Les améliorations du noyau Linux 2.6

par Serge Lefranc(1)

Depuis 1991, le système GNU/Linux n'a cessé d'évoluer pour arriver à un niveau de maturité lui permettant aujourd'hui de rivaliser avec les Unix propriétaires. Afin de bien comprendre le rôle de ces évolutions, un rappel de l'histoire de Linux introduira une partie plus technique sur les modifications majeures du noyau 2.6.
Ceci sera ensuite illustré par une comparaison réalisée par Rusty Russel entre un noyau 2.4.18 et 2.6.0. Les résultats obtenus nous permettront de conclure sur la possibilité d'utiliser Linux sur un serveur.

The Linux kernel project was started in 1991 by Linus Torvalds. It was integrated in the GNU project which gives birth to the GNU/Linux operating system. Since then, this O.S. has never stop evolving to a level that enables it to compete against corporate Unix in the server field, and against Microsoft Windows in the desktop field.
In order to better understand the role played by this different evolutions, a short history of the main Linux kernel revisions will allow us to introduce more technical parts about some of the fundamental modifications of the new Linux kernel 2.6 in term of performances.
This technical part will be illustrated by a benchmark conducted by Rusty Russel in order to show the differences of performances between a Linux kernel 2.4.18 and a 2.6 one. The results of this study will give us enough informations to conclude about the possible use we can make of this new version of Linux server.

Depuis 1991, le noyau Linux et son système d'exploitation, GNU/Linux, n'ont cessé d'évoluer pour arriver à un niveau de maturité tel que cet ensemble est aujourd'hui en mesure de rivaliser avec les autres systèmes de type Unix dans le domaine des serveurs, et, dans une moindre mesure, avec Windows dans le domaine des postes de travail.

Afin de mieux comprendre le rôle joué par ces évolutions, un rappel de l'histoire de Linux et de ses versions les plus marquantes permet d'introduire une partie plus technique sur les modifications majeures du noyau 2.6 en terme d'amélioration de performances.

Cette partie technique est illustrée par une comparaison réalisée par Rusty Russel entre les performances du noyau 2.4.18 et du noyau 2.6. Les résultats de cette étude fournissent les éléments nécessaires pour conclure et mettre en perspective ces résultats dans le cadre de l'utilisation de Linux sur un serveur.

Histoire du noyau Linux

Les origines

Le Finlandais Linus Torvalds a créé le noyau Linux en 1991 comme une variante du système d'exploitation Minix pour ordinateur Intel x86. Linux est basé sur une

(1) Ingénieur de l'Armement, Expert en Architecture et en Sécurité des Systèmes d'Information – DGA/DCE/ETC5/CELAR

architecture de type Unix et, dès cette époque, Linux disposait déjà d'une cinquantaine d'appels système offrant les fonctionnalités de base : gestion des processus, de la mémoire et des fichiers.

La première distribution officielle de Linux (version 1.0) date de mars 1994, elle supportait uniquement le processeur i386 et les machines mono-processeur. On peut considérer que la structure générale des noyaux actuels était déjà présente dans ce noyau. A partir de cette version, Linus Torvalds a également décidé de gérer simultanément deux versions du noyau en circulation : l'une avec un numéro mineur pair, correspondant à une version stable, et l'autre avec un numéro mineur impair correspondant à une version en cours de développement (noyau stable : 2.2.17 : numéro majeur 2, numéro mineur 2 et numéro de révision 17).

Linux 1.2 est apparu en mars 1995. C'était la première version à offrir un support pour d'autres types de processeurs (Alpha, Sparc, 68000, MIPS…) et à permettre le chargement de modules dans le noyau. Il devenait donc possible de compiler des pilotes de périphériques sous forme de fichiers externes et de les insérer dans la mémoire du noyau en fonctionnement, sans recompilation du noyau.

Linux 2.0 est arrivé en juin 1996 et fait suite au noyau de développement 1.3 dont les modifications profondes ont amené Linus Torvalds à faire évoluer le numéro majeur de version. Il incluait une nouvelle fois le support de nouveaux processeurs (dont le PowerPC), mais fut surtout la première révision à supporter les machines multiprocesseurs (*Symetric Multi Processing, SMP*) et à adopter pour mascotte le désormais célèbre pingouin…

La version 2.2

Le noyau Linux 2.2 est apparu en janvier 1999. Son développement fut très long (plus de deux ans et demi). De nombreuses fonctionnalités ont été ajoutées, concernant notamment la norme Posix.1b (relative au temps réel) et Posix.1c (concernant les *threads*). L'autre amélioration de taille fut le passage à la bibliothèque GlibC en remplacement des bibliothèques C spécifiques à Linux qui ne permettaient pas une homogénéisation de l'architecture de programmation. Enfin, d'autres changements ont été apportés au noyau, notamment une amélioration des performances sur les machines multiprocesseurs, ainsi que le support d'un éventail de matériels encore plus important.

