→ 2.6.18, Ext4 et listes chaînées avec list_head

Matthieu Barthélemy et Éric Lacombe

EN DEUX MOTS **L'objectif de cette nouvelle rubrique est de vous informer sur les grandes lignes de l'actualité du noyau Linux : le dernier sorti, mais aussi la version en préparation et le décryptage de leurs principales nouveautés.**

Nous approfondirons aussi souvent que possible le sujet par quelques brèves expliquant une partie spécifique du noyau, une nouveauté méritant attention...

Bienvenue et bonne lecture !

De retour pour cette deuxième édition, nous vous proposons ce mois-ci un tour d'horizon des nouveautés apportées par le noyau 2.6.18 et non moins de trois brèves. La première explique le rôle des différentes branches de développement du noyau Linux, la suivante est centrée sur le futur de ext3 qui donnera le jour à ext4, et finalement une brève détaille le fonctionnement des listes doublement chaînées sous Linux. Ce numéro arrive après deux évènements relatifs au noyau Linux, s'étant déroulés en juillet à Ottawa (Canada) : le Kernel Summit, rendez-vous annuel des développeurs Linux, suivi du Linux Symposium, conférence ouverte à tous et traitant de thèmes divers très orientés noyau. Attendez-vous à retrouver certains des sujets abordés lors de ces conférences dans le Kernel Corner (KC pour les intimes) des mois qui suivent ;)

Série 2.6.18

Tout d'abord, la série 2.6.18 marque la suppression de devfs. En effet le système de fichier virtuel était noté « obsolète » depuis un moment. Son code a été retiré du noyau, au profit de solution *userland* (udev) se fondant sur les informations fournies par sysfs.

Une fonctionnalité propre aux systèmes temps réel a été ajoutée au noyau. Il s'agit de l'implémentation de verrou avec héritage de priorité. Expliquons brièvement en quoi cela consiste. Considérons trois processus : P1 de priorité supérieure à P3 et P2 entre ces deux. Imaginons que P3 ait pris le verrou sur lequel P1 est en attente. P1 ne pourra s'exécuter que lorsque P3 aura relâché le verrou. Cependant, si P2 (de priorité supérieure à P3) préempte P3, P1 sera bloqué tant que P2 n'aura pas fini son travail. Par conséquent il y aura ce que nous appelons une inversion de priorité entre P1 et P2. Pour pallier ce problème, une solution d'héritage de priorité a vu le jour. Il s'agit d'augmenter temporairement la priorité d'un processus tenant un verrou à celle du processus de plus haute priorité attendant sur ce même verrou. Attention cependant, il ne s'agit pas d'une nouvelle fonctionnalité fournie pour les développeurs du noyau (Linus s'y est fermement opposé : « *Friends don't let friends use priority inheritance. Just don't do it. If you really need it, your system is broken anyway.* » Traduction : « *Camarades ne laissez pas utiliser l'héritage de priorité par vos homologues. Ne le faites tout simplement pas. De toutes façons, si vous en avez vraiment besoin c'est que votre système est mal conçu.* »). Cette fonctionnalité est à la discrétion de l'espace utilisateur via les *futex*. Ces derniers sont des objets permettant une implémentation performante de tous les mécanismes de synchronisation (mutex, barrière mémoire, etc.) en espace utilisateur, tout en se déroulant le plus possible à ce niveau (i. e. sans intervention du noyau). Les nouvelles opérations ajoutées à l'appel système `futex()` sont `FUTEX_LOCK_PI` et `FUTEX_UNLOCK_PI` (`PI` pour *Priority Inheritance*). Ingo Molnar a même implémenté un principe de chaînage dans les futex. Si le processus héritant d'une priorité plus forte est bloqué à son tour sur un futex, cette priorité est automatiquement propagée au processus tenant ce second futex. Au niveau des fonctionnalités facilitant le *debuggage* du noyau, *lockdep* a fait son apparition. Il s'agit d'un mécanisme permettant de valider les modèles de prises de verrous dans le noyau, afin d'éviter les *deadlocks* (et *livelock*). Pour cela, une instrumentalisation du code relatif à la synchronisation est faite afin de pouvoir valider leur utilisation en cours d'exécution. Un cas typique de deadlock de la forme ABBA (où A et B sont deux verrous qui sont pris dans l'ordre inverse par deux processus différents) entraîne, avec cette fonctionnalité activée, un message d'avertissement avec les informations nécessaires à la correction du problème. Toutes les opérations de verrouillage sont interceptées par *lockdep* et un certain nombre de tests est effectué afin de détecter des problèmes de synchronisation. Un des tests met à profit un historique sur la prise de verrous. Il recense les verrous déjà tenus lorsqu'un nouveau est pris. Ainsi, lors de la prise d'un verrou, il vérifie s'il n'a pas rencontré une situation dans laquelle ce verrou était pris après ceux actuellement tenus. Le cas échéant, une violation dans l'ordre de prise des verrous est notifiée. Dans le domaine des système SMP (*Symmetric Multi-Processor*), une grosse amélioration a été faite au niveau du *load balancing* (répartition de charges sur les différents processeurs). Jusqu'à présent, chaque processeur possédait sa propre liste de processus à exécuter (*runqueue*), et l'ordonnanceur migrait des processus d'une runqueue à l'autre en regard de la longueur de ces runqueues et afin d'équilibrer la charge. Cependant, il est facile

de constater que sur un système SMP faisant tourner des processus de priorités différentes, cette gestion de charge peut mettre en défaut les processus de priorités fortes face aux faibles. En effet, le load balancing ne prenant pas en compte les priorités des processus, une situation dans laquelle un processeur ferait tourner deux processus de hautes priorités, alors qu'un autre en exécuterait deux de faibles priorités, est tout à fait envisageable. Pour pallier cette faiblesse, chaque processus se voit octroyer un poids (*load weight*) dépendant de sa priorité et permettant à l'ordonnanceur d'équilibrer les runqueues des processeurs d'une manière équitable. La gestion des systèmes NUMA (*Non Uniform Memory Access*) se voit également améliorée. Ces systèmes sont structurés en un réseau de nœuds, chacun possédant un unique bus mémoire et un ou plusieurs processeurs. Lorsqu'une page de données sur un nœud a besoin d'être accédée par un autre, il est nécessaire qu'un service de migration de pages entre nœuds soit disponible. La façon de procéder était alors de *swapper* la page en question afin de faire intervenir une faute de page sur le nœud souhaitant y accéder et de la charger alors dans la mémoire physique de ce nœud. Cette implémentation reste très coûteuse pour les raisons dont vous vous doutez. C'est pourquoi un mécanisme de migration directe de pages sans intervention du swap (*swapless page migration*) a été créé. Cependant, certaines situations empêchent le déplacement de pages, entraînant une dégradation des performances. Rendre sans échec l'opération du déplacement de pages dans une région quelconque reste un problème d'actualité.

Une amélioration notable pour l'architecture i386 rend possible le placement aléatoire dans l'espace d'adressage, de la page liée à l'accès aux appels système (appelée « i386 VDSO » *Virtual Dynamic Shared Object*) depuis l'espace utilisateur, laquelle résidait auparavant statiquement à l'avant-dernière page de l'espace d'adressage des processus (voir le KC du mois dernier pour plus de détails). Un des premiers bénéfices apportés par cette fonctionnalité est au niveau de la sécurité. Un attaquant pouvait profiter pour son attaque, notamment, d'une adresse ne variant jamais (quel que soit le système Linux i386 utilisant le VDSO) et pointant sur une instruction assembleur `RET ;)` D'autres bénéfices découlent de cette *randomisation* pour les systèmes virtualisés dans lesquels les hyperviseurs ou VMM (*Virtual Machine Monitor*) résident en général, tout comme le VDSO, en haut de l'espace d'adressage des processus et doivent gérer constamment les fautes de pages qu'entraînent les accès à la page du VDSO depuis l'espace utilisateur. Un comble, alors que ce VDSO est censé rendre moins lourd l'accès aux appels système. Cette randomisation est rendue possible en translatant ce VDSO dans un VMA (*Virtual Memory Area* : ce sont les régions mémoire d'un processus comme .text, .data, .bss, stack, heap, etc. référencées pour chaque processus). Lors de l'exécution d'un programme, un VMA est créé et contient le VDSO. L'adresse de ce VMA est passée à la `glibc` lorsqu'elle fait appel à `execve()` de façon à ce qu'elle puisse continuer d'effectuer les appels système pour le processus créé. Un dernier bénéfice lié à la création d'un VMA pour le VDSO est le fait de permettre aux *debuggers* de créer des *breakpoints* dans ce VMA (via le flag `MAYWRITE_VM` qui, ceci dit, ne semble pas compatible avec un système sous PaX avec les restrictions sur `mprotect()` activées).

PaX est un patch pour le noyau Linux. Un des mécanismes utilisés est de configurer chaque segment de la mémoire de sorte qu'il puisse être accédé soit en écriture, soit en exécution, mais jamais les deux. Ceci permet de limiter le risque d'exploitation d'un débordement de tampon (buffer overflow).

Du côté des LSM (*Linux Security Module*), quelques nouveaux *hooks* ont été ajoutés pour SELinux, lequel se voit également affecté par un certain nombre d'améliorations. La gestion des IRQ (interruptions en cours de travail du processeur utilisées pour dialoguer avec les périphériques) passe sous Linux par une couche générique pour la majorité des architectures supportées. L'API a été améliorée par de nouvelles fonctions, qui gèrent plus finement les IRQ en fonction de leur type (par front ou par niveau), et remplacent l'ancienne et unique fonction `__do_IRQ()`. Elle gère désormais l'architecture ARM, qui utilisait jusqu'alors du code spécifique.

Du côté des systèmes de fichiers, l'actualité est très riche, avec l'annonce de la création de l'ext3dev qui devrait s'appeler ext4, le salon *Linux FileSystem Workshop*, et la reprise cordiale des négociations pour l'inclusion de ReiserFS4 dans le noyau. En attendant que les passionnantes discussions et idées qui y ont été débattues se concrétisent, les systèmes de fichiers du noyau 2.6.18 subissent moins d'évolutions que ses deux prédécesseurs ; l'attention a été portée essentiellement sur CIFS avec le support de l'authentification NTLM v2 et LanMan (W95), NFS avec l'amélioration des transferts directs (*direct I/O*), et JFFS qui gère maintenant les attributs étendus comme les ACL *Posix*. Par contre, c'est l'ordonnanceur des entrées-sorties, et donc des accès disque, qui marque le changement le plus notable. De la même manière qu'il est possible de choisir entre plusieurs politiques d'ordonnancement de tâches sous Linux 2.6, on peut également utiliser le *I/O scheduler* de son choix. Désormais, c'est l'ordonnanceur d'entrées-sorties appelé *Complete Fair Queuing* qui est activé par défaut. Son grand intérêt est qu'il permet la prioritisation modifiable des accès : ainsi, à la manière des commandes `nice` ou `renice`, l'utilitaire `ionice` associé à ce scheduler modifie la priorité d'accès aux disques pour un processus. Il est ainsi possible, par exemple, de redonner un peu de réactivité à votre application qui tourne péniblement pendant que vous avez lancé un `find` sur la racine en diminuant l'`ionice` de ce dernier :

```
$ ionice -c2 -n7 -p `pidof find`
```

La gestion du SATA fait, quant à elle, un pas en avant ; plus robuste aux erreurs de transmission, elle gère désormais le *hotplug* (ajout à chaud) et la

Série 2.6.18

norme NCQ : plusieurs commandes sont envoyées simultanément au disque, qui décide en fonction de paramètres tels que la position des têtes sur le disque, un ordre optimal d'exécution. Le gain de performances serait significatif. La section réseau n'est pas en reste, avec l'apparition de deux nouveaux algorithmes. Le premier, TCP-LP (*Low Priority*), va dans le sens opposé des solutions déjà existantes, dans la mesure où il se destine à gérer, sur un lien, des flux réseau non prioritaires (*backups*, P2P...), en laissant la meilleure réactivité à des flux TCP classiques (applications d'entreprise, VoIP...) circulant conjointement. Plus intelligent que la simple QoS, il peut allouer à ces flux dits « basse priorité » de la bande passante supplémentaire en détectant le surplus laissé disponible. Le second, TCP Veno, se destine aux liaisons sans fil en y apportant une meilleure réaction face aux pertes de paquets. Pour résumer, il permet de meilleurs taux de transfert moyens, en détectant plus finement la cause de la perte, au contraire de l'algorithme TCP Reno, qui est le plus utilisé et a tendance a systématiquement diminuer le débit face à ce type d'événements. Quant à NetFilter (`iptables`), supportant le protocole H323 depuis Linux 2.6.17, il se voit ajouter la gestion de SIP (OpenWengo, Ekiga, MS-Windows Messenger...).

Passons maintenant aux habituels ajouts de drivers. Notre OS, désigné comme étant celui faisant fonctionner le plus de matériels au monde, nous apporte dans sa version 2.6.18 :

- Le support des périphériques Wifi équipés de la puce ZyDAS ZD1211, ceci concerne entre autres certains matériels de chez Acer, Olitec et ASUS. Il faut signaler que ce fabricant a publié les spécifications de ce matériel ainsi qu'un driver GPL.
- Le support de l'authentification WPA pour les puces Wifi BroadCom (*driver* bcm43xx) via la pile SoftMAC générique (cf. numéro précédent).
- Le fonctionnement du son sur les iMac G5 iSight et du *touchpad* pour les MacBook pro.

A noter également, le son HDA haute définition (24Kbits/195KHz et 7.1) fait son apparition pour de nombreux matériels.

Notre synthèse n'étant pas exhaustive, nous vous invitons à la compléter par les excellents **lwn.net** et **kernelnewbies.org** :-)

Les différentes branches de développement de Linux

Le développement du noyau a une organisation bien spécifique. Ainsi, si l'on prend l'exemple d'un nouveau *driver*, il faut savoir que ce n'est pas parce que vous postez le code sur la LKML qu'il sera inclus dans la prochaine mouture de votre OS préféré.

Le noyau Linux que vous utilisez est en fait un exemplaire provenant de la « branche principale », ayant éventuellement été quelque peu retouché par votre distribution. Avant d'arriver jusque-là, le code qui le compose est très probablement passé par plusieurs chemins.

En effet, ce qu'on appelle « le noyau » est une des différentes *branches* de l'*arbre de développement* ; celle que l'on dit « officielle ». Le code qui la compose a été testé et vérifié. Avant d'arriver jusque-là, si nous reprenons l'exemple du nouveau pilote de périphérique que vous avez développé, vous devrez le soumettre via la LKML, pour relecture et commentaires (sur l'efficacité, la syntaxe du code…). S'il satisfait les règles de codage en vigueur pour Linux, il sera vraisemblablement inclus dans une des différentes branches de développement. La plus générique est celle d'Andrew Morton, mainteneur officiel de la série 2.6 du noyau ; elle est basée sur la version en cours de la branche officielle, à laquelle sont ajoutés tous les patchs nécessitant d'être testés le plus possible avant candidature pour publication.

Ces patchs peuvent éventuellement être au préalable passés par une des autres branches de développement, plus spécifiques. Il existe par exemple des branches dédiées aux sous-systèmes SCSI, ACPI, *suspend*, Wifi… Les noyaux de la branche d'Andrew Morton portent le suffixe –mm.

Après chaque sortie d'une version de Linux, la branche officielle est ouverte pendant un intervalle de deux semaines pour inclusion des patchs pré-testés introduisant des nouveautés ou de nouvelles fonctionnalités. Passé ce délai, Linux version 2.6.x+1 est publié en rc1, et seules les corrections de bogues sont possibles sur votre nouveau pilote jusqu'à la prochaine « fenêtre d'ouverture ». Sortiront alors plusieurs autres –rc, sans délai prédéfini entre chacune, qui sont consacrées à la stabilisation du code.

Il n'y a pas non plus de délai fixe entre deux versions de Linux. En cas de bogues ou de failles de sécurité dans une version officielle, plusieurs versions mineures peuvent être publiées ; elles seront estampillées 2.6.x.y. Cette branche de post-correction est distincte et est suffixée –stable, et a été créée à l'initiative de Greg KH, Chris Wright et Adrian Bunk. Toute cette organisation du développement a été décidée et formalisée après le début des séries 2.6. Pour finir, il faut noter qu'il existe d'autres branches de développement, ayant des objectifs spécifiques, par exemple celle d'Ingo Molnar qui vise à fournir un noyau fonctionnant en mode temps réel (suffixe –rt).

Vers l'Ext4...

C'est sans doute la nouvelle qui a fait le plus de bruit cet été : la création (ou pas) d'une nouvelle série du système de fichiers ext (*Extended FileSystem*). Faisant suite à un long débat sur la LKML, la décision de créer une nouvelle branche du système de fichiers ext au lieu d'ajouter des fonctionnalités supplémentaires à ext3 a été prise, ceci pour plusieurs raisons :

- Éviter de complexifier encore le code actuel d'ext3, qui doit déjà en l'état adopter des comportements différents suivant le contexte (compatibilité ext2, fonctionnalités activées...) et présente donc de belles successions d'instructions conditionnelles.
- Ne pas introduire de bugs dans un système de fichiers stable et éprouvé. C'est entre autres Linus Torvalds qui a soulevé ce point : un utilisateur ou un développeur détesterait voir son système rendu instable ou crasher pour un problème de système de fichiers.

Mais quelles sont les évolutions du système de fichiers le plus répandu justifiant cette nouvelle branche ? Avant tout, l'ext4 sera un système de fichiers gérant des tailles de partitions et de fichiers bien supérieures à ext3. Celui-ci est en effet limité à 8 To par partition et 2 To maximum par fichier. Ce qui laisse le temps de voir venir sur les postes de bureau, mais peut d'ores et déjà être contraignant sur des systèmes ayant besoin de grosses capacités de stockage. Afin d'augmenter ces limites de capacité, plusieurs parties ayant trait à la gestion des systèmes de fichiers devront êtres modifiées sous Linux.

Petit résumé du fonctionnement d'un système de fichiers : prenons une partition formatée ext2/3. Toute son étendue est découpée en petits blocs (ensemble de secteurs du disque physique) de quelques Ko, chacun possédant une « adresse » permettant d'y accéder. Ce numéro de bloc est une donnée ayant une longueur finie. Ainsi sous Linux, il est codé sur un entier signé de 32 bits. On a donc un maximum de 2^{31} blocs (soit, avec des blocs de 4 Ko chacun, 2^{31}x4096=8,79x10^{12} octets, ou encore 8 To) adressables : voici donc notre premier élément bloquant. S'il est théoriquement envisageable d'augmenter la taille des blocs afin d'adresser plus d'espace total, cette solution n'est pas viable concrètement à cause de la perte de place que cela occasionne : sur un système (surtout Linux : penser à `/etc`...) contenant beaucoup de très petits fichiers, mettons de taille réelle < 1Ko, chacun d'entre eux occupe un bloc complet (donc 4 Ko par défaut en ext3).

La structure des *inodes* ext3, quant à elle, décrit l'espace occupé en blocs par un fichier dans un tableau de 15 pointeurs vers des index de blocs : les 12 premiers pointent sur les 12 premiers blocs du fichier (12x4=48Ko max). Si 12 blocs ne suffisent pas, le 13ème octet pointe sur un bloc spécifique (bloc indirect) contenant un tableau de 1024 adresses de blocs (4 Mo donc). Si le fichier pèse plus de 4144 Ko, le 14ème octet pointe sur un bloc contenant un tableau de ces blocs indirects, chacun contenant des adresses de blocs de données. Vous aurez peut-être deviné que le 15ème octet décrit un tableau de tableaux de blocs indirects ;-) Ainsi, on a une structure en arbre. On s'épargnera le calcul qui démontre qu'on peut arriver à décrire assez de blocs pour représenter au total 2 To, mais on retiendra que cette structure n'est pas forcément optimale pour les performances lors d'accès aux gros fichiers, et qu'elle représente notre deuxième limitation d'ext3.

L'ext4, qui sera appelé ext3dev le temps de sa maturation, sera basé sur un *fork* d'ext3 et pourrait selon des patchs proposés par Mingming Cao d'IBM décrire les numéros des blocs sur 48 bits (1024PB), et utiliser une nouvelle structure appelée « *extent* » qui fait en sorte de stocker les données en blocs contigus sur le disque. Ainsi un extent peut décrire une bonne partie voire tous les blocs qu'occupe un fichier simplement par deux données : son premier bloc et le nombre de blocs contigus, évitant ainsi de recourir á de nombreux pointeurs de blocs. Parallèlement au changement de la structure des inodes ext3, la couche de journalisation FS du noyau (JBD) *forkerait* également pour adopter le 64 bits et devenir commune à ext4 et OCFS2 (cf. Kernel Corner du numéro précédent). Le nouveau système de fichiers respectera une compatibilité ascendante, il sera possible de monter ses partitions ext3.

La réflexion sur l'ext4 ne s'arrête pas là, un long fil de discussion sur la LKML ouvrait le débat à l'ajout de fonctionnalités complémentaires ou inédites, telles que :

- les *snapshots* : image à chaud et en lecture seule d'un point de montage, qui ne contient que les métadonnées des fichiers (gain de place par rapport à une vraie copie), mais permet de garder les données de ce fichier en cas de suppression dans le FS original ; les blocs de données associés ne sont pas rendus disponibles tant que le snapshot les référençant existe (sous Solaris ZFS par exemple).
- l'annulation de l'effacement : pouvoir récupérer un fichier effacé, par exemple en le copiant dans un répertoire spécifique au lieu de le supprimer réellement ;
- l'effacement sécurisé : au lieu de rendre disponible l'inode pour écriture, comme le fait une suppression habituelle, effacer réellement les données associées (remplissage par des zéros, etc.) pour des raisons de sécurité et de confidentialité ;
- les sommes de contrôle (*checksums*) : pour chaque fichier, on calcule sa somme via un algorithme qui aboutit à une valeur unique, qui sera différente à la moindre altération du fichier.

Ext2 a été créé à l'origine par Rémy Card. Sa création fut très influencée par FFS (Berkeley Fast File System) qui était généralement utilisé sur les systèmes UNIX.

Vers l'Ext4...

On sait ainsi facilement si un fichier est cohérent ou corrompu.

Le *Linux File Systems Workshop*, salon qui s'est tenu en juin à Portalnd, USA, mériterait un article à lui tout seul, mais on en retiendra ici qu'une des préoccupations majeures qui y a été discutée traitait de la fiabilité de systèmes de fichiers toujours plus gros et de la difficulté de lancer une vérification sur les partitions de très grandes tailles. La taille des disques évoluant bien plus rapidement que leurs débits et temps d'accès, des opérations comme le `fsck` prennent de plus en plus de temps ; pour l'anecdote, il a fallu près d'une semaine pour lancer cet outil sur une partition du serveur de **kernel.org**. Ceci étant difficilement envisageable sur un serveur de production (pour rappel, un `fsck` se lance sur un FS non monté), la discussion a été lancée et a abouti au concept appelé « ChunkFS ». L'idée est de séparer un système de fichiers en plusieurs petits FS (*chunks*) ayant chacun un *dirty bit*, c'est-à-dire un bit qui à tout instant indique si le système de fichiers est dans un état cohérent (transactions finies, pas de vérification nécessaire même en cas de crash) ou pas (transactions non achevées, vérification à faire). Ainsi, en cas de problème sur notre gros FS, seuls les chunks (morceaux) marqués comme « sales » feront l'objet d'une vérification. Le concept Chunkfs réunit les travaux de Arjan van de Ven, développeur noyau et Valerie Henson, développeuse ZFS.

Toutes les fonctionnalités listées ci-dessus ne sont que des exemples ayant été débattus, rien de ce qui sera effectivement retenu n'a été tranché à l'heure actuelle.

Les listes doublement chaînées de Linux

Les listes doublement chaînées sont des structures de données basiques employées avec abondance à l'intérieur du noyau Linux. Elle sont utilisées, par exemple, pour lier les processus entre eux de multiples façons (relations de parenté, etc.). Bien que ce type de structure est un grand classique, la « façon Linux » de les aborder peut rebuter un programmeur habitué à ce qu'on lui a appris à l'école. Vous vous dites certainement « Hmhm... Comment une structure aussi classique pourrait elle me surprendre ? ».

Voyons cela à présent ;)

La définition sous Linux d'une liste doublement chaînée générique est fournie dans le fichier `include/linux/list.h`. En voici une copie :

```
struct list_head {
        struct list_head *next, *prev;
};
```

A première vue, nous nous disons qu'il manque un membre dans cette structure pour contenir les données à y stocker. Détrompez-vous, dans ce modèle, ce n'est pas la liste qui contient les données, mais plutôt les données qui l'embarquent dans leur structure !

Voyons comment cela fonctionne. Imaginons des structures d'un même type `T` que nous souhaitons lier entre elles. Pour cela, il nous faut embarquer à l'intérieur de `T` une structure `list_head`. Prenons l'exemple suivant :

```
struct T {
        int num;
        char *val;

        struct list_head liste_T;
};
```

Nous pouvons créer une liste via la macro `LIST_HEAD(name)` qui déclare une variable `list_head` du nom `name` et initialise ces champs avec son adresse. La figure 1 schématise le cas d'une liste à deux éléments.

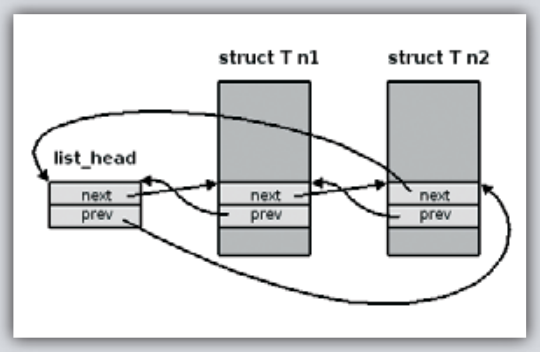

Fig. 1 : Liste doublement chaînée à deux éléments

La liaison de ces structures se fait grâce à leur membre `liste_T` (voir Fig. 1). Les opérations d'ajout, de retrait, etc. se font de manière basique sur les structures `list_head`. Le problème qui se pose est de savoir comment retrouver la structure `T` (ce qui nous intéresse principalement) lorsque nous nous déplaçons sur la liste. Linux userait-il de magie ?

En fait, pour résoudre ce problème, l'astuce est de soustraire à l'adresse du membre `liste_T` son offset au sein de la structure `T`. De cette façon, on retombe sur l'adresse du début de `T` ;) Intéressant, mais comment récupérer l'offset d'un membre à l'intérieur d'une structure ?! Voici la réponse dans cette macro (`include/linux/stddef.h`) :

```
#define offsetof(TYPE, MEMBER) ((size_t) &((TYPE *)0)->MEMBER)
```

Les listes doublement chaînées de Linux

Cela est compris par GCC de la façon suivante. Le chiffre 0 est considéré comme l'adresse d'une structure de type TYPE. La macro renvoie alors l'adresse qu'aurait le membre MEMBER. Étant donné que l'adresse de la structure est nulle, la macro renvoie en fait l'offset du membre dans cette structure.

La magie n'existe pas (comme le dirait un célèbre personnage du nom de Nakor d'un roman d'Heroic-Fantasy ;). Nous avons à présent tous les éléments pour comprendre comment récupérer une structure à partir d'une liste chaînée list_head. Voyons l'implémentation de la fonction renvoyant l'adresse d'une structure à partir de son type, du nom du membre embarquant le nœud list_head et de l'adresse de ce dernier (définition dans include/linux/kernel.h).

```
/**
 * container_of - cast a member of a structure out to the
containing structure
 * @ptr:        the pointer to the member.
 * @type:       the type of the container struct this is
                embedded in.
 * @member:     the name of the member within the struct.
 *
 */
#define container_of(ptr, type, member) ({
\
        const typeof( ((type *)0)->member ) *__mptr = (ptr);
\
        (type *)( (char *)__mptr - offsetof(type,member) );})
```

Nous avons ici, en quelque sorte, une primitive permettant d'élaborer les fonctions de type get d'une liste chaînée. La bibliothèque de liste définie dans include/linux/list.h propose la fonction list_entry correspondant exactement à container_of.

```
/**
 * list_entry - get the struct for this entry
 * @ptr:        the &struct list_head pointer.
 * @type:       the type of the struct this is embedded in.
 * @member:     the name of the list_struct within the struct.
 */
#define list_entry(ptr, type, member) \
        container_of(ptr, type, member)
```

De nombreuses fonctions sont définies dans la bibliothèque dont certaines utilisent, dans leur construction, la primitive list_entry. Il s'agit de la série des fonctions list_for_each_entry*() qui parcourent une liste dans son intégralité.

Ces listes permettent une grande souplesse dans leur utilisation, mais impliquent une plus grande attention pour le développeur. Il est parfois difficile (voire impossible) de déduire certaines liaisons entre structures sans regarder le code les liant.

Prenons un exemple illustrant cela. Au sein d'un descripteur de processus se trouve un champ menant à la liste des signaux qu'il a reçus et qui attendent d'être traités.

La structure qui permet l'accès à cette liste est de type sig_pending. Elle a pour premier champ une list_head dont le nom est list. A cette liste vient s'accrocher des structures de type sig_queue contenant les informations pour chaque signal reçu par le processus.

Le côté perturbant est que cette dernière structure possède également comme premier champ une list_head dont le nom est aussi list ;) Nous constatons donc une grande flexibilité dans l'utilisation de ces listes au risque par contre de s'y perdre quelque peu.

Outre le fait que ce type de listes soit générique à souhait, il introduit une facilité de gestion dans certains cas de figure. En effet, la construction particulière de ces listes permet de rebondir d'une liste à une autre de façon aisée lorsqu'un même type de structure est lié de multiples façons. Par exemple, prenons le cas où nous souhaitons déterminer l'ensemble des *threads* d'un même processus P.

Rappelons en premier lieu que Linux implémente un modèle *multithread* de 1 à 1. Une unité d'exécution sous Linux est représentée, entre autres, par son descripteur task_struct.

La notion de thread correspond à cette unité d'exécution et un processus correspond à une boîte virtuelle contenant des threads. Sous Linux, l'implémentation de ces boîtes se fait par la liaison (par une liste de type list_head) des task_struct des threads composant le processus. Il n'existe aucune structure sous Linux représentant directement la notion de processus.

Revenons à notre problème. Il nous faut tout d'abord parcourir la liste de tous les processus (i. e. de leur descripteur task_struct) qui est accessible via leur champ tasks. Une fois le processus P trouvé, nous allons chercher l'ensemble de ces threads. Précisons que l'identifiant au sens POSIX d'un processus correspond à l'identifiant PID (propre à chaque task_struct) du *Thread Group Leader* (le thread initial d'un processus).

La notion de *Thread Group* correspond à celle de processus POSIX. Le TGID (*Thread Group IDentifier*) de tous les threads de P est égal à ce PID en question. Comme nous l'avons dit, ils sont liés entre eux par une liste list_head, laquelle est accessible via leur membre pids[TGID].pid_list.

Par conséquent, il suffit de parcourir cette nouvelle liste, à partir du champ correspondant du descripteur de P, pour récupérer toutes les task_struct souhaitées.

Creative Commons

Paternité - Pas d'Utilisation Commerciale - Pas de Modification 2.0

Creative Commons n'est pas un cabinet d'avocats et ne fournit pas de services de conseil juridique. La distribution de la présente version de ce contrat ne crée aucune relation juridique entre les parties au contrat présenté ci-après et Creative Commons. Creative Commons fournit cette offre de contrat-type en l'état, à seule fin d'information. Creative Commons ne saurait être tenu responsable des éventuels préjudices résultant du contenu ou de l'utilisation de ce contrat.

Contrat

L'Oeuvre (telle que définie ci-dessous) est mise à disposition selon les termes du présent contrat appelé Contrat Public Creative Commons (dénommé ici « CPCC » ou « Contrat »). L'Oeuvre est protégée par le droit de la propriété littéraire et artistique (droit d'auteur, droits voisins, droits des producteurs de bases de données) ou toute autre loi applicable. Toute utilisation de l'Oeuvre autrement qu'explicitement autorisée selon ce Contrat ou le droit applicable est interdite.

L'exercice sur l'Oeuvre de tout droit proposé par le présent contrat vaut acceptation de celui-ci. Selon les termes et les obligations du présent contrat, la partie Offrante propose à la partie Acceptante l'exercice de certains droits présentés ci-après, et l'Acceptant en approuve les termes et conditions d'utilisation.

1. Définitions

a. **« Oeuvre »** : oeuvre de l'esprit protégeable par le droit de la propriété littéraire et artistique ou toute loi applicable et qui est mise à disposition selon les termes du présent Contrat.
b. **« Oeuvre dite Collective »** : une oeuvre dans laquelle l'oeuvre, dans sa forme intégrale et non modifiée, est assemblée en un ensemble collectif avec d'autres contributions qui constituent en elles-mêmes des oeuvres séparées et indépendantes. Constituent notamment des Oeuvres dites Collectives les publications périodiques, les anthologies ou les encyclopédies. Aux termes de la présente autorisation, une oeuvre qui constitue une Oeuvre dite Collective ne sera pas considérée comme une Oeuvre dite Dérivée (telle que définie ci-après).
c. **« Oeuvre dite Dérivée »** : une oeuvre créée soit à partir de l'Oeuvre seule, soit à partir de l'Oeuvre et d'autres oeuvres préexistantes. Constituent notamment des Oeuvres dites Dérivées les traductions, les arrangements musicaux, les adaptations théâtrales, littéraires ou cinématographiques, les enregistrements sonores, les reproductions par un art ou un procédé quelconque, les résumés, ou toute autre forme sous laquelle l'Oeuvre puisse être remaniée, modifiée, transformée ou adaptée, à l'exception d'une oeuvre qui constitue une Oeuvre dite Collective. Une Oeuvre dite Collective ne sera pas considérée comme une Oeuvre dite Dérivée aux termes du présent Contrat. Dans le cas où l'Oeuvre serait une composition musicale ou un enregistrement sonore, la synchronisation de l'oeuvre avec une image animée sera considérée comme une Oeuvre dite Dérivée pour les propos de ce Contrat.
d. **« Auteur original »** : la ou les personnes physiques qui ont créé l'Oeuvre.
e. **« Offrant »** : la ou les personne(s) physique(s) ou morale(s) qui proposent la mise à disposition de l'Oeuvre selon les termes du présent Contrat.
f. **« Acceptant »** : la personne physique ou morale qui accepte le présent contrat et exerce des droits sans en avoir violé les termes au préalable ou qui a reçu l'autorisation expresse de l'Offrant d'exercer des droits dans le cadre du présent contrat malgré une précédente violation de ce contrat.

2. Exceptions aux droits exclusifs. Aucune disposition de ce contrat n'a pour intention de réduire, limiter ou restreindre les prérogatives issues des exceptions aux droits, de l'épuisement des droits ou d'autres limitations aux droits exclusifs des ayants droit selon le droit de la propriété littéraire et artistique ou les autres lois applicables.

3. Autorisation. Soumis aux termes et conditions définis dans cette autorisation, et ceci pendant toute la durée de protection de l'Oeuvre par le droit de la propriété littéraire et artistique ou le droit applicable, l'Offrant accorde à l'Acceptant l'autorisation mondiale d'exercer à titre gratuit et non exclusif les droits suivants :

a. reproduire l'Oeuvre, incorporer l'Oeuvre dans une ou plusieurs Oeuvres dites Collectives et reproduire l'Oeuvre telle qu'incorporée dans lesdites Oeuvres dites Collectives;
b. distribuer des exemplaires ou enregistrements, présenter, représenter ou communiquer l'Oeuvre au public par tout procédé technique, y compris incorporée dans des Oeuvres Collectives;
c. lorsque l'Oeuvre est une base de données, extraire et réutiliser des parties substantielles de l'Oeuvre.

Les droits mentionnés ci-dessus peuvent être exercés sur tous les supports, médias, procédés techniques et formats. Les droits ci-dessus incluent le droit d'effectuer les modifications nécessaires techniquement à l'exercice des droits dans d'autres formats et procédés techniques. L'exercice de tous les droits qui ne sont pas expressément autorisés par l'Offrant ou dont il n'aurait pas la gestion demeure réservé, notamment les mécanismes de gestion collective obligatoire applicables décrits à l'article 4(d).

4. Restrictions. L'autorisation accordée par l'article 3 est expressément assujettie et limitée par le respect des restrictions suivantes :

a. L'Acceptant peut reproduire, distribuer, représenter ou communiquer au public l'Oeuvre y compris par voie numérique uniquement selon les termes de ce Contrat. L'Acceptant doit inclure une copie ou l'adresse Internet (Identifiant Uniforme de Ressource) du présent Contrat à toute reproduction ou enregistrement de l'Oeuvre que l'Acceptant distribue, représente ou communique au public y compris par voie numérique. L'Acceptant ne peut pas offrir ou imposer de conditions d'utilisation de l'Oeuvre qui altèrent ou restreignent les termes du présent Contrat ou l'exercice des droits qui y sont accordés au bénéficiaire. L'Acceptant ne peut pas céder de droits sur l'Oeuvre. L'Acceptant doit conserver intactes toutes les informations qui renvoient à ce Contrat et à l'exonération de responsabilité. L'Acceptant ne peut pas reproduire, distribuer, représenter ou communiquer au public l'Oeuvre, y compris par voie numérique, en utilisant une mesure technique de contrôle d'accès ou de contrôle d'utilisation qui serait contradictoire avec les termes de cet Accord contractuel. Les mentions ci-dessus s'appliquent à l'Oeuvre telle qu'incorporée dans une Oeuvre dite Collective, mais, en dehors de l'Oeuvre en elle-même, ne soumettent pas l'Oeuvre dite Collective, aux termes du présent Contrat. Si l'Acceptant crée une Oeuvre dite Collective, à la demande de tout Offrant, il devra, dans la mesure du possible, retirer de l'Oeuvre dite Collective toute référence au dit Offrant, comme demandé. Si l'Acceptant crée une Oeuvre dite Collective, à la demande de tout Auteur, il devra, dans la mesure du possible, retirer de l'Oeuvre dite Collective toute référence au dit Auteur, comme demandé.

b. L'Acceptant ne peut exercer aucun des droits conférés par l'article 3 avec l'intention ou l'objectif d'obtenir un profit commercial ou une compensation financière personnelle. L'échange de l'Oeuvre avec d'autres Oeuvres protégées par le droit de la propriété littéraire et artistique par le partage électronique de fichiers, ou par tout autre moyen, n'est pas considéré comme un échange avec l'intention ou l'objectif d'un profit commercial ou d'une compensation financière personnelle, dans la mesure où aucun paiement ou compensation financière n'intervient en relation avec l'échange d'Oeuvres protégées.
c. Si l'Acceptant reproduit, distribue, représente ou communique l'Oeuvre au public, y compris par voie numérique, il doit conserver intactes toutes les informations sur le régime des droits et en attribuer la paternité à l'Auteur Original, de manière raisonnable au regard au médium ou au moyen utilisé. Il doit communiquer le nom de l'Auteur Original ou son éventuel pseudonyme s'il est indiqué ; le titre de l'Oeuvre Originale s'il est indiqué ; dans la mesure du possible, l'adresse Internet ou Identifiant Uniforme de Ressource (URI), s'il existe, spécifié par l'Offrant comme associé à l'Oeuvre, à moins que cette adresse ne renvoie pas aux informations légales (paternité et conditions d'utilisation de l'Oeuvre). Ces obligations d'attribution de paternité doivent être exécutées de manière raisonnable. Cependant, dans le cas d'une Oeuvre dite Collective, ces informations doivent, au minimum, apparaître à la place et de manière aussi visible que celles à laquelle apparaissent les informations de même nature.
d. Dans le cas où une utilisation de l'Oeuvre serait soumise à un régime légal de gestion collective obligatoire, l'Offrant se réserve le droit exclusif de collecter ces redevances par l'intermédiaire de la société de perception et de répartition des droits compétente. Sont notamment concernés la radiodiffusion et la communication dans un lieu public de phonogrammes publiés à des fins de commerce, certains cas de retransmission par câble et satellite, la copie privée d'Oeuvres fixées sur phonogrammes ou vidéogrammes, la reproduction par reprographie.

5. Garantie et exonération de responsabilité

a. En mettant l'Oeuvre à la disposition du public selon les termes de ce Contrat, l'Offrant déclare de bonne foi qu'à sa connaissance et dans les limites d'une enquête raisonnable :
    i. L'Offrant a obtenu tous les droits sur l'Oeuvre nécessaires pour pouvoir autoriser l'exercice des droits accordés par le présent Contrat, et permettre la jouissance paisible et l'exercice licite de ces droits, ceci sans que l'Acceptant n'ait aucune obligation de verser de rémunération ou tout autre paiement ou droits, dans la limite des mécanismes de gestion collective obligatoire applicables décrits à l'article 4(e);
b. L'Oeuvre n'est constitutive ni d'une violation des droits de tiers, notamment du droit de la propriété littéraire et artistique, du droit des marques, du droit de l'information, du droit civil ou de tout autre droit, ni de diffamation, de violation de la vie privée ou de tout autre préjudice délictuel à l'égard de toute tierce partie.
c. A l'exception des situations expressément mentionnées dans le présent Contrat ou dans un autre accord écrit, ou exigées par la loi applicable, l'Oeuvre est mise à disposition en l'état sans garantie d'aucune sorte, qu'elle soit expresse ou tacite, y compris à l'égard du contenu ou de l'exactitude de l'Oeuvre.

6. Limitation de responsabilité. A l'exception des garanties d'ordre public imposées par la loi applicable et des réparations imposées par le régime de la responsabilité vis-à-vis d'un tiers en raison de la violation des garanties prévues par l'article 5 du présent contrat, l'Offrant ne sera en aucun cas tenu responsable vis-à-vis de l'Acceptant, sur la base d'aucune théorie légale ni en raison d'aucun préjudice direct, indirect, matériel ou moral, résultant de l'exécution du présent Contrat ou de l'utilisation de l'Oeuvre, y compris dans l'hypothèse où l'Offrant avait connaissance de la possible existence d'un tel préjudice.

7. Résiliation

a. Tout manquement aux termes du contrat par l'Acceptant entraîne la résiliation automatique du Contrat et la fin des droits qui en découlent. Cependant, le contrat conserve ses effets envers les personnes physiques ou morales qui ont reçu de la part de l'Acceptant, en exécution du présent contrat, la mise à disposition d'Oeuvres dites Dérivées, ou d'Oeuvres dites Collectives, ceci tant qu'elles respectent pleinement leurs obligations. Les sections 1, 2, 5, 6 et 7 du contrat continuent à s'appliquer après la résiliation de celui-ci.
b. Dans les limites indiquées ci-dessus, le présent Contrat s'applique pendant toute la durée de protection de l'Oeuvre selon le droit applicable. Néanmoins, l'Offrant se réserve à tout moment le droit d'exploiter l'Oeuvre sous des conditions contractuelles différentes, ou d'en cesser la diffusion; cependant, le recours à cette option ne doit pas conduire à retirer les effets du présent Contrat (ou de tout contrat qui a été ou doit être accordé selon les termes de ce Contrat), et ce Contrat continuera à s'appliquer dans tous ses effets jusqu'à ce que sa résiliation intervienne dans les conditions décrites ci-dessus.

8. Divers

a. A chaque reproduction ou communication au public par voie numérique de l'Oeuvre ou d'une Oeuvre dite Collective par l'Acceptant, l'Offrant propose au bénéficiaire une offre de mise à disposition de l'Oeuvre dans des termes et conditions identiques à ceux accordés à la partie Acceptante dans le présent Contrat.
b. La nullité ou l'inapplicabilité d'une quelconque disposition de ce Contrat au regard de la loi applicable n'affecte pas celle des autres dispositions qui resteront pleinement valides et applicables. Sans action additionnelle par les parties à cet accord, lesdites dispositions devront être interprétées dans la mesure minimum nécessaire à leur validité et leur applicabilité.
c. Aucune limite, renonciation ou modification des termes ou dispositions du présent Contrat ne pourra être acceptée sans le consentement écrit et signé de la partie compétente.
d. Ce Contrat constitue le seul accord entre les parties à propos de l'Oeuvre mise ici à disposition. Il n'existe aucun élément annexe, accord supplémentaire ou mandat portant sur cette Oeuvre en dehors des éléments mentionnés ici. L'Offrant ne sera tenu par aucune disposition supplémentaire qui pourrait apparaître dans une quelconque communication en provenance de l'Acceptant. Ce Contrat ne peut être modifié sans l'accord mutuel écrit de l'Offrant et de l'Acceptant.
e. Le droit applicable est le droit français.

Creative Commons n'est pas partie à ce Contrat et n'offre aucune forme de garantie relative à l'Oeuvre. Creative Commons décline toute responsabilité à l'égard de l'Acceptant ou de toute autre partie, quel que soit le fondement légal de cette responsabilité et quel que soit le préjudice subi, direct, indirect, matériel ou moral, qui surviendrait en rapport avec le présent Contrat. Cependant, si Creative Commons s'est expressément identifié comme Offrant pour mettre une Oeuvre à disposition selon les termes de ce Contrat, Creative Commons jouira de tous les droits et obligations d'un Offrant.

A l'exception des fins limitées à informer le public que l'Oeuvre est mise à disposition sous CPCC, aucune des parties n'utilisera la marque « Creative Commons » ou toute autre indication ou logo afférent sans le consentement préalable écrit de Creative Commons. Toute utilisation autorisée devra être effectuée en conformité avec les lignes directrices de Creative Commons à jour au moment de l'utilisation, telles qu'elles sont disponibles sur son site Internet ou sur simple demande.

Creative Commons peut être contacté à http://creativecommons.org/.