Conception et développement de systèmes d'exploitation

Rencontres Mondiales du Logiciel Libre 2005

En 2005, les Rencontres Mondiales du Logiciel Libre auront lieu à Dijon du 5 au 9 juillet. Dijon est situé dans l'Est de la France, à environ 1h30 de train de Paris. Plus d'information sur le lieu des conférences et les possibilités d'hébergement sont disponibles sur le site http://www.rencontresmondiales.org

Ce document contietn le programme du thème *Conception et développement de systèmes d'exploitation* des Rencontres Mondiales du Logiciel Libre 2005. De nombreux autres thèmes techniques ou non-techniques seront présents aux RMLL 2005, leurs programmes sont disponible en ligne sur le site officiel.

Pour plus d'informations concernant le thème *Conception et développement de systèmes d'exploitation*, vous pouvez contacter les responsables de thème Ludovic Courtès <ludovic *dot* courtes *at* laas *dot* fr> et Thomas Petazzoni <thomas *dot* petazzoni *at* enix *dot* org>.

Programme des conférences

Mardi

Début	***Fin***	***Titre***	***Orateur(s)***	***Durée***
16h00	16h10	**Ouverture du thème**	*Ludovic Courtès, Thomas Petazzoni*	10 min
16h10	17h40	**GNU Hurd, présentation générale**	*Gaël Le Mignot*	1h30

Mercredi

Début	***Fin***	***Titre***	***Orateur(s)***	***Durée***
9h30	10h40	**Porter le GNU Hurd sur le micro-noyau L4**	*Marcus Brinkmann*	1h10
10h40	11h00	**Pause**		20 min
11h00	11h50	**Une approche basée sur un marché pour la gestion des ressources dans le système d'exploitation multi-serveurs GNU Hurd**	*Neal Walfield*	50 min
11h50	12h40	**Mesures de l'impact du gestionnaire de mémoire virtuelle dans Linux**	*Mel Gorman*	50 min
12h40	14h00	**Déjeuner**		1h20
14h00	15h10	**Plan 9 de Bell Labs**	*Charles Forsyth*	1h10
15h10	15h40	**Un aperçu du système d'exploitation EROS** ***part 1 of the talk***	*Jonathan S. Shapiro* (Johns Hopkins University)	30 min
15h40	16h00	**Pause**		20 min

Début	*Fin*	*Titre*	*Orateur(s)*	*Durée*
16h00	16h30	**Un aperçu du système d'exploitation EROS** *part 2 of the talk*	*Jonathan S. Shapiro* (Johns Hopkins University)	30 min
16h30	17h40	**D'EROS à Coyotos/BitC : le source ouvert rencontre les preuves ouvertes**	*Jonathan S. Shapiro* (Johns Hopkins University)	1h10

Jeudi

Début	*Fin*	*Titre*	*Orateur(s)*	*Durée*
9h00	9h50	**Les systèmes à image unique pour les grappes : un état de l'art**	*Christine Morin (IRISA)*	50 min
9h50	10h40	**Kerrighed : un système à image unique pour le calcul à haute performance sur grappe Linux**	*Renaud Lottiaux et Pascal Gallard (IRISA)*	50 min
10h40	11h00	**Pause**		20 min
11h00	11h50	**openMosix**	*Moshe Bar*	50 min
11h50	12h40	**Scheduler Activations : principe et implémentation dans le noyau Linux**	*Vincent Danjean*	50 min
12h40	14h00	**Déjeuner**		1h20
14h00	18h40	Sessions plénières		

Vendredi

Début	*Fin*	*Titre*	*Orateur(s)*	*Durée*
9h00	9h50	**JNode.org : pourquoi Java est adapté pour les systèmes d'exploitation modernes**	*Ewout Prangsma*	50 min
9h50	10h40	**Haut-niveau mais neutre ?**	*Frode Vatvedt Fjeld*	50 min
10h40	11h00	**Pause**		20 min
11h00	11h40	**Autour de hOp : des systèmes d'exploitation sans *Novlangue***	*Jérémy Bobbio*	40 min
11h40	12h10	**SOS: la réalisation étape par étape d'un OS de type Unix**	*David Decotigny and Thomas Petazzoni*	30 min
12h10	12h40	**Toy Lovelace: une adaptation de SOS en Ada 95**	*Xavier Grave*	30 min
12h40	14h00	**Déjeuner**		1h20

