrés peuvent être ouverts.

- `stop()` met fin à l'activité d'une carte réseau et libère les ressources système occupées. Le périphérique réseau n'est dorénavant plus actif ; cependant, il se trouve encore dans la liste des périphériques réseau enregistrés.
- `hard_start_xmit()` initie la transmission d'un paquet. En cas de succès (le paquet a été transmis à la carte mais on ne sait pas s'il est parvenu au destinataire), la méthode revient à l'état antérieur avec une valeur de renvoi 0, autrement avec 1.
- `set_multicast_list()` est appelée soit lorsque la liste de multidiffusion change, soit lorsque le drapeau afférent (`IFF_MULTICAST`) change. Elle transmet la liste des adresses MAC de multidiffusion à la carte réseau qui doit réceptionner les paquets.
- `do_ioctl()` permet d'implémenter la commande `ioctl()` spécifique à l'interface.
- `set_config()` change la configuration de l'interface. Cette possibilité peut être utilisée par l'administrateur système. Les pilotes pour les cartes modernes n'ont pas besoin d'implémenter cette méthode en général.
- `tx_timeout()` est appelée lorsqu'on considère que la transmission d'un paquet a échouée parce que le délai s'est écoulé.

14.1.7.2 Méthodes dépendant du protocole MAC

Les méthodes dépendant du protocole MAC sont les suivantes :

- `hard_header()` permet de construire l'en-tête matériel à partir des adresses source et de destination qui ont été récupérées antérieurement.

  La fonction `eth_header()` est l'instance par défaut pour les cartes Ethernet.
- `rebuild_header()` est utilisée pour reconstruire l'en-tête matériel avant qu'un paquet ne soit transmis.

  Dans les versions de Linux antérieures à 2.4, cette méthode était le point d'entrée dans le protocole ARP. Depuis, grâce à l'adaptation au cache `neighbour`, il devrait déjà exister un en-tête matériel enregistré. Par conséquent cette méthode n'est plus appelée que pour de fausses informations dans le cache d'en-tête matériel.
- `set_mac_address()` sert à modifier l'adresse matérielle d'une carte réseau si l'interface permet le changement de celle-ci.
- `hard_header_cache()` est appelée pour remplir une entrée de cache d'en-tête matériel, de type `struct hh_cache` étudié dans la section suivante, avec les résultats obtenus par une requête ARP. Elle remplit un en-tête de trame à l'aide des données transmises. On peut ainsi accéder, lors des processus d'émission suivants, à un en-tête de trame préfabriqué.
- `header_cache_update()` met à jour l'adresse de destination de l'entrée de cache d'en-tête matériel, de type `struct hh_cache`.
- `change_mtu()` modifie l'unité maximale de transfert (MTU) d'un périphérique réseau et met en œuvre les modifications nécessaires.
- `hard_header_parse(skb, haddr)` extrait l'adresse source du paquet contenu dans le paramètre `skb` et la copie dans le tampon situé à l'adresse `haddr`. Elle renvoie la longueur de cette adresse.

14.1.8 Autres champs

14.1.8.1 Champs relatifs aux ponts

Le champ `br_port` indique le numéro de port du pont associé à la carte. Le type `struct net_bridge_port`, défini dans le fichier `linux/net/bridge/br_private.h`, ne nous intéressera pas dans cet ouvrage.

14.1.8.2 Trame de déroutement

Code Linux 2.6.10

Le type `struct divert_blk` du champ `divert` est défini dans le fichier `linux/include/linux/divert.h`:

```
14 #define MAX_DIVERT_PORTS        8       /* Nombre max de ports a derouter (tcp, udp) */
15
16 /* Protocoles pouvant etre deroutes */
17 #define DIVERT_PROTO_NONE       0x0000
18 #define DIVERT_PROTO_IP         0x0001
19 #define DIVERT_PROTO_ICMP       0x0002
20 #define DIVERT_PROTO_TCP        0x0004
21 #define DIVERT_PROTO_UDP        0x0008
22
23 /*
24  *      Ceci est un bloc d'option Ethernet Frame Diverter
25  */
26 struct divert_blk
27 {
28         int           divert;  /* sommes-nous actif */
29         unsigned int  protos;  /* protocoles */
30         u16           tcp_dst[MAX_DIVERT_PORTS]; /* ports de dst tcp specifiques a derouter */
31         u16           tcp_src[MAX_DIVERT_PORTS]; /* ports src tcp specifiques a derouter    */
32         u16           udp_dst[MAX_DIVERT_PORTS]; /* ports de dst udp specifiques a derouter */
33         u16           udp_src[MAX_DIVERT_PORTS]; /* ports src udp specifiques a derouter    */
34 };
```