C'est à partir de ce moment que Linux a commencé à se démocratiser auprès du grand public, en particulier par l'apparition de distributions (RedHat, Mandrake…) en facilitant l'installation et bénéficiant de nombreuses fonctions multimédias.

La version 2.4

Le noyau Linux 2.4 est sorti en janvier 2001. Il incorporait une nouvelle fois un certain nombre d'améliorations cruciales. La gestion de la mémoire virtuelle fut optimisée pour éviter les copies inutiles de zones mémoires. Les performances furent également améliorées (au niveau du support multiprocesseurs) et un *firewall* fonctionnant en mode *statefull* fut incorporé au noyau (il existait déjà dans les versions libres de l'Unix BSD). Le système de fichier journalisé Ext3 fut ajouté dans le noyau 2.4.15, et améliora le fonctionnement des machines demandant un taux de disponibilité élevé. Comme pour la précédente version, le support de nouveaux périphériques bureautiques fut ajouté (USB, IEEE 1394, tuners TV, webcam…).

Cependant, un certain nombre de problèmes apparurent lorsque ce noyau fut déployé à grande échelle. Les premiers concernèrent la gestion de la mémoire virtuelle. Ce fut corrigé dans la version 2.4.11, mais ensuite d'autres instabilités apparurent à cause de la fonction symlink() qui corrompait les inodes du système de fichiers. Encore une fois, ce fut réparé dans la version 2.4.15, mais un autre problème de corruption du système de fichiers obligea les concepteurs à sortir la version 2.4.16 une journée après la version 2.4.15!

La version 2.6

Le noyau 2.4 n'a jamais brillé par ses caractéristiques temps réel. Le temps de réponse séparant l'arrivée d'une interruption matérielle du lancement de la tâche associée pouvait dépasser la centaine de millisecondes. Conscient du problème, le noyau 2.6 a été profondément modifié afin d'offrir des performances temps réel nettement améliorées. Pour arriver à ce résultat, deux modifications notables ont été apportées.

Le noyau a été rendu préemptif. En effet, dans la précédente version, aucun réordonnancement des tâches n'était possible tant que le noyau n'avait pas terminé un traitement associé à un appel système ou achevé d'autres types de fonctions internes. Il était alors impossible de prédire à quel moment une tâche de plus haute priorité pouvait être effectivement exécutée. Pour réduire cet inconvénient, des points de préemption (« hooks ») ont donc été incorporés dans le noyau 2.6. Ce sont des instructions qui permettent au noyau de s'interrompre afin d'exécuter un processus de plus haute priorité.

La deuxième amélioration concerne l'ordonnanceur. Il a été entièrement réécrit par Ingo Molnar afin d'éliminer les lenteurs de l'algorithme d'ordonnancement des précédents noyaux. Cela lui permet de mieux tenir la charge et, surtout, d'améliorer le fonctionnement du noyau sur des machines à plusieurs processeurs (tous les nouveaux processeurs Intel sont multicores). Quel que soit le nombre de processus, l'ordonnanceur met toujours le même temps pour prendre sa décision, il fonctionne donc en _(1). Par contre, l'algorithme reste en _(n) par rapport au nombre n de processeurs.

Afin de bien évaluer l'impact de ces améliorations et de leurs implications dans l'utilisation possible du noyau 2.6 dans les systèmes d'information des programmes d'armement, nous allons détailler ces évolutions dans le paragraphe suivant.

Analyse de quelques nouveautés du noyau Linux 2.6

Il paraît utile dans un premier temps de définir les termes de *thread*, processus, ordonnancement et préemption.

Un processus est un programme en exécution. C'est l'unité de travail dans un système en temps partagé. Un *thread*, également appelé processus léger, est une unité de base d'utilisation du processeur. Il partage avec d'autres *threads* appartenant au même processus sa section de code, sa section de données et d'autres ressources du système d'exploitation (tels que les fichiers ouverts et les signaux). Puisque le processus possède des flots de contrôles multiples, il peut effectuer plusieurs tâches a la fois.

Le ***schéma suivant*** illustre la différence entre processus à flot unique (monoflot ou monothread) et processus à flots multiples (multithreads ou multiflots).

L'objectif de la multiprogrammation est de disposer toujours de quelques processus en exécution pour maximiser l'utilisation du processeur. Afin de ne pas permettre à un processus trop gourmand en ressources de s'accaparer le processeur, la notion de temps partagé est utilisée. L'objectif du temps partagé est de commuter le processeur entre les processus si fréquemment que les utilisateurs peuvent interagir avec chaque programme pendant qu'il s'exécute. Sur un système monoprocesseur il n'y aura jamais plus d'un processus en exécution. C'est l'ordonnanceur qui est responsable de la commutation du processeur entre tous les processus.