Début	*Fin*	*Titre*	*Orateur(s)*	*Durée*
14h00	14h50	**Amélioration de la sûreté de fonctionnement par l'utilisation de machines virtuelles**	*Joshua LeVasseur*	50 min
14h50	15h40	**User-Mode-Linux (UML)**	*Jeff Dike*	50 min
15h40	16h00	**Pause**		20 min
16h00	16h50	**THINK, un cadre pour des noyaux de système d'exploitation à base de composants**	*Juraj Polakovic*	50 min
16h50	17h40	**Bossa, un cadre pour le développement d'ordonnanceurs**	*Julia Lawall (DIKU, University of Copenhagen)*	50 min

Samedi

Début	*Fin*	*Titre*	*Orateur(s)*	*Durée*
9h00	10h40	**Merveilleux voyage à l'intérieur d'un OS, pour les débutants** *partie 1, en salle de TD*	*Thomas Petazzoni*	1h40
10h40	11h00	**Pause**		20 min
11h00	12h40	**Merveilleux voyage à l'intérieur d'un OS, pour les débutants** *partie 2, en salle de TD*	*Thomas Petazzoni*	1h40

Résumé des conférences

GNU Hurd, présentation générale, par Gaël Le Mignot

Le GNU Hurd, ensemble de serveurs gravitant autour d'un micro-noyau, est le coeur du projet GNU. Cette conférence vise à présenter son historique, les raisons de son existence, son rattachement au projet GNU et à la philosophie GNU, et surtout les principes techniques de son fonctionnement et son architecture. En particulier, les méchanismes de communication inter-processus, la notion de translators, la gestion de la mémoire sous Mach ainsi que le modèle de sécurité par jetons du Hurd seront présentés. En conclusion, une présentation rapide du futur du Hurd, L4, pourra servir d'introduction à la conférence de Marcus.

Présentation en anglais.

Porter le GNU Hurd sur le micro-noyau L4, par Marcus Brinkmann

Une sympathique révolution dans le monde des OS

GNU Hurd est un système d'exploitation multi-serveurs qui fonctionnent au dessus d'un micro-noyau. Bien que l'implémentation actuelle basée sur Mach ai permis l'implémentation d'un prototype fonctionnel, permettant de lancer des centaines d'applications différentes écrites pour les systèmes POSIX, on peut identifier plusieurs problèmes importants qui limitent l'efficacité et la robustesse du système. L'expérience

du micro-noyau serait-elle morte ?

Alors que le Monde Réel (TM) a continué à construire son succès sur des concepts éprouvés mais anciens de conception des système d'exploitation, les poussant jusqu'aux limites les plus extrêmes, la recherche sur les systèmes d'exploitation, bien que marginalisée, s'est poursuivie. De nouvelles générations de micro-noyaux, comme le micro-noyau L4 développé par Jochen Liedtke à Karlsruhe, ont été conçus pour résoudre les problèmes de conception fondamentaux des micro-noyaux de première génération.

Le port port de GNU Hurd au dessus du micro-noyau L4 est une initiative visant à mettre ces nouveaux principes de conception en pratique, tout en préservant, voire en étendant la liberté de l'utilisateur, qui est à la base de la conception du Hurd.

Dans ma présentation, j'expliquerai les deux principaux problèmes que nous avons identifiés dans le système de transmission de messages (RPC) du micro-noyau Mach et dans sa gestion de la mémoire, et décrirait comment nous envisageons de les résoudre dans le contexte du micro-noyau L4 Pistachio. J'illustrerai comment ces solutions permettent d'améliorer le GNU Hurd, tout en respectant encore plus sa philosophie de conception. J'expliquerai pourquoi nous avons bon espoir de résoudre les principaux problèmes rencontrés avec Mach, mais j'exposerai également de nouveaux problèmes.