14.1.8.3 Classe du périphérique

Les périphériques peuvent appartenir à une classe. Le type général `struct class_device` du champ `class_dev` était défini dans le fichier `linux/include/linux/device.h` dans le cas du noyau 2.6.0. Il ne l'est plus pour le noyau 2.6.10, ce qui montre qu'il n'est plus utilisé.

14.2 Entrée de cache d'en-tête matériel

Pour éviter d'avoir à recomposer à chaque fois l'en-tête matériel de la trame, on utilise une entrée de cache d'en-tête matériel. Celle-ci est une entité du type `struct hh_cache`, défini dans le fichier `linux/include/linux/netdevice.h`:

Code Linux 2.6.10

```
192 struct hh_cache
193 {
194         struct hh_cache *hh_next;       /* Entree suivante                  */
195         atomic_t        hh_refcnt;      /* nombre d'utilisateurs            */
196         unsigned short  hh_type;        /* identificateur de protocole, par exemple ETH_P_IP
197                                          *  NOTE : Pour les VLAN, ceci sera le
198                                          *  type encapsule. --BLG
199                                          */
200         int             hh_len;         /* longueur de l'en-tete            */
201         int             (*hh_output)(struct sk_buff *skb);
202         rwlock_t        hh_lock;
```

```
203
204         /* en-tete physique en cache ; utiles pour les necessites d'alignement des
               machines. */
205 #define HH_DATA_MOD     16
206 #define HH_DATA_OFF(__len) \
207         (HH_DATA_MOD - ((__len) & (HH_DATA_MOD - 1)))
208 #define HH_DATA_ALIGN(__len) \
209         (((__len)+(HH_DATA_MOD-1))&~(HH_DATA_MOD - 1))
210         unsigned long   hh_data[HH_DATA_ALIGN(LL_MAX_HEADER) / sizeof(long)];
211 };
```

qui comporte à la fois des champs de contrôle (qui constitueraient un descripteur dans d'autres circonstances) et des données. Les champs de contrôle sont les suivants :

- Les entrées du cache sont placées dans une liste chaînée. Le champ `hh_next` permet de retrouver l'entrée suivante.
- Le champ `hh_refcnt` est le compteur de références habituel, qui permet de ne pas détruire l'entrée si celui-ci n'est pas nul.
- Le verrou rotatif `hh_lock` permet d'éviter un accès simultané à l'entrée du cache.

Les champs de données sont les suivants :

- Le type `hh_type` est l'identificateur du protocole, par exemple `ETH_P_IP` pour un paquet IP.
- La taille de l'en-tête est spécifiée par le champ `hh_len`.
- La méthode `hh_output()` permet de récupérer un en-tête à partir d'un descripteur de tampon.
- L'en-tête matériel lui-même constitue le champ `hh_data[]`.

  La constante `LL_MAX_HEADER` est définie dans le fichier `linux/include/linux/netdevice.h` :

Code Linux 2.6.10

```
81 /*
82  *      Calcule la longueur de l'en-tete dans le pire cas en fonction des protocoles
73  *      utilises.
84  */
85
86 #if !defined(CONFIG_AX25) && !defined(CONFIG_AX25_MODULE) && !defined(CONFIG_TR)
87 #define LL_MAX_HEADER   32
88 #else
89 #if defined(CONFIG_AX25) || defined(CONFIG_AX25_MODULE)
90 #define LL_MAX_HEADER   96
91 #else
92 #define LL_MAX_HEADER   48
93 #endif
94 #endif
```