Les décisions d'ordonnancement peuvent avoir lieu dans les quatre circonstances suivantes :
- lorsque le processus passe de **l'état d'exécution à l'état d'attente** (à cause d'une requête d'entrée/sortie par exemple) ;
- lorsque le processus passe de **l'état d'exécution à l'état prêt** (si une interruption est levée par exemple) ;
- lorsque le processus passe de **l'état d'attente à l'état prêt** (à la fin d'une requête d'entrée/sortie par exemple) ;
- lorsque le processus **se termine**.

Lorsque l'ordonnancement n'a lieu que dans le premier et dernier cas, on dit que le schéma d'ordonnancement est non préemptif (ou coopératif). Dans les autre cas, on dit que le schéma est préemptif. Avec l'ordonnancement non préemptif, le processus auquel l'unité centrale a été allouée en conserve l'exclusivité jusqu'à ce qu'il libère l'unité centrale, en se terminant ou en passant à l'état d'attente.

L'ordonnanceur de processus

Nous avons mentionné précédemment que l'ordonnanceur du noyau 2.6 avait été réécrit par Ingo Molnar et qu'il était bien plus performant que celui du noyau 2.4. Nous allons montrer pourquoi.

Dans l'ordonnanceur du noyau 2.4, le calcul d'attribution de tranches de temps processeur pour chaque processus était relancé lorsque tous les processus étaient arrivés à la fin de leur tranche de temps. Sur un système multiprocesseur, les processeurs étaient inactifs (« *idle* ») jusqu'à ce que tous les processus aient fini leur tranche de temps, ce qui avait un impact non négligeable sur les performances. De plus, durant la phase d'inactivité, les processeurs exécutaient des processus d'attente dont la tranche de temps n'étaient pas encore expirée (par exemple, lorsqu'un processus occupait un processeur et que l'autre processeur se mettait à exécuter un processus d'attente). Il pouvait arriver que lorsque le premier

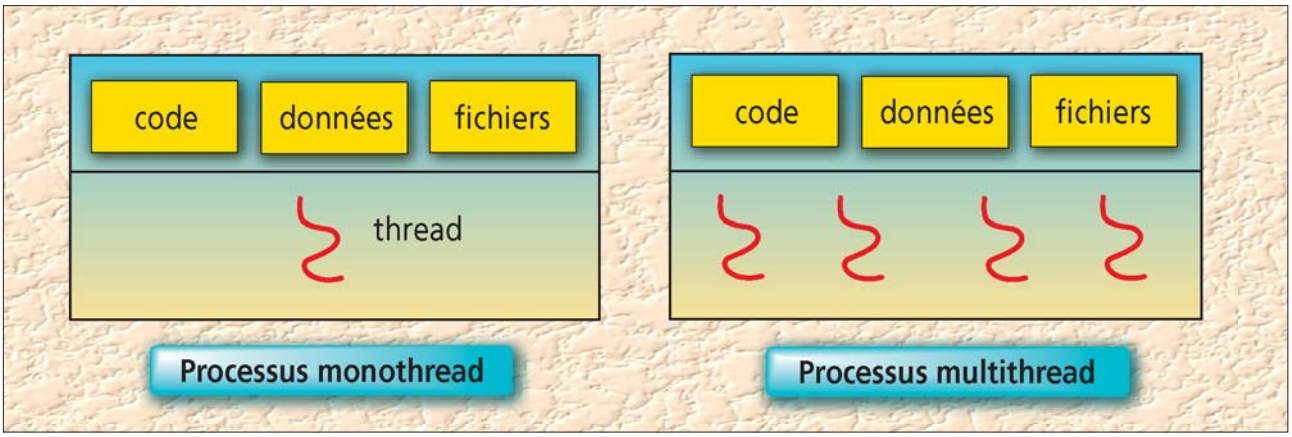


Figure 1. Différence entre processus à flot unique et processus à flots multiples.

processus était terminé, le deuxième processeur étant occupé avec un processus d'attente, le premier processeur se retrouvait dans la configuration du deuxième processeur, et se mettait à exécuter un processus d'attente pour attendre le deuxième…

Cela provoquait donc un « aller et retour » des processus entre les processeurs et affectait donc les performances globales du système.

Le nouvel ordonnanceur corrige ces problèmes en distribuant les tranches de temps par processeur et en éliminant le mécanisme de synchronisation globale et de nouveau calcul des tranches de temps.

Pour ce faire, l'ordonnanceur utilise deux listes de priorité : **la liste active** et la **liste expirée**. La liste active contient toutes les tâches qui sont affines à un processeur et dont la tranche de temps n'est pas encore arrivée à expiration. La liste expirée contient une liste triée de toutes les tâches dont la tranche de temps est arrivée à expiration. Lorsque toutes les tâches de la liste active sont arrivées à expiration de leur tranche de temps, les pointeurs sur ces listes sont échangés et la liste expirée devient la liste active et vice versa.

