Système d'exploitation dédié service

Gilles Carpentier
Institut Supérieur d'Electronique de Paris
21 rue d'Assas 75006 Paris France
+33 1 49545264
+33 1 49545251
gilles.carpentier@isep.fr

Résumé – La virtualisation des serveurs permet de limiter le nombre d'équipements tout en isolant les différents services pour assurer une meilleure fiabilité. Mais, il faut cependant installer un système d'exploitation complet et universel (Windows ou Unix) sur chaque machine virtuelle pour n'offrir qu'un service unique et prédéterminé. Le système que je propose sera configuré statiquement et automatiquement en fonction de l'application ou service que l'on désire installer et des capacités physiques du serveur. Seuls les composants nécessaires à l'application seront installés.
Le code du système et de l'application sera stocké sur une partition en lecture seule tandis que les données non statiques du système et de l'application seront stockées sur une partition en lecture-écriture. Un système de fichiers virtuel masquera la localisation de fichiers d'un même répertoire stockés sur des partitions différentes.
Le système d'exploitation sera transparent pour l'application grâce à une bibliothèque d'appels système compatible dont l'interface sera identique à la celui du système pour lequel cette application avait été conçue.

Mots-clés : système d'exploitation, noyau statique, architecture dédiée serveur

Abstract — In order to limit the number of physical servers, virtualization splits the hardware resources into multiple virtual machines, each dedicated to one service or application.
A complete operating system should be installed on each virtual machine.
Why multi-tasking and multi-purpose operating systems are used for running one single application?
If the operating system crashes, all services will be unavailable or impacted. Virtual machines limit trouble spreading.
The system that I propose will have to be configured statically and automatically according to the capabilities of the physical system and the profile of the application that will have to be deployed. Only the components necessary to operation of the application will be installed.
The code of the system and application will be stored on a logical drive in read-only mode while non-static data will be store on a read-write logical drive. A virtual file system will hide the real location of files.
This operating system will be transparent for application, providing a traditional system call interface in order to avoid porting the applications.

***Keywords** : operating system, static kernel, server dedicated architecture*

REFERENCES

[1] Gary McGraw, Software Security : Building Security In, Addison-Wesley, 2006
[2] Jacques Printz, Puissance et limites des systèmes informatisés, Hermès/Lavoisier, 1999
[3] Robert Sedgewick, Algorithms in C, Parts 1-4: Fundamentals, Data Structures, Sorting, Searching, 3rd Edition, Addison-Wesley, 1997

1 LA PROBLEMATIQUE SERVEUR

1-1 Pourquoi virtualiser les serveurs ?

Virtualiser les serveurs consiste à partitionner un serveur physique en installant plusieurs machines virtuelles dédiées chacune à un service ou une application, dans le but de réduire le nombre de systèmes physiques.
Il est toutefois nécessaire d'installer un système d'exploitation complet (Windows ou Unix) sur chaque machine virtuelle.
Or, ces systèmes d'exploitation sont censés être multi-tâches. Pourquoi utiliser des systèmes universels pour une seule application ? Une des réponses est la fiabilité, la sûreté de fonctionnement. Si le système d'exploitation rencontre un problème, l'ensemble des applications sur ce système est impacté. Les machines virtuelles limitent la propagation des problèmes.
Si l'on admet qu'un système d'exploitation universel ne peut pas être réputé fiable, pourquoi continuer à installer un système complet multi-tâches alors que seul un ensemble réduit de fonctionnalités sont nécessaires pour assurer l'exécution d'une application unique ?

1-2 Performances des serveurs

Les systèmes d'exploitations classiques (Windows, Unix) ont été conçus (surtout Windows) pour répondre aux besoins des postes de travail. La prévision d'activité et donc de charge n'est dans ce cas pas possible, l'allocation des ressources doit donc s'effectuer dynamiquement. Une grande partie du temps est consacrée à l'allocation et à la libération de ses ressources. Notamment, les applications selon le modèle client-serveur, consacrent beaucoup de temps à allouer et libérer des ressources réseaux (sockets).
Dans le cas d'un serveur dédié à une application, cette gestion dynamique des ressources n'est plus nécessaire. Au lancement du système, celui-ci devrait allouer la totalité des ressources disponibles à l'application.

1-3 Complexité et sûreté de fonctionnement

La complexité des systèmes universels comme Unix et Windows non seulement dégrade les performances, mais aussi la stabilité et prévisibilité du système [1]. Et cela ne va pas en s'améliorant. La complexité des systèmes (systèmes d'exploitations mais aussi systèmes d'information) croît plus vite que les performances des équipements [2].
La complexité (et les faibles performances) des systèmes est en partie due à sa modularité (modèle en couches) qui permet d'assurer l'interopérabilité, la portabilité et la facilité d'évolution. Si la nécessité des systèmes ouverts et des réseaux était évidente au début des années 80, qu'en est-il aujourd'hui ?
Côté réseau, il ne reste plus que TCP/IP sur Ethernet ou WiFi. Côté plateforme matérielle, c'est essentiellement du PC Intel ou compatible. La standardisation et la loi du marché simplifient les choix par rapport à la multitude de systèmes et protocoles propriétaires des années 70.

1-4 Historique et marché

Des systèmes d'exploitations dédiés pour les serveurs ont déjà existé par le passé. Novell Netware en est l'exemple certainement le plus connu. Et ses performances et sa stabilité étaient satisfaisantes tant qu'il était utilisé pour partager des fichiers et des imprimantes. Mais ensuite, son champ d'application s'est élargi pour devenir un serveur d'applications. Contrairement à mes objectifs, Netware n'était pas reconstruit pour optimiser l'accès à une application unique.
Plus généralement, est-il possible aujourd'hui d'espérer rencontrer du succès en proposant un nouvel OS étant donné le poids de Windows et Linux ? D'autres (et non des moindres : BeOS, PLAN 9, Inferno) ont essayé sans rencontrer plus qu'un succès d'estime ou de niche.

2 PROPOSITIONS POUR UN NOUVEL OS

2-2 Présentation

Le système devra se configurer statiquement et automatiquement en fonction des capacités du système physique et de l'application qui devra être déployée. Seules les parties nécessaires au fonctionnement de l'application seront installées. Une fois configuré, le système ne devra nécessiter qu'un minimum d'administration. Le code du système et le code de l'application seront stockés sur une partition (ou un support) en lecture seule.
Si possible, ce système d'exploitation devra être vu de l'application comme un système classique (Unix ou Windows) afin d'éviter de porter les applications pour ce nouvel OS.
Cet OS devra dans un second temps être distribué pour assurer la balance de charge et la tolérance aux pannes de l'application.

2-2 Composants de l'architecture

Cette architecture est constituée de 3 composants :

1. Le profileur
2. le configurateur
3. le système proprement dit

Le profileur sera utilisé avant toute installation d'un serveur pour une nouvelle application. Le configurateur sera utilisé un nouveau serveur (ou machine virtuelle) ou une nouvelle version du système d'exploitation mais sur le même serveur.

2-3 Le profileur

Le but de cet outil est d'observer le fonctionnement en charge d'un logiciel orienté serveur (un service web ou mail par exemple).
Il recensera toutes les ressources systèmes qui sont utilisées par l'application :

- appels systèmes
- accès disques
- nombre de blocs mémoire alloués par taille de bloc
- nombre de processus
- nombre de sockets …

Le résultat est un fichier de profil au format XML qui sera utilisé par le configurateur.
Le profileur doit être capable de fonctionner sous Windows et sous Unix. Bien sûr, ce type d'outil existe déjà et il s'agit plus d'un problème d'intégration que de développement. Toutefois, le profileur est la partie la plus critique de cette architecture. La qualité (précision, complétude) du profil, a un impact direct sur la performance du serveur et donc de l'application supportée.
Les appels systèmes sont consignés dans le but de détecter les relations entre l'application et les composants du système. Dans l'étape suivante, le configurateur réutilisera cette information pour octroyer des droits à l'application lorsqu'elle effectue un appel système.

2-4 Le configurateur

La configuration du serveur s'effectue en 2 étapes : construire le système et l'installer. Construire le système étant l'étape la plus complexe, nous allons la simplifier en utilisant une bibliothèque de composants paramétrables et éviter de recompiler tout le noyau à chaque installation.
À l'installation du système, le configurateur recense les ressources matérielles du serveur :

- nombre et type de processeurs,
- taille mémoire
- capacité disque
- interfaces réseaux
- …

Il récupère le profil correspondant à l'application que l'administrateur système souhaite installer sur ce serveur. En fonction du profil de l'application et des ressources matérielles disponibles, le configurateur crée le système et l'installe sur le serveur ainsi que l'application. Une mise à jour du système d'exploitation ne nécessite pas forcément de repasser par cette étape. Par contre, l'installation d'une nouvelle version de l'application supportée est comme installer une autre application. Il faudra donc produire un nouveau profil et reconstruire le système.

2-5 Le système

Une fois le système installé, il démarre en lançant tous les programmes et services nécessaires au fonctionnement du serveur et de l'application. Plus aucun programme ne sera ensuite lancé. Les seuls accès au disque ne concernent que les données de l'application. Ainsi, si un problème doit survenir en raison d'un mauvais dimensionnement du système ou d'une incompatibilité entre composants, il a plus de chance d'être détecté dès le lancement du système et non pas aléatoirement pendant l'utilisation en production du système.
Je ne prévois pas d'inclure une couche d'abstraction matérielle dans le noyau. La première implantation sera pour une plateforme Intel et comme j'espère que le code ne sera pas très volumineux, le portage pour une autre plateforme ne devrait pas être trop fastidieux.

2-5-1 Le système de fichiers

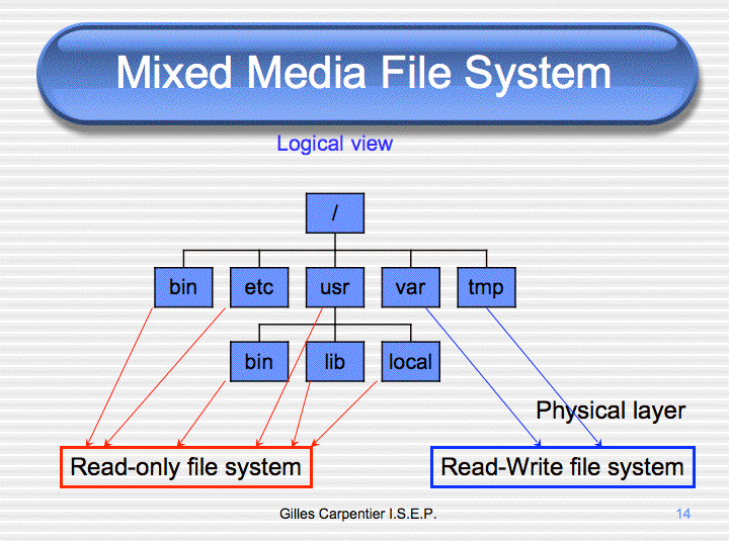
Le système de fichiers se compose de deux couches :

- Une couche logique qui présente à l'application le système de fichier classique du système pour lequel elle a été conçue (NTFS, ou un système de fichier Unix)
- Une couche physique qui mappe chaque fichier du système de fichier logique, soit sur une partition en lecture seule (pour les programmes et fichiers de configuration, soit vers une partition en lecture-écriture pour les données de l'application.

Le but principal d'une telle architecture est la sécurité sans trop de complexité. Il sera facile d'appliquer des règles simples comme :

- Un programme ne peut pas être modifié ou supprimé,
- Un fichier de configuration non plus,
- Seuls les fichiers de log et les fichiers temporaires peuvent être créés ou modifiés mais seulement par le composant identifié dans le profil.

Un autre avantage est qu'il est éventuellement possible de graver une image bootable du système une fois installé. En fait, sur la plateforme de démonstration, nous avons choisi le format ISO CDFS pour implanter la partie en lecture seule de la couche physique du système de fichiers. En cas de panne du disque dur du serveur, il sera assez facile de se servir de cette image pour le nouveau disque, ou alors se passer de disque dur pour la partie du système en lecture seule.



2-5-2 Gestion de la mémoire

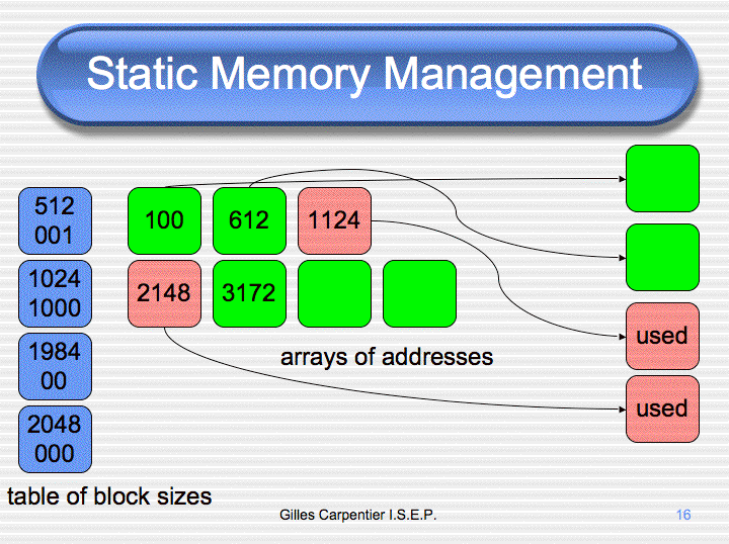
Le profileur a consigné toutes les allocations mémoires effectuées par l'application cible. Comme nous connaissons donc combien de blocks pour chaque taille de block seront nécessaires au fonctionnement de l'application, le gestionnaire de mémoire pourra être conçu aussi simplement que possible pour assurer une efficacité optimale. Typiquement, les structures de données pour stocker la liste des blocks libres, sera statique.
Le choix de l'implantation n'est pas définitif, mais pourrait ressembler à ceci :