14.3 Implémentation de l'ordre réseau des octets

14.3.1 Implémentation générique

Code Linux 2.6.10

Les fonctions relatives à l'ordre réseau sont définies de façon générique dans le fichier `linux/include/linux/byteorder/generic.h`:

```
1   #ifndef _LINUX_BYTEORDER_GENERIC_H
2   #define _LINUX_BYTEORDER_GENERIC_H
3
4   /*
5    * linux/byteorder_generic.h
6    * Support pour reordonner les octets de facon generique
7    *
8    * Francois-Rene Rideau <fare@tunes.org> 19970707
9    *    a rassemble toutes les bonnes idees depuis tous les asm-foo/byteorder.h en un fichier,
10   *    les a nettoyes.
11   *    J'espere que c'est compatible avec les compilateurs non-GCC.
12   *    J'ai decide de mettre __BYTEORDER_HAS_U64__ dans byteorder.h,
13   *    puisque je n'etais pas sur que ce serait ok de le mettre dans types.h
14   *    A mis a jour pour 2.1.43
15   * Francois-Rene Rideau <fare@tunes.org> 19971012
16   *    A mis a jour pour 2.1.57
17   *    pour faire plaisir a Linus T., a remplace l'enorme #ifdef entre little/big endian
18   *    par des fichiers #include imbriques.
19   * Francois-Rene Rideau <fare@tunes.org> 19971205
20   *    Fait pour 2.1.71 ; maintenant cosmetique :
21   *    Mis les fichiers dans include/linux/byteorder/
22   *    Ventile swab du support generique.
23   *
24   * A FAIRE :
25   *   = Les mainteneurs reguliers du noyau devraient aussi remplacer toutes les macros
26   *    byteswap manuelles qui restent, disseminees parmi les pilotes,
27   *    apres quelques grep sur les sources...
28   *   = Linus peut vouloir renommer tous ces macros et fichiers a son gout,
29   *    pour correspondre a son schema de nommage personnel.
30   *   = il semble que quelques pilotes devraient aussi apprecier
31   *    le support d'echange des demi-octets...
32   *   = chaque architecture devrait ajouter sa macro byteswap dans asm/byteorder.h
33   *    voir comment quelques architectures l'ont deja fait (i386, alpha, ppc, etc)
34   *   = cpu_to_beXX et beXX_to_cpu pourraient un jour avoir besoin d'etre bien
35   *    distinguees dans le noyau. Ceci n'est pas le cas a l'heure actuelle,
36   *    puisque les machines petit-boutiennes, grand-boutiennes et pdp-boutiennes n'en ont pas
         besoin.
37   *    Mais ceci pourrait etre le cas pour, disons, un portage de Linux sur les
38   *    architectures a 20/21 bits (et Linux F21 en a besoin ?).
39   */
40
41  /*
42   * Les macros suivantes doivent etre definies dans <asm/byteorder.h>:
43   *
44   * Conversion de long et short int entre reseau et format hote
45   *      ntohl(__u32 x)
46   *      ntohs(__u16 x)
47   *      htonl(__u32 x)
48   *      htons(__u16 x)
49   * Il semble que quelques programmes (lesquels ? ou ? ou peut-etre un standard ? POSIX ?)
50   * pourraient aimer que celles-ci soient des fonctions, pas des macros (pourquoi ?).
51   * si c'est vrai, alors detecter-les et prenez les mesures necessaires.
52   * En tout cas, la mesure est : definir seulement ___ntohl en tant que macro,
53   * et dans un fichier separe, puis
54   * unsigned long inline ntohl(x){return ___ntohl(x);}
```

```
55   *
56   * De meme pour les arguments constants
57   *      __constant_ntohl(__u32 x)
58   *      __constant_ntohs(__u16 x)
59   *      __constant_htonl(__u32 x)
60   *      __constant_htons(__u16 x)
61   *
62   * Conversion des entiers XX-bits (16- 32- or 64-)
63   * entre un format de CPU natif et le format petit/grand boutien
64   * chose 64-bits seulement definie pour les architectures memes
65   *      cpu_to_[bl]eXX(__uXX x)
66   *      [bl]eXX_to_cpu(__uXX x)
67   *