Les principaux apports du nouvel ordonnanceur sont :

- **efficacité SMP :** s'il y a du travail à réaliser, tous les processeurs vont travailler ;
- **processus en attente :** aucun processus ne restera sans tranche de temps processeur pendant une longue période de temps. Réciproquement, aucun processus n'est censé prendre un temps processeur important ;
- **répartition de charge :** l'ordonnanceur va diminuer dynamiquement la priorité d'un processus qui génère plus de charge que le processeur ne peut en supporter ;
- **amélioration de l'interactivité :** l'utilisateur ne devrait plus voir le système se bloquer ou mettre trop de temps pour répondre à un clic de souris ou à l'utilisation du clavier, même lors d'une utilisation importante du système.

Le noyau préemptif

Cette fonctionnalité a été développée par Robert Love et introduite en tant que patch au noyau 2.5. Elle permet de diminuer de façon significative les temps de latence des applications interactives et offre de vraies caractéristiques temps-réel au noyau 2.6.

Dans les versions précédentes, il n'était pas possible d'interrompre une tâche qui s'exécutait en mode noyau, sauf si cette tâche s'interrompait elle-même ou passait à l'état d'attente. Avec cette nouvelle version, le noyau est capable de s'interrompre lui même et d'occuper le processeur à une autre tâche. Le système gagne ainsi en interactivité et l'utilisateur a l'impression que le système est plus réactif.

Certaines parties critiques du noyau sont cependant restées non-préemptives afin de garantir que les structures et l'état des données d'un processus particulier à un processeur soient protégées. Illustrons ceci par un exemple :

```
int arr[NR_CPUS] ;

arr[smp_processor_id()] = i ;
//préemption du noyau
j = arr[smp_processor_id()]
// i et j ne sont pas égaux donc
smp_processor_id() peut être
// différent
```

Si le noyau avait été préemptif et que la préemption ait eu lieu à l'endroit indiqué par le commentaire « `préemption du noyau` », la tâche aurait été assignée à un autre processeur au moment du réordonnancement et la valeur de `smp_processor_id()` aurait alors été différente. Cette situation peut être évitée grâce au mécanisme de verrous.

Un autre cas où la préemptivité doit être évitée est celui du calcul en mode FPU (*Floating Point Unit*). Dans ce cas, l'état du FPU n'est pas sauvé lorsque le noyau exécute des instructions en virgules flottantes. Si la préemption a lieu, l'état du FPU est complètement différent lors du réordonnancement. Tous les calculs en mode FPU doivent utiliser le mécanisme de verrous.

Pour ce faire, il est possible de désactiver la préemption lorsque cela est nécessaire, puis de la réactiver ensuite. Le noyau 2.6 fournit donc différentes fonctions le permettant :

- `preempt_enable()` : décrémente le compteur de préemption ;
- `preempt_disable()` : incrémente le compteur de préemption ;
- **`get_cpu ()`** : appelle la fonction `preempt_disable()` suivi d'un appel à la fonction `smp_processor_id()` ;
- **`put_cpu()`** : réactive la préemption.

En utilisant ces fonctions, nous pouvons modifier le code précédent afin d'éviter la modification de l'état des données du processus :

```
int arr[NR_CPUS] ;

arr[get_cpu()] = i ; // désactive la
préemption
j = arr[smp_processor_id()]
// réalisation de la tâche critique…
get_cpu() // réactivation de la
préemption
```

L'amélioration de la gestion des *threads* et support de NPTL

Un travail important a été réalisé afin d'améliorer les performances des *threads* dans le noyau 2.5. Ce nouveau modèle de *thread* a également été réalisé par Ingo Molnar. Il est basé sur un modèle de *thread* 1:1 (un *thread* noyau pour un *thread* utilisateur) et supporte, au niveau du noyau, la nouvelle librairie de *thread* Posix (*Native Posix Threading Library, NPTL*) développé par Molnar et Ulrich Drepper. Il est à noter que la distribution grand public RedHat intègre cette librairie depuis sa version 9.0 et la distribution professionnelle Advanced Server l'intègre également depuis sa version 3.0.

Les performances relatives aux *threads* s'en trouvent nettement améliorées car le noyau 2.6 est maintenant capable de gérer un nombre arbitraire de *threads* et jusqu'à deux milliards de PIDs (sur une architecture 32 bits). Ce grand nombre de *threads* simultanés est possible grâce à l'introduction des variables TLS (*Thread Local Storage*) qui permettent d'allouer une ou plusieurs entrées dans la GDT (*Global Descriptor Table*) servant de registre pour les *threads*. Cela permet de créer un modèle de *thread* 1:1 sans limitation du nombre de *threads* créés. Pour information, le noyau 2.4 n'autorisait qu'un nombre maximum de 8192 *threads* par processeurs.

Cependant, cette nouvelle implémentation de gestion de *threads* n'est pas compatible avec les anciennes versions et les anciens programmes se basant dessus doivent être réécrits pour pouvoir bénéficier de ces améliorations.

La gestion de la mémoire virtuelle

Nous avons mentionné que le noyau 2.4 souffrait de problèmes de gestion de la mémoire virtuelle et que cela avait un impact fort sur les performances dans certains cas particuliers. Pour pallier ce problème, les travaux réalisés par Rik Van Riel sur le *reverse mapping* (r–map) ont été incorporés au nouveau noyau.

Le noyau Linux utilise le principe de la mémoire virtuelle. A chaque page mémoire virtuelle correspond une page mémoire physique. La translation d'adresses entre les pages virtuelles et physiques est réalisée par l'HPT (*Hardware Page Table*). Pour une page virtuelle donnée, une entrée dans la table des pages donnera une correspondance avec une page physique, ou, si la page n'est pas présente, une indication d'absence. Cependant, il arrive parfois que la correspondance ne soit pas unique : à une page physique peuvent correspondre plusieurs pages virtuelles. La situation devient problématique lorsque le noyau veut libérer une page physique, car il doit parcourir toute la table des pages afin de retrouver toutes les références à la page physique. La page physique ne peut être libérée que lorsqu'elle n'est plus référencée dans la table des pages… Cela peut entraîner un important ralentissement de la mémoire virtuelle.

Les modifications apportées par le *reverse mapping* corrige ce problème par l'introduction d'une structure de donnée appelée pte_chain dans la structure des pages physique. La `pte_chain` est une simple liste qui pointe vers la PTE (*Page Table Entry*) de la page et qui peut donner une liste des PTE dans lesquelles une page est référencée. La libération de pages devient alors plus simple puisqu'il est facile de connaître rapidement toutes les pages de mémoire virtuelle qui font référence à une page physique.

Le modèle de périphérique unifié (*Unified Device Model*)

Le modèle de périphérique est l'infrastructure avec laquelle le noyau détecte et détermine l'usage des ressources de tous les composants du système.

Les versions de Linux antérieures à la 2.2 avaient seulement une approche simpliste de la gestion des périphériques. Toute la connaissance du matériel était laissée aux modules. Cependant, un système ne doit pas seulement avoir connaissance des ressources nécessaires au fonctionnement du périphérique, il doit aussi savoir à quel bus il est connecté et s'il peut être reconfiguré pour utiliser d'autres ressources. Cette architecture décentralisée disposaient de bus aux API incompatibles.

Le noyau Linux 2.4 a été la première étape d'un modèle de périphérique unifié en joignant les bus PCI, PC Card et ISA Plug-and-Play dans une simple structure de périphérique avec une interface commune.

Linux 2.6 a étendu cette architecture pour produire une vue complètement nouvelle de la façon dont le noyau Linux perçoit le matériel sur lequel il fonctionne, mais également en fournissant une interface centralisée de gestion de périphériques.

Le point central de cette nouvelle architecture est une nouvelle interface orientée objet que tous les dispositifs bas niveau doivent employer. Cette structure est appelée kobject (*kernel object*) et permet l'utilisation d'une API commune à tous les dispositifs bas niveau.

La sécurité

Le support du protocole IPSec (*IP Security*) se fait maintenant au niveau noyau. C'est la première fois que des éléments cryptographiques sont implémentés directement dans l'espace noyau. Cela permet une sécurité au niveau des protocoles réseau. Les algorithmes cryptographiques supportés par le noyau Linux sont actuellement HMAC, MD4, MD5, SHA-1, DES, triple DES et Blowfish.

Le noyau Linux est maintenant capable de supporter les « `capabilities` » (mécanismes de gestion plus fine des droits utilisateurs) sur la base d'un contrôle

utilisateur beaucoup plus fin que l'ancien modèle du « super utilisateur ». Par défaut, l'ancien système où l'utilisateur « *root* » avait tous les pouvoirs est maintenu, mais des exemples de configurations alternatives permettent de définir un contrôle d'accès aux ressources beaucoup plus granulaire que précédemment.

Une autre modification importante est celle qui a été apportée aux modules binaires du noyau (comme, par exemple, ceux qu'une entreprise peut fournir si elle ne souhaite pas divulguer le code source de son logiciel). Ils ne peuvent plus modifier la table des appels système. Cela réduit de manière significative les accès au noyau que peuvent faire des modules non *open-source*.

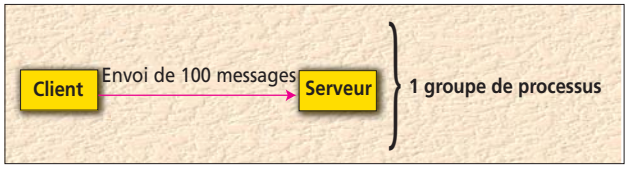
Comparaison entre les noyaux 2.4 et 2.6

L'ensemble des tests suivants permet de mesurer, entre autres, les performances de l'ordonnanceur du noyau 2.6.0 test 9 et de les comparer à celles du noyau 2.4.18. Les tests simulent la connexion à un serveur de « chat » via l'utilisation de processus clients et serveurs dialoguant entre eux. Les processus clients et serveurs sont sur la même machine, ce test n'utilise pas le réseau.