- Une table de hachage indexée par la taille de block contiendrait la table des blocks pour cette taille de block et un tableau de bits pour indiquer les blocks libres,
- La taille de chaque tableau est déterminée par le profil de l'application,
- Chaque cellule du tableau contient un pointeur vers le bloc mémoire pré-alloué, ou plus simplement le block mémoire lui-même.

Tous les blocks seront alloués avant le lancement de l'application.
Puisque nous connaissons à l'avance le nombre de tailles de blocks, le nombre de clés et la taille de la table de hachage seront connus, ce qui simplifie et optimise le choix de la fonction de hachage [3].
Pendant le fonctionnement opérationnel de l'application, 2 événements peuvent se produire :

- Une demande d'allocation pour une taille de bloc non prévue,
- Plus de block disponible pour une taille de bloc attendue.

Dans les 2 cas, un bloc libre de la taille supérieure à celle demandée, sera alloué et une notification sera consignée dans le log du gestionnaire mémoire. L'administrateur système pourra créer des statistiques à partir des données du log et évaluer s'il est nécessaire de modifier le profil et de régénérer le système pour diminuer l'occurrence de ces événements.



2-5-3 L 'API

Comme le système de fichiers, l'API sera dépendant de l'application. Si l'application cible était supposée fonctionner sous Linux, notre système présentera des appels systèmes conformes à Linux. Une bibliothèque de fonctions établira la correspondance entre les appels systèmes Linux et les appels systèmes natifs.
Notre système n'appartient donc pas à la famille des exo-kernels. Notre but est de simuler un système classique (Windows, Unix) afin de ne pas avoir à modifier le code de l'application. Mais comme pour une architecture fondée sur un exo-kernel, nous n'aurons qu'à réécrire les fonctions utilisées par la plupart des applications.

2-5-4 L'ordonnanceur

Une fois de plus, nous allons tirer profit de la connaissance apportée par le profileur.
Premièrement, le profil fournit la liste de tous les éléments qui doivent êtres présents dans le noyau ou les services du système (daemons). Ainsi, le configurateur pourra dimensionner statiquement les files d'attente de la gestion des tâches.
Deuxièmement, le profil révèle quel composant du système est appelé par quel autre composant ou par l'application et combien de fois (en fait seule la distinction entre une fois et plusieurs fois nous intéresse). Pour des raisons de sécurité et de sûreté de fonctionnement, le noyau ne permettra pas d'appel système qui n'a pas été prévu par le profil.
Je n'ai pas l'intention de concevoir un système temps-réel, je recherche plus la performance que le respect d'échéances temporelles. Toutefois, certains mécanismes utilisés dans l'implantation de systèmes temps-réels pourront se révéler utiles même dans un système non temps-réel.

2-6 Sécurité et sûreté de fonctionnement

La principale source de sécurité et de fiabilité est fournie conjointement par le système de fichiers et la matrice des appels systèmes autorisés. Deux autres aspects du système vont contribuer à la fiabilité globale.
Premièrement, il n'y a pas de gestion d'utilisateurs. A l'exception des comptes ou groupes requis par certaines applications (par exemple Oracle, Ingres, Globus Toolkit, …), le seul compte local est celui d'administrateur système dont le seul rôle est de pouvoir arrêter le système.
Deuxièmement, le configurateur n'installe que les commandes et fonctions requises par l'application. Par exemple, ni la commande ni l'appel système « chmod » ne seront présent sur le disque en condition normale d'exploitation, elles ne sont disponibles que pendant l'installation du système et de l'application. Le but est de protéger les fichiers en lecture-écriture contre une faille possible de l'application pendant le fonctionnement opérationnel du serveur.

3 CONCLUSION

Les spécifications techniques sont en cours d'écriture. Une plateforme de démonstration pour le système de fichiers a déjà été développée en Java. Ainsi, sous Windows XP, nous pouvons afficher le contenu d'un dossier dont les fichiers sont stockés les uns sur le disque dur, les autres sur un CD-ROM et sur une clé USB.
Le but principal est de réaliser une plateforme afin de mesurer les performances pour une application typique des serveurs (le serveur web Apache ou le serveur de messagerie exim).