68   * De meme, mais prendre un pointeur sur la valeur a convertir
69   *      cpu_to_[bl]eXXp(__uXX x)
70   *      [bl]eXX_to_cpup(__uXX x)
71   *
72   * De meme, mais changer in situ
73   *      cpu_to_[bl]eXXs(__uXX x)
74   *      [bl]eXX_to_cpus(__uXX x)
75   *
76   * Voir asm-foo/byteorder.h pour des exemples de comment fournir
77   * des versions optimisees pour l'architecture
78   *
79   */
80
81
82  #if defined(__KERNEL__)
83  /*
84   * a l'interieur du noyau nous pouvons utiliser des alias ;
85   * au dehors, nous devons eviter de polluer l'espace de noms POSIX...
86   */
87  #define cpu_to_le64 __cpu_to_le64
88  #define le64_to_cpu __le64_to_cpu
89  #define cpu_to_le32 __cpu_to_le32
90  #define le32_to_cpu __le32_to_cpu
91  #define cpu_to_le16 __cpu_to_le16
92  #define le16_to_cpu __le16_to_cpu
93  #define cpu_to_be64 __cpu_to_be64
94  #define be64_to_cpu __be64_to_cpu
95  #define cpu_to_be32 __cpu_to_be32
96  #define be32_to_cpu __be32_to_cpu
97  #define cpu_to_be16 __cpu_to_be16
98  #define be16_to_cpu __be16_to_cpu
99  #define cpu_to_le64p __cpu_to_le64p
100 #define le64_to_cpup __le64_to_cpup
101 #define cpu_to_le32p __cpu_to_le32p
102 #define le32_to_cpup __le32_to_cpup
103 #define cpu_to_le16p __cpu_to_le16p
104 #define le16_to_cpup __le16_to_cpup
105 #define cpu_to_be64p __cpu_to_be64p
106 #define be64_to_cpup __be64_to_cpup
107 #define cpu_to_be32p __cpu_to_be32p
108 #define be32_to_cpup __be32_to_cpup
109 #define cpu_to_be16p __cpu_to_be16p
110 #define be16_to_cpup __be16_to_cpup
111 #define cpu_to_le64s __cpu_to_le64s
112 #define le64_to_cpus __le64_to_cpus
113 #define cpu_to_le32s __cpu_to_le32s
114 #define le32_to_cpus __le32_to_cpus
115 #define cpu_to_le16s __cpu_to_le16s
116 #define le16_to_cpus __le16_to_cpus
```

```
117 #define cpu_to_be64s __cpu_to_be64s
118 #define be64_to_cpus __be64_to_cpus
119 #define cpu_to_be32s __cpu_to_be32s
120 #define be32_to_cpus __be32_to_cpus
121 #define cpu_to_be16s __cpu_to_be16s
122 #define be16_to_cpus __be16_to_cpus
123 #endif
124
125
126 #if defined(__KERNEL__)
127 /*
128  * Manipule ntohl et autres. Ceci a divers problemes de compatibilite -
129  * comme nous voulons donner le prototype avant meme que nous ayons
130  * une macro pour lui dans le cas ou quelque etrange programme
131  * veut prendre l'adresse de la chose ou de quelque chose..
132  *
133  * Noter que ceci utilisait un "long" a renvoyer dans libc5, alors que
134  * long est souvent 64 bits de nos jours... D'ou les conversions.
135  *
136  * On a besoin que ce soit des macros afin que tous les pliages
137  * soient corrects - si l'argument passe dans une fonction inline
138  * ce n'est plus constant avec gcc...
139  */
140
141 #undef ntohl
142 #undef ntohs
143 #undef htonl
144 #undef htons
145
146 /*
147  * Fait les prototypes. Quelqu'un peut vouloir prendre l'adresse
148  * ou quelque sottise de la sorte...
149  */
150 extern __u32                    ntohl(__be32);
151 extern __be32                   htonl(__u32);
152 extern __u16                    ntohs(__be16);
153 extern __be16                   htons(__u16);
154
155 #if defined(__GNUC__) && (__GNUC__ >= 2) && defined(__OPTIMIZE__)
156
157 #define ___htonl(x) __cpu_to_be32(x)
158 #define ___htons(x) __cpu_to_be16(x)
159 #define ___ntohl(x) __be32_to_cpu(x)
160 #define ___ntohs(x) __be16_to_cpu(x)
161
162 #define htonl(x) ___htonl(x)
163 #define ntohl(x) ___ntohl(x)
164 #define htons(x) ___htons(x)
165 #define ntohs(x) ___ntohs(x)
166
167 #endif /* OPTIMIZE */
168
169 #endif /* KERNEL */
170
171
172 #endif /* _LINUX_BYTEORDER_GENERIC_H */
```

14.3.2 Cas des microprocesseurs Intel

On se sert de plusieurs drapeaux pour connaître l'ordre adopté par l'architecture du microprocesseur, définis dans les fichiers en-tête `linux/include/linux/byteorder/little_endian.h`: Code Linux 2.6.10

```
1 #ifndef _LINUX_BYTEORDER_LITTLE_ENDIAN_H
2 #define _LINUX_BYTEORDER_LITTLE_ENDIAN_H
3
4 #ifndef __LITTLE_ENDIAN
5 #define __LITTLE_ENDIAN 1234
6 #endif
7 #ifndef __LITTLE_ENDIAN_BITFIELD
8 #define __LITTLE_ENDIAN_BITFIELD
9 #endif
```

et `linux/include/linux/byteorder/big_endian.h`: Code Linux 2.6.10

```
1 #ifndef _LINUX_BYTEORDER_BIG_ENDIAN_H
2 #define _LINUX_BYTEORDER_BIG_ENDIAN_H
3
4 #ifndef __BIG_ENDIAN
5 #define __BIG_ENDIAN 4321
6 #endif
7 #ifndef __BIG_ENDIAN_BITFIELD
8 #define __BIG_ENDIAN_BITFIELD
9 #endif
```

On se réfère enfin au fichier `linux/include/asm/byteorder.h`, par exemple au fichier `linux/include/asm-i386/byteorder.h` pour les microprocesseurs Intel: Code Linux 2.6.10

```
1  #ifndef _I386_BYTEORDER_H
2  #define _I386_BYTEORDER_H
3
4  #include <asm/types.h>
5  #include <linux/compiler.h>
6
7  #ifdef __GNUC__
8
9  /* Pour eviter bswap sur i386 */
10 #ifdef __KERNEL__
11 #include <linux/config.h>
12 #endif
13
14 static __inline__ __attribute_const__ __u32 ___arch__swab32(__u32 x)
15 {
16 #ifdef CONFIG_X86_BSWAP
17         __asm__("bswap %0" : "=r" (x) : "" (x));
18 #else
19         __asm__("xchgb %b0,%h0\n\t"     /* echange les octets bas   */
20                 "rorl $16,%0\n\t"       /* echange les mots         */
21                 "xchgb %b0,%h0"         /* echange les octets hauts */
22                 :"=q" (x)
23                 : "" (x));
24 #endif
25         return x;
26 }
27
28 static __inline__ __attribute_const__ __u64 ___arch__swab64(__u64 val)
29 {
30         union {
31                 struct { __u32 a,b; } s;
32                 __u64 u;
33         } v;
```

```
34         v.u = val;
35 #ifdef CONFIG_X86_BSWAP
36         asm("bswapl %0 ; bswapl %1 ; xchgl %0,%1"
37             : "=r" (v.s.a), "=r" (v.s.b)
38             : "" (v.s.a), "1" (v.s.b));
39 #else
40    v.s.a = ___arch__swab32(v.s.a);
41         v.s.b = ___arch__swab32(v.s.b);
42         asm("xchgl %0,%1" : "=r" (v.s.a), "=r" (v.s.b) : "" (v.s.a), "1" (v.s.b));
43 #endif
44         return v.u;
45 }
46
47 /* Ne pas definir swab16. Gcc est assez fin pour reconnaitre la version "C" et
48    les convertir en rotation ou echange.  */
49
50 #define __arch__swab64(x) ___arch__swab64(x)
51 #define __arch__swab32(x) ___arch__swab32(x)
52
53 #define __BYTEORDER_HAS_U64__
54
55 #endif /* __GNUC__ */
56
57 #include <linux/byteorder/little_endian.h>
58
59 #endif /* _I386_BYTEORDER_H */
```