Le test crée un ensemble de processus multiples d'un nombre spécifié par l'utilisateur. Ce nombre est un multiple de 25 dans notre cas.

Chaque couple client/serveur écoute sur une *socket*, l'émetteur (client) écrit 100 messages dans chaque *socket* et le receveur (serveur) écoute sur la *socket*. Dans ce cas, le nombre total de messages envoyés est égal à 100 fois le nombre de processus spécifiés. Le schéma suivant récapitule le principe de fonctionnement du test.

Les résultats et le protocole de ces tests sont basés sur les travaux de Rusty Russel de l'Open Source Developement Labs, Inc. Le code source du test est disponible à l'adresse suivante : `http://developer.osdl.org/craiger/hackbench/src/hackbench.c`.



Ce code est appelé par un wrapper écrit en langage Perl, disponible à l'adresse suivante : `http://developer.osdl.org/craiger/hackbench/src/runit_pl.txt`.

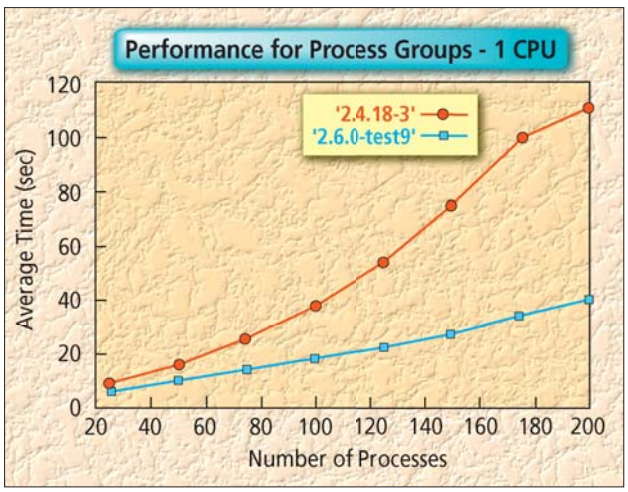
Les tests ont été réalisés entre un noyau 2.4.18 et un 2.6.0 test 9 et sur plusieurs machines de type SMP (***voir annexe***).

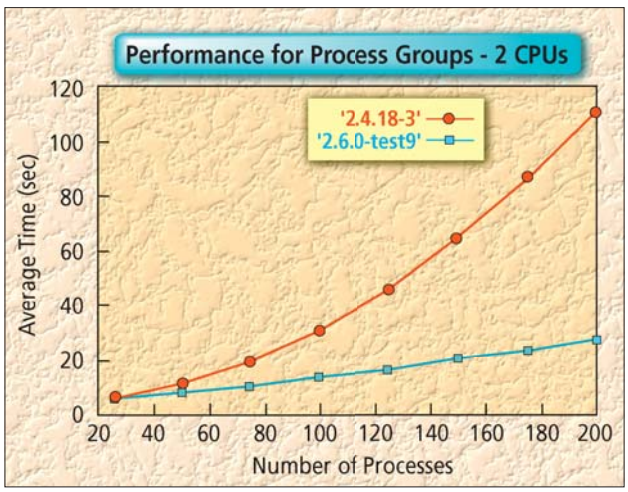
Evaluation de l'efficacité de l'ordonnanceur

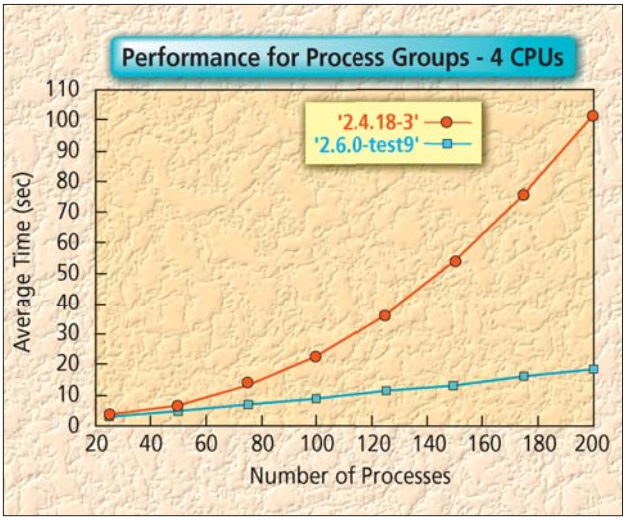
Ce test a été réalisé avec un maximum de 200 groupes d'émetteurs/receveurs (envoi de 20 000 messages). Les graphiques suivants montrent le temps moyen qu'il a fallu pour réaliser les tests en fonction du nombre de groupes de processus.

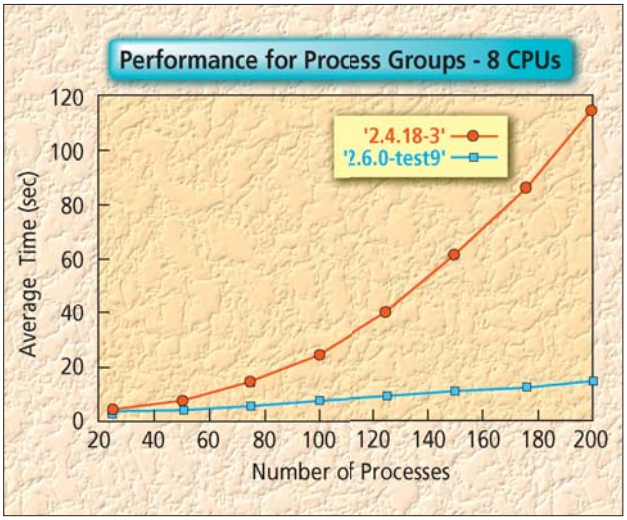
En comparant les résultats, on remarque que ceux du noyau 2.4 forment une courbe parabolique : plus le nombre de processus augmente, plus le noyau met du temps à traiter les requêtes. Au dessus d'un certain nombre de processus, il est probable que le noyau ne sera pas à même de les traiter.

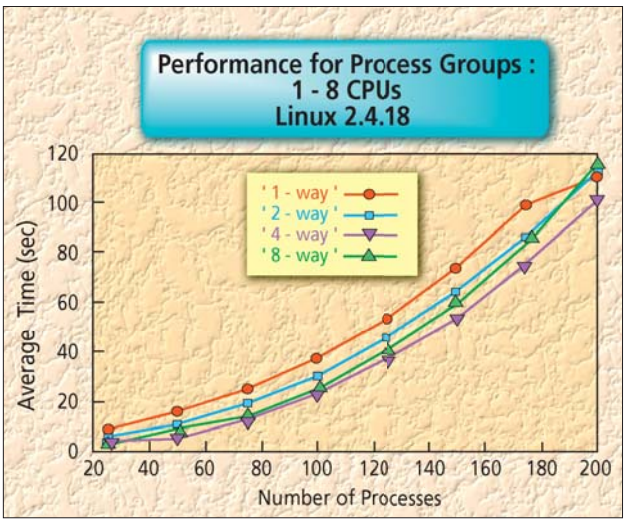
A l'inverse, les résultats du noyau 2.6.0 test 9 forment une courbe beaucoup plus linéaire avec une pente relativement faible qui montre que le nouvel ordonnanceur est capable de gérer beaucoup plus de processus simultanément. La courbe montre également qu'il faut moins de temps au nouveau noyau pour réaliser le test qu'il n'en a fallu au noyau 2.6.

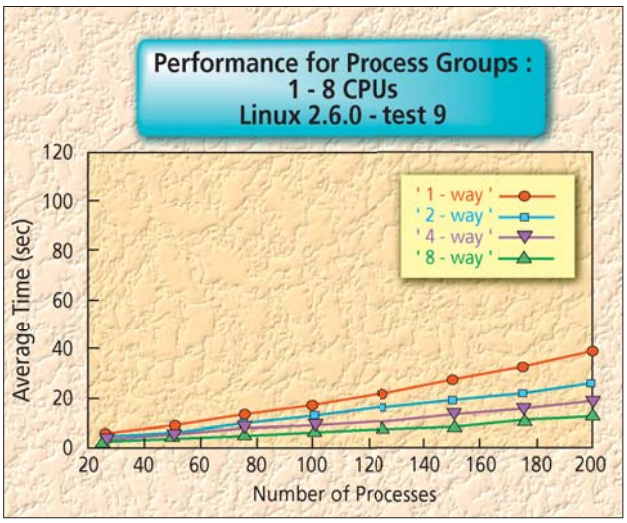












Evolution des performances lors de l'augmentation du nombre de processeurs

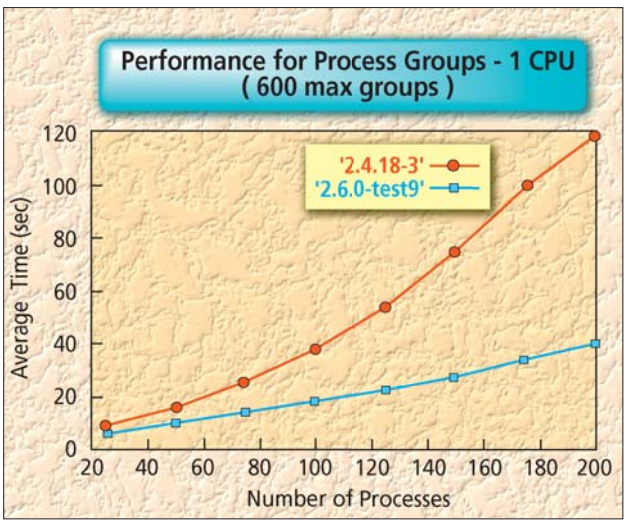
Les graphiques suivants représentent l'ensemble des tests précédents réalisés sur les différentes architectures (SMP et non SMP).