Présentation en anglais.

Scheduler Activations : principe et implémentation dans le noyau Linux, par Vincent Danjean

Le multithreading est de plus en plus utilisé dans le calcul hautes performances pour exploiter les machines multiprocesseurs. Cependant, les bibliothèques de threads POSIX sont rarement adaptées : fonctionnalités inutiles (gestion des signaux, ...) ou manquantes (gestion applicative de l'ordonnancement, ...), threads trop coûteux en ressources, ... Des bibliothèques ad-hoc de niveau utilisateur sont alors développées, mais elles rendent problématique l'utilisation d'appels systèmes bloquants : l'application entière est suspendue. Le mécanisme des *Scheduler Activations* permet de contourner cette limitation. Le mécanisme original proposé par Anderson sera exposé ainsi que les modifications qui lui ont été apportées pour cibler le système Linux et plus particulièrement le domaine du calcul hautes performances. Quelques résultats obtenues avec ce mécanisme et la bibliothèque de thread Marcel seront présentés.

Présentation en anglais.

Plan 9 de Bell Labs, par Charles Forsyth

Plan 9 est un système d'exploitation réparti nouveau écrit par le centre de recherche de Bell Labs (celui qui a écrit Unix), et qui est sorti il y a quelques années en temps que Logiciel Libre. Le système présente une implémentation propre de quelques mécanismes simples et puissants permettant de construire des systèmes répartis.

Cette présentation (ou ces présentations) donnera un aperçu des composants de Plan 9, de sa conception et de sa mise en oeuvre. Le système représente toutes les ressources (y compris l'affichage, les interfaces réseau, les piles de protocoles et les services) sous la forme de fichiers dans un espace de nom hiérarchisé, propre à un processus et assemblé dynamiquement. Cette représentation universelle sous la forme de fichiers permet à un simple protocole de service de fichiers (9P) d'implémenter des services et de lier les composants ensemble. Il fournit un support structurel fort au niveau système pour construire des applications réparties, remplaçant ainsi une pléthore de protocoles spécifiques. Beaucoup de services sont donc mis en oeuvre sous la forme de serveurs de fichiers : le système de fenêtrage (rio), une "interface utilisateur pour les programmeurs" (Acme), la communication inter-processus (la "tuyauterie" ou

plumbing), la découverte de services, le serveur de noms de domaine (dns), et l'authentification sécurisée (factotum). Le serveur de fichiers sur disque (fossil) a également des caractéristiques inhabituelles : il fournit une structure de désignation par dessus un service d'archivage (venti) et permet un accès direct par leur nom à des fichiers du passé. Des applications clef, incluant la plupart de celles mentionnées précédemment, utilisent le support de la programmation répartie offert par Plan 9. Des principes similaires sont appliqués à l'intérieur même du noyau. Par exemple, un pilote de périphérique fournit un ensemble de noms représentant les données et les chemins de contrôles du périphérique (ou réseau) sous-jacent. Le système est portable : il tourne sur x86, PowerPC, ARM, SPARC et d'autres architectures. Il est aussi indépendant : il fournit ses propres suites de compilateurs et de commandes.

Présentation en anglais.

Amélioration de la sûreté de fonctionnement par l'utilisation de machines virtuelles, par Joshua LeVasseur

Je décris notre technique de réutilisation de pilotes de périphériques non modifiés pour améliorer la sûreté de fonctionnement en utilisant des machines virtuelles. Nous faisons tourner un pilote non modifié, avec son système d'exploitation original, indépendamment du fournisseur du SE ou du pilote, et nous réduisons significativement la barrière pour les efforts de construction de nouveaux SE. En confinant des pilotes distincts dans des machines virtuelles séparées, cette technique isole les fautes causées par un pilote défectueux ou malicieux et améliore ainsi la sûreté de fonctionnement du système.

