Chapitre 11

Vue d'ensemble sur l'implémentation du sous-système réseau de Linux

Comme nous l'avons dit dans la préface, nous n'allons pas nous intéresser dans cet ouvrage au sous-système réseau de Linux dans son intégralité mais uniquement à l'envoi d'un datagramme UDP et à sa réception par un ordinateur d'extrémité.

La structuration en couches devrait nous permettre d'étudier l'envoi et la réception pour chacune des couches. L'expérience montre que ceci n'est pas pédagogique, au moins pour l'implémentation Linux.

La logique voudrait qu'on s'intéresse d'abord à l'envoi d'un datagramme UDP puis à sa réception. En fait l'expérience montre qu'il vaut mieux, toujours pour des raisons pédagogiques, le contraire.

11.1 Réception de données *via* un datagramme UDP

Lorsqu'un utilisateur envoie des données *via* le protocole UDP, celles-ci sont encapsulées dans un datagramme UDP, puis dans un paquet IP puis dans une trame. Nous supposerons que l'ordinateur récepteur est relié au réseau par une carte Ethernet.

11.1.1 Réception de la trame Ethernet par la carte réseau

La trame voyage sur le support physique à travers le réseau. Il y a trois possibilités :

- elle se perd lors de ses pérégrinations ;
- elle arrive corrompue sur la carte Ethernet du destinataire ;
- elle arrive dans son intégrité sur la carte Ethernet du destinataire.

Nous n'avons vu que le protocole Ethernet exige qu'un minuteur soit déclenché lors de l'émission d'une trame. Dans le premier cas, la machine émettrice ne recevra pas d'accusé de réception. Au bout d'un certain délai, elle enverra à nouveau la trame (en itérant un certain nombre de fois).

Dans le second cas, si ceci est détecté grâce au CRC, la carte réseau de la machine réceptrice écarte la trame. La machine émettrice ne recevra donc pas d'accusé de réception et on se retrouvera dans la situation du premier cas.

Dans le second cas, si ceci n'est pas détecté par le CRC, et dans le troisième cas, le contrôleur de la carte réseau compare l'adresse matérielle du destinataire de la trame avec celle de la carte réseau de la machine réceptrice sur laquelle est arrivée la trame. Si celles-ci concordent, il déclenche une interruption matérielle pour annoncer au microprocesseur de l'ordinateur qu'un paquet est disponible dans la mémoire de la carte réseau. Il laisse le pilote de cette carte agir à sa guise.

Si les deux adresses ne concordent pas, il y a trois cas dans lesquels le contrôleur de la carte a quand même un comportement analogue : dans le cas d'une trame de diffusion générale, dans celui d'une trame de multidiffusion dont l'adresse est acceptée par la carte et dans le cas d'un mode spécial, dit **mode de promiscuité**. Nous ne nous occuperons pas de ces cas dans cet ouvrage.

Le mode de promiscuité, comme son nom le laisse entendre, indique à la carte réseau d'accepter toutes les trames qui parviennent à celles-ci. Ceci est utile pour les **renifleurs**, qui analysent tout ce qui passe sur le réseau, soit dans un but avouable de contrôle, soit dans un but d'espionnage.

Dans les autres cas, la trame est rejetée par le contrôleur.

11.1.2 Récupération du paquet par l'ordinateur

Nous venons de voir que lorsqu'une trame arrive en bon état sur la carte réseau de l'ordinateur qui lui est destiné, le contrôleur de la carte déclenche une interruption. Le gestionnaire de cette interruption, qui est une partie du **pilote** de la carte réseau, alloue un emplacement en mémoire vive pour ce paquet, appelé **tampon de socket** (voir chapitres 12 et 13), effectue quelques vérifications sur ce paquet et, si elles sont concluantes, le place dans ce tampon de socket, ainsi qu'une partie de l'en-tête de trame, comprenant les adresses matérielles source et de destination ainsi que le type du paquet. Il alloue également un **descripteur de tampon de socket** contenant des informations sur celui-ci (début de son emplacement, sa taille, son heure d'arrivée...).

Suivant le type du paquet, le gestionnaire fait appel à une fonction de traitement de celui-ci, ayant pour seul paramètre l'adresse du descripteur de tampon de socket. Dans le cas d'un paquet IPv4, il s'agit de `ip_rcv()`.

11.1.3 Traitement du paquet par la couche réseau

Dans le cas d'une trame destinée à IPv4, la couche réseau traite le paquet grâce à la fonction `ip_rcv()`. Celle-ci vérifie d'abord s'il s'agit d'un paquet entier ou d'un sous-paquet dû à une fragmentation. Dans ce dernier cas, elle place le sous-paquet dans une file d'attente et déclenche un minuteur en attendant de recevoir tous les sous-paquets contituant le paquet originel. Si le délai s'écoule sans que tous les sous-paquets soient arrivés, elle détruit les sous-paquets arrivés se trouvant dans la file d'attente et envoie un message ICMP à la machine émettrice pour indiquer de renvoyer le paquet. Sinon elle réassemble les sous-paquets pour obtenir le paquet originel qui sera placé dans un seul tampon de socket puis démultiplexie la couche de transport grâce au champ concernant celle-ci de l'en-tête IP. Ceci consiste à transmettre le descripteur de tampon de socket à la fonction `udp_rcv()` dans le cas de UDP.

11.1.4 Traitement du datagramme par la couche de transport

Dans le cas de UDP, la couche de transport exécute une vérification minimale puis transmet le message à l'application concernée.

On commence par vérifier la longueur du datagramme. Si celle-ci est inférieure à celle d'un en-tête UDP, on se contente d'afficher un message noyau et de détruire le tampon de socket. Sinon, si la longueur du datagramme est supérieure à celle indiquée dans l'en-tête UDP, on se contente également d'afficher un message noyau et de détruire le tampon de socket. On calcule ensuite la somme de contrôle du datagramme. Si elle ne concorde pas avec celle spécifiée dans l'en-tête UDP, on se contente également d'afficher un message noyau et de détruire le tampon de socket.

Si toutes les vérifications sont concluantes, on place le tampon de socket (en fait un nouveau descripteur, celui de couche de transport) dans la file d'attente de réception des datagrammes UDP.

11.1.5 Récupération par l'utilisateur

L'utilisateur crée une socket de la façon vue au chapitre 6 et lit, dans le cas non connecté, à travers celle-ci grâce à la fonction `recvfrom()`. Celle-ci fait appel à la fonction de réception spécifique à la famille de protocoles, à savoir `sock_common_recvmsg()` dans le cas de IPv4. Cette dernière se contente de démultiplexer suivant la couche de transport : elle fait appel à la fonction `udp_recvmsg()` dans le cas de UDP.

La fonction `udp_recvmsg()` essaie de récupérer un tampon de socket dans la file d'attente de réception UDP, en faisant appel à `skb_recv_datagram()` et en attendant que le bon tampon de socket arrive s'il ne s'y trouve pas déjà. On lit alors le nombre de caractères voulu, que l'on place dans l'espace utilisateur.

11.2 Envoi de données *via* un datagramme UDP

11.2.1 Envoi par l'utilisateur

L'utilisateur crée une socket de la façon vue au chapitre 6 et envoie, dans le cas non connecté, à travers celle-ci grâce à la fonction `sendto()`. Celle-ci crée un message à partir des données, comprenant en particulier un en-tête de message, puis fait appel à la fonction `sock_sendmsg()`, elle-même faisant appel à la fonction `__sock_sendmsg()`, qui fait appel à la fonction d'envoi spécifique à la famille de protocoles, à savoir `inet_sendmsg()` dans le cas de IPv4. Cette dernière

attribue un numéro de port au message puis démultiplexie suivant la couche de transport : elle fait appel à la fonction `udp_sendmsg()` dans le cas de UDP.

11.2.2 Traitement du datagramme par la couche de transport

La fonction `udp_sendmsg()` commence par constituer le datagramme UDP à partir du message. Si la taille des données est supérieure à 65 535 octets, celles-ci ne peuvent pas tenir dans un datagramme ; on s'arrête donc purement et simplement. Il en est de même si on essaie d'envoyer des données urgentes (grâce à un positionnement des drapeaux), puisque ceci n'est pas prévu dans le cas de UDP. On récupère l'adresse et le numéro de port pour renseigner l'en-tête UDP. On essaie d'initialiser une entrée de cache de routage ; on est sûr d'y parvenir si la table de routage possède une adresse par défaut. On fait enfin appel à la fonction `udp_flush_pending_frames()` pour constituer le datagramme UDP puis à la fonction `udp_push_pending_frames()` pour envoyer le datagramme à la couche réseau. Il n'y a pas beoin de démultiplexer dans ce sens, puisque la couche réseau est nécessairement IPv4.

11.2.3 Envoi des paquets ordinaires sous IPv4

La première partie consiste à router le paquet grâce à la fonction `ip_route_output_flow()`, chargée de fournir une entrée de cache de routage. On remarquera que l'on a déjà parlé de cette action dans la sous-section précédente ; ceci est dû au fait que Linux ne tient pas compte des couches de façon stricte. L'entrée de cache de routage précise en particulier que le paquet devra être transmis, dans le cas d'un paquet ordinaire, à la fonction `ip_output()`.

La seconde étape consiste à constituer le paquet IP. Pour cela on essaie de récupérer le premier descripteur de tampon de socket de la file d'attente en écriture du descripteur de couche de transport passé en argument à la fonction `ip_push_pending_frames()`. On récupère ensuite tous les descripteurs de cette file d'attente et on les place dans la liste des fragments. On renseigne les champs d'un nouvel en-tête IP puis on fait appel à la fonction `ip_output()` déterminée précédemment.

La troisième étape consiste à transférer le paquet de la couche réseau à la couche inférieure. Pour cela, on fragment éventuellement le paquet, on positionne le type du paquet de la sous-couche 2b à `ETH_P_IP` et on transmet à la fonction `ip_finish_output2()`, qui fait appel à la fonction `dev_queue_xmit()` d'envoi d'une trame.

11.2.4 Envoi de la trame

La fonction `dev_queue_xmit()`, où `skb` désigne le descripteur de tampon de socket, indique à la carte réseau qu'il y a des données à envoyer et leur emplacement en mémoire vive. La carte s'occupe du reste, utilisant l'accès direct à la mémoire (DMA) pour récupérer les données. À la fin de l'émission, la carte lève une interruption pour prévenir le noyau que la trame a été envoyée avec succès ou que le délai s'est écoulé.

Les données consistent en un paquet de la couche réseau. La carte s'occupe d'encapsuler celui-ci pour obtenir une trame.

11.3 Historique

Une implémentation UUCP a tourné sous Linux presque dès le début. L'implémentation de TCP/IP a commencé en 1992 lorsque Ross Biro et d'autres ont créé ce qui fut ensuite connu sous le nom de **Net-1**.

Ross Biro ne s'occupa plus du développement de Linux à partir de mai 1993. Fred van Kempen commença alors à travailler sur une nouvelle implémentation, qui sera connue sous la dénomination de **Net-2**. La première version publique, Net-2d, apparut dans le noyau 0.99.10 de l'été 1993. Alan Cox commença a s'intéresser au projet, concevant Net-2Debugged, puis passa à la version **Net-3** pour Linux 1.2 et Linux 2.0. Le noyau 2.2 utilise `Net-4`.