Les résultats du noyau 2.4 montrent qu'à partir d'environ 150 processus, les performances ne changent quasiment pas lorsqu'on augmente le nombre de processeurs de 2 à 8. Ce noyau n'apporte donc pas de performances supplémentaires lors du passage sur des systèmes fortement SMP.

A l'inverse, les résultats pour le noyau 2.6 montrent une nette différence entre les tests réalisés sur les différentes architectures. Les systèmes fortement SMP bénéficient donc grandement des améliorations du nouvel ordonnanceur.

Utilisation des ressources système

Afin de déterminer le nombre maximal de processus que peut supporter le système avant d'être à cours de ressources, le nombre maximum de groupes de processus a progressivement été augmenté jusqu'à l'apparition d'un message d'erreur : `groupfork() (error : Ressource temporarily unavailable)`. Aucune amélioration n'a été apportée dans la configuration du noyau.



Le nombre maximum de groupes de processus a été fixé à 600 pour le système monoprocesseur. Les deux noyaux ont échoué au-delà de 200 processus.

Cette différence de performances entre le système quadri et octo-processeur pour le noyau 2.6.0 est très vraisemblablement dûe au programme de test lui-même : l'erreur provient certainement de l'allocation d'un *buffer* de taille trop petite lors de l'initialisation. Un système octo-processeur alloue plus de mémoire au noyau qu'un système quadri-processeur. Il est donc possible que le test ait moins de mémoire allouée pour son exécution que dans le cas précédent. Pour améliorer ces résultats il faudrait configurer plus finement le noyau utilisé.

Conclusion

Les changements apportés dans le noyau 2.6 sont tellement importants par rapport au noyau 2.4 qu'il a été envisagé de changer de numéro de version majeur pour en faire un Linux 3.0. Finalement, Linus Torvalds a décidé que le changement de numéro n'était pas nécessaire, bien que de nombreuses améliorations fondamentales aient été apportées.

Le noyau Linux est aujourd'hui beaucoup plus rapide dans son fonctionnement, mais également dans le support qu'il apporte au matériel et dans la gestion multiprocesseur.

Ce système d'exploitation est donc suffisamment mûr pour rivaliser pleinement avec les autres systèmes Unix du marché qui incorporaient déjà un certain nombre des nouveaux mécanismes présents dans le noyau Linux. Avant la version 2.6, le principal avantage de Linux était le fait d'être moins onéreux que les Unix propriétaires. Avec la nouvelle version, il est devenu aussi performant. C'est un système de choix pour déployer des serveurs d'applications, mais également pour en faire un nœud central dans des architectures sécurisées : pare-feu, passerelle chiffrante… Cependant, le noyau n'étant sorti qu'en décembre 2003, il reste encore à valider ses performances et sa stabilité en situation réelle. ✪

Description de la configuration matérielle de test

Ordinateur mono-processeur

Carte mère :	SuperMicro
CPU :	1GHz Pentium III w/256k L2 cache
BogoMips :	1982.46
Mémoire :	512MB

Ordinateur bi-processeur

Carte mère :	HP LH3000r
CPU :	850MHz Pentium III w/256k L2 cache x2 (Coppermine)
BogoMips :	1708.03 - 2 processors activated (3407.87 BogoMIPS)
Mémoire :	1GB

Ordinateur quadri-processeur

Carte mère :	HP LT6000r P1757AV
CPU :	700 MHz Pentium III w/1024k L2 cache x4 (Cascades)
BogoMips :	1395.91 - 4 processors activated (5593.49 BogoMIPS)
Mémoire :	4GB

Ordinateur octo-processeur

Carte mère :	HP LXr8500 (Intel SPK8) P1231AV
	Corollary, Inc Intel 8-way XEON Profusion Chipset (Cache Coherency Filter)
CPU:	700 MHz Pentium III w/1024k L2 cache x8 (Cascades)
BogoMips:	1399.19 x8 Total of 8 processors activated (11190.27 BogoMIPS)
Mémoire:	8GB

<Références>

[1] A. SIBERSCHATZ, P. GALVIN, G. GAGNE. « Principes appliqués des systèmes d'exploitation », Vuibert Informatique, Paris 2001.

[2] « Voyage au centre du noyau », Linux Magazine, Episode 1, Hors Série 16, septembre 2003.

[3] « Voyage au centre du noyau », Linux Magazine, Episode 2, Hors Série 17, novembre 2003.

[4] R. RUSSEL. "Linux Process Scheduler Improvement in Version 2.6.0". Open Source Development Labs, 2003.

[5] A. K. SANTHANAM. « Towards Linux 2.6 », IBM developerWorks, Septembre 2003.