Nous démontrons la réutilisation de pilotes avec un environnement de "para-virtualisation" sur Linux 2.6 et Linux 2.4. Nous montrons que notre technique requiert une infrastructure minimale, fournit de bonnes performances, et fournit une forte isolation des fautes. La performance réseau de notre prototype se situe dans les 3% à 8% du système Linux natif. Chaque nouvelle machine virtuelle augmente l'utilisation du processeur de 0,12%. Nous avons réussi à réutiliser une grande variété de pilotes réseau, disque, et PCI de Linux.

Présentation en anglais.

Les systèmes à image unique pour les grappes : un état de l'art, par Christine Morin (IRISA)

Depuis quelques années, nous observons un regain d'intérêt pour les systèmes à image unique (SSI) pour grappes, en particulier dans le domaine du calcul à haute performance. Un système à image unique offre l'illusion qu'une grappe est une machine unique. Un tel système simplifie l'utilisation et la programmation des grappes pour le calcul parallèle. Un SSI gère globalement les ressources d'une grappe pour masquer leur distribution dans les différents noeuds de la grappe. Il est constitué d'un ensemble de services distribués pour gérer les processus, la mémoire, les flux de données et les fichiers à l'échelle d'une grappe.

Dans cet exposé, nous nous intéresserons plus particulièrement aux systèmes à image unique construits par extension au système d'exploitation Linux, tels que OpenSSI, openMosix et Kerrighed. Ces systèmes offrent l'interface Linux étendue avec des appels système spécifiques aux grappes tels que la migration de processus. L'intérêt de ces systèmes réside qu'ils permettent l'exécution d'une gamme d'applications existantes conçues pour des machines SMP sous système Linux sans modification.

Après une présentation de la problématique de conception et mise en oeuvre des systèmes à image unique, nous présenterons une étude comparative des systèmes OpenSSI, openMosix et Kerrighed fondés sur Linux et disponibles sous forme de logiciel libre. Cette étude porte d'une part sur la couverture des propriétés SSI et d'autre part sur une évaluation des performances des mécanismes clés d'un système SSI.

Présentation en anglais.

Kerrighed : un système à image unique pour le calcul à haute performance sur grappe Linux, par Renaud Lottiaux et Pascal Gallard (IRISA)

Kerrighed est un système à image unique pour le calcul à haute performance sur grappe. Il est conçu et mis en oeuvre au sein du projet INRIA PARIS de l'IRISA dans le cadre d'un partenariat avec EDF R&D. Fondé sur le système Linux qu'il étend, Kerrighed est un système d'exploitation qui virtualise l'ensemble des ressources d'une grappe et gère leur partage entre plusieurs applications. Kerrighed offre l'illusion qu'une grappe est une machine multiprocesseur à mémoire partagée (SMP) sous système Linux.

L'architecture logicielle du système Kerrighed comporte trois niveaux. Au niveau le plus bas, Kerrighed met en oeuvre un service de communication à haute performance et très haute réactivité offrant une interface de niveau noyau adaptée à la construction de services système de plus haut niveau et portable par rapport aux technologies de réseau d'interconnexion (Gigabit Ethernet, Myrinet, Infiniband, ...). Le niveau intermédiaire met en oeuvre les concepts clés de Kerrighed : la notion de conteneur pour la gestion globale et le partage de mémoire, la notion de processus fantôme pour la gestion globale des processus et la gestion de points de reprise, la notion de flux de données dynamiques pour la gestion globale des flux de données en présence de migration de processus. Au niveau supérieur, Kerrighed offre différents services tels qu'un ordonnanceur global de processus, un système de gestion de fichiers distribué, un service de gestion de mémoire et les interfaces de communication inter-processus traditionnelles (tube, sockets, ...).

Le système Kerrighed se distingue des autres système à image unique fondés sur Linux par son haut degré de spécialisation. Kerrighed permet en particulier de choisir la poltique d'ordonnancement global de processus qui peut être changée à chaud. En outre, plusieurs fonctionnalités SSI peuvent être activées ou désactivées, processus par processus, à la demande par programmation ou par l'intermédiaire d'une interface ligne de commande.

Kerrighed offre la compatibilité binaire pour les applications MPI (validé sur MPICH), multithreadées à la norme Posix, OpenMP (validé avec deux compilateurs OpenMP ciblant les threads Posix). Des expérimentations avec des applications industrielles de grande envergure ont été menées dans le cadre d'un contrat avec la DGA.

L'exposé porte sur une présentation des principes de conception de Kerrighed, sa mise en oeuvre et sur les résultats d'expérimentation.

Présentation en anglais.

JNode.org : pourquoi Java est adapté pour les systèmes d'exploitation modernes, par Ewout Prangsma

JNode.org, un nouveau système d'exploitation pour utilisation personnelle sur matériel récent, a progressé de manière significative ces deux dernières années et est maintenant devenu le seul système d'exploitation en Java activement développé dans la communauté *open source*.

Le système d'exploitation *JNode.org* est implémenté en totalité avec le language de programmation *Java,* et contient donc une machine virtuelle Java (elle même implémentée en Java) au coeur de l'OS. Ceci rend la conception traditionnelle des systèmes d'exploitation basée sur des espaces noyau et des espaces utilisateur obsolète. Un *framework* flexible de plugins a été implémenté pour rendre l'OS très modulaire. Tous les pilotes de périphériques, la pile réseau, les systèmes de fichiers sont des plugins séparés qui

peuvent être chargés, déchargés et rechargés à la demande. Un *framework* moderne pour les pilotes de périphériques qui utilise les possibilités du langage *Java* permet de concevoir et d'implémenter aisément des pilotes de périphériques qui prennent en compte un environnement en évolution permanente.

Durant cette conférence, l'architecture de base et la conception de cet OS seront exposées, suivies d'un aperçu plus détaillé du *framework* de plugins et du *framework* de pilotes de périphériques. Il mettra en lumière ce nouveau et excitant système d'exploitation et répondra à la question « pourquoi Java est adapté pour les systèmes d'exploitation modernes ».

Présentation en anglais.

Haut-niveau mais neutre ? Par Frode Vatvedt Fjeld

Movitz est un environnement d'exécution et un compilateur Common Lisp pour les processeurs x86 et est conçu pour servir de plateforme pour des noyaux de systèmes d'exploitation et des systèmes à application unique. Un aspect important de Movitz est sa neutralité («*policy free*») : le concepteur de l'application ou de l'OS reste libre de mettre en place sa propre architecture et ses propres implémentations pour des mécanismes aussi basiques que le ramasse-miettes (*garbage collector*), les fils d'exécution (*threads*), la protection, et la liaison dynamique (*dynamic binding*). Cette conférence propose un aperçu de la plateforme Movitz, et met l'accent sur certains des défis que nous avons rencontré durant la conception et l'implémentation d'un environnement de programmation qui est à la fois neutre et de haut-niveau.

Présentation en anglais.

THINK, un cadre pour des noyaux de système d'exploitation à base de composants, par Juraj Polakovic

À strictement parler, THINK n'est pas un système d'exploitation, mais une plateforme logicielle pour les systèmes d'exploitation à base de composants. En d'autres termes, THINK permet la construction de systèmes d'exploitation spécifiques par assemblage et spécialisation de composants, tout en encourageant la réutilisation de code. De nombreux choix de conception du système sont laissés au concepteur et ne sont pas prédéterminés dans la plateforme THINK. Par exemple, THINK permet aussi bien de construire un OS à espace d'adressage unique sans protection de la mémoire assistée par le matériel qu'un OS avec des espaces d'adressage virtuels. THINK se concentre sur les systèmes d'exploitation embarqués temps réel, c'est-à-dire les systèmes avec des contraintes fortes en termes de ressources, et des performances déterministes. L'architecture à composants de THINK permet l'obtention de noyau dont l'empreinte mémoire est faible, et dont toutes les fonctionnalités inutiles ont été supprimées. La plateforme inclut une bibliothèque de composants qui propose divers services d'un OS comme l'ordonnancement, la gestion des interruptions, le réseau ou les systèmes de fichiers. La plupart de ces composants sont indépendants du matériels, et lorsque que des dépendances sur le matériel existent, elles sont isolées dans des sous-composants bien identifiés. THINK permet également le développement d'extensions pour implémenter des propriétés non fonctionnelles comme la QoS, la sécurité ou la reconfiguration dynamique. Après une brève description de THINK, la présentation se concentrera sur les mécanismes permettant l'implémentation de ces extensions non fonctionnelles.

Présentation en anglais.

Un aperçu du système d'exploitation EROS, par Jonathan S. Shapiro (Johns Hopkins University)

Que faudrait-il pour construire un système d'exploitation robuste et sécurisé ? Plus important encore, un système d'exploitation qui permettrait le développement d'applications robustes et sécurisées ? Pas uniquement des services sécurisés, mais sécurisé dans le sens ou tous les utilisateurs sont sécurisés ? En bref, le système d'exploitation dont nous avons besoin au 21ème siècle (et peut-être, rétrospectivement, au 20ème). Cette liste de souhaits est particulièrement courte, and pourtant il n'y a aucun système d'exploitation aujourd'hui qui ne commence même à la remplir. Le système EROS essaie de répondre à ce besoin. La présentation à ce sujet sera découpée en trois phases.

Tout d'abord, je décrirai brièvement une application de tous les jours, un navigateur, et j'expliquerai pourquoi sa vulnérabilité est structurelle, et (au moyen d'illustrations) montrerai comment le rendre sécurisé par l'utilisation d'une structuration basée sur des capacités (*capabilities*). Au cours de cette description, je mettrai en lumière divers aspects de l'interface du système d'exploitation qu'il est nécessaire de désactiver pour permettre cette structuration de fonctionner. A la fin de cette partie, les raisons pour lesquelles UNIX, Windows, et les autres architectures de noyau similaires ne sont pas le bon point de départ pour architecturer les applications seront claires. Je terminerai cette partie en énumérant les principes généraux qu'il illustre.

Deuxièment, je décrirai brièvement les services proposés par le noyau EROS, et identifierai quelques points sur lesquels la conception d'EROS est radicalement différente de celle des autres systèmes d'exploitation -- en particulier sa confiance en un mécanisme d'allocation de ressources entièrement responsable et la présence d'un sous-système de points de reprise (*checkpointing*). Le temps ne permettra pas un aperçu en profondeur ; l'objet de cette partie est de montrer que le noyau et les sous-systèmes critiques d'EROS proposent un substrat quasi-idéal pour développer des applications efficaces et sécurisées.

Troisièmement, je parlerai des défauts techniques du système EROS, et des raisons pour lesquelles (au moins à mon avis), le système EROS n'a pas pu aboutir à la création d'un système complet et utilisable. La plupart des problèmes techniques sont mineurs d'un point de vue de l'architecture et ont des solutions relativement simpels. L'un d'eux a des implications bien plus larges. Ces causes sont plus larges et constituent une occasion pour la communauté -- de manière surprenante, des changements mineurs dans notre API nous permettraient d'amener nos applications vers une plateforme plus sécurisée avec peu de modifications, et il serait intéressant de démarrer la transformation du code source maintenant.

Le projet EROS va continuer indépendamment sous le nom CapROS. Si il y a un message à retenir du travail d'EROS, c'est qu'il est réellement possible de construire des systèmes sécurisés, et que ce n'est pas réellement possible en étendant des plates-formes existantes. La conférence suivante sur le système Coyotos permettra d'exposer où nous allons avec le successeur des travaux sur EROS.

Présentation en anglais.

D'EROS à Coyotos/BitC : le source ouvert rencontre les preuves ouvertes, parJonathan S. Shapiro (Johns Hopkins University)

Trois systèmes ont émergé suite aux travaux effectués sur EROS : le projet CapROS essaie d'avancer avec l'architecture d'EROS telle qu'elle existe, le projet L4++ (L5?) tente de créer un successeur à L4 entièrement basé sur les capacités (*capabilities*), et le système Coyotos, qui est notre propre successeur. Le projet Coyotos a deux objectifs. Le premier est de régler les imperfections techniques d'EROS. Le

second est de repenser la façon dont les clients logiciels pensent la sécurité et la fiabilité. Nous espérons lancer un mouvement vers des systèmes dont le code est «sûr» et des systèmes critiques dont le code est vérifié de manière *ouverte*. Coyotos sera utilisé à la fois en tant que projet initial pour tester notre infrastructure de vérification, et d'exemple montrant comment de telles choses peuvent être construites. Le système Coyotos sera construit en deux éditions : une première version en C pour valider la conception, suivie d'une seconde version en BitC, notre language de programmation pour systèmes vérifiables.

Un concept clé des travaux portant sur Coyotos/BitC est la notion de « Preuves ouvertes ». Notre point de vue est que le processus de Critère Commun a entièrement échoué pour deux raisons. D'abord, l'inspection et le test ne sont pas assez rigoureux pour permettre la création de systèmes sécurisés et robustes. Deuxièment, le client ne paie pas pour la validation. Le vendeur peut alors exploiter la compétition sur les prix pour tirer la qualité de la validation vers le bas (et il le fait). Nous proposons qu'une meilleure fondation serait des implémentations à sources ouvertes dans lesquels les objectifs de vérification sont formellement exprimés et rigoureusement testés d'une manière permettant la création une chaîne de preuves librement disponibles et reproductibles. Le client (ou un contracteur travaillant pour lui) peut ré-exécuter ces preuves par eux-mêmes comme une vérification du vendeur. Nous nommons ce processus et cette méthodologie « Preuves ouvertes ». En plus de proposer une base viable pour tester la sécurité, les preuves ouvertes, couplées avec le sources ouvertes, permettent aux clients de réaliser des modifications et des adaptations personnalisées, desquelles ils peuvent déterminer si elles ont par inadvertance violé des propriétés de sécurité ou de fiabilité.

Durant les 25 dernières années, des progrès étonnants ont été réalisés dans la communauté de la recherche sur la vérification des programmes. La principale difficulté à une adoption plus large est que peu de «créateurs» ont réellement essayé de transformer ces technologies en des outils réellement utilisables. Les outils de vérification de programmes actuels sont plus ou moins « tout ou rien », et beaucoup de choses que nous avons besoin d'exprimer en tant que programmeurs système ne peuvent être exprimées ou vérifiées dans de telles plateformes limitées. Il y a un fossé à la fois dans les outils et dans l'orientation des créateurs des outils. En tant que hard-core créateurs de système, la vision de mon groupe est « Exprimons ce que nous avons besoin d'exprimer, et voyons sur quel sous-ensemble de cela nous pouvons utiliser des techniques de vérification. Pendant que nous aprenons, nous adapterons le code source, les outils de vérification et les objectifs de vérification pour faire de mieux en mieux. Nous avons besoin de faire une vérification complète sur très peu de programmes. Pour le reste, peut-être nous voulons simplement une meilleure technique pour vérifier notre travail » En utilisant le language BitC et une plateforme logique, l'implémentation du coeur de Coyotos fournira des exemples d'application des techniques de vérification avec différents compromis coût/bénéfice.

Cette présentation se déroulera en deux phases. Tout d'abord, je décrirai les service du coeur du noyau Coyotos en pointant les différences avec les travaux précédents sur EROS. Au fil de la description, j'indiquerai quelles sont les fonctionnalités qui ont des similitudes avec celles du sucesseur sécurisé de L4. En deuxième phase, je donnerai quelques exemples de propriétés que nous aimerions vérifier dans l'implémentation de Coyotos, des simples tests de cohérence relativement simples à exprimer et à penser aux problèmes de correction globale qui représenteront une grosse partie du travail. J'expliquerai pourquoi nous croyons que des propriétés plus complexes peuvent fonctionner dans Coyotos alors qu'elles ont échoué dans de précédents systèmes.

Présentation en anglais.

User-Mode-Linux (UML), par Jeff Dike

La présentation parlera des travaux en cours sur UML et de leur rapport à la virtualisation en général. Mon point de vue sur la virtualisation est que ça va devenir une partie intégrante des systèmes d'exploitation et de leurs applications, plus qu'un composant séparé, isolé. Je vois UML comme étant un élément essentiel pour apporter la virtualisation aussi bien au niveau noyau qu'au niveau utilisateur étant donné qu'il s'agit à la fois d'un noyau virtualisé, le rendant utile dans le noyau hôte, et d'un processus, le rendant utilisable par les processus utilisateur. Je parlerai des avantages que je vois à rendre la virtualisation disponible dans ces deux domaines, comment UML y contribue exactemement, et des travaux en cours.

Présentation en anglais.

Autour de hOp : des systèmes d'exploitation sans *Novlangue*, par Jérémy Bobbio

Quasiment tous les systèmes d'exploitation depuis 35 ans ont été écrits en C. Même si cela avait du sens à l'époque, ne serait-ce pas devenu une limite en termes de design, de taille de code ou de sécurité ? Pendant tout ce temps, la recherche sur les langages de programmation ne s'est pas arrêté. Pourquoi les découvertes qui ont été faites ne pourraient-elle pas profiter au code de nos noyaux ?

À partir des expériences sur hOp/House et du langage purement fonctionnel Haskell, cette présentation essaiera de faire le tour des avantages et des inconvénients des langages de programmation modernes pour l'écriture de systèmes d'exploitation.

Présentation en anglais.

Bossa, un cadre pour le développement d'ordonnanceurs, par Julia Lawall (DIKU, Université de Copenhague)

Écrire un nouvel ordonnanceur et l'intégrer dans un système d'exploitation existant est une tâche longue et difficile, nécessitant la compréhension de plusieurs mécanismes internes et bas niveau du noyau. Ainsi, l'écriture d'un nouvel ordonnanceur est hors de portée pour un développeur d'application, bien que ceux-ci soient les mieux placés pour connaître les besoins d'ordonnancement de leur application.

Nous proposons une plateforme, Bossa, qui permet aux développeurs d'applications d'implémenter leurs propres ordonnanceurs noyau de manière simple et sûre. Cette plateforme définit une interface d'ordonnancement qui est implémentée dans une version du noyau Linux par un expert du domaine en utilisant une approche dérivée de la Programmation Orientée Aspect. Les ordonnanceurs eux-mêmes sont écrits en utilisant un langage spécifique au domaine (*domain specific language,* ou *DSL*) qui offre des abstractions de haut niveau spécifiques à l'ordonnancement pour simplifier le développement de politiques d'ordonnancement. L'utilisation d'un DSL facilite le développement de l'ordonnanceur et permet de vérifier que la politique d'ordonnancement est compatible avec les possibilités de l'OS. Nous avons trouvé que Bossa donne de bonnes performances en pratique. La présentation couvrira le DSL Bossa, son implémentation dans le noyau Linux 2.4 et son utilisation dans le contexte d'applications multimédias.

Présentation en anglais.

openMosix, présentation générale, par Moshe Bar

openMosix est une extension du noyau pour rassembler en un cluster à image unique plusieurs ordinateurs (*single-system image clustering*). Cette extension transforme un réseau d'ordinateurs classiques en un super-ordinateur pour des applications Linux. Une fois qu'openMosix est installé, les noeuds du cluster commencent à discuter entre eux et le cluster s'adapte de lui-même à la charge de travail. Les processus

originaires d'un noeud peuvent migrer vers un autre noeud si le premier est trop occupé par rapport aux autres. openMosix essaie d'optimiser l'allocation des ressources de manière permanente.

Nous réalisons ceci grâce à un patch pour le noyau Linux, créant une plateforme rapide, peu coûteuse et sûre pour des clusters à image unique qui s'adaptent et passent à l'échelle de manière linéaire. Avec la découverte automatique d'openMosix, un nouveau noeud peut être ajouté au cluster pendant son fonctionnement et le cluster pourra utiliser automatiquement les nouvelles ressources.

Il n'est pas nécessaire d'utiliser des applications spécifiques à openMosix. Puisque toutes les extensions openMosix se trouvent dans le noyau, n'importe quelle application Linux bénéficiera automatiquement et de manière transparente du concept de calcul distribué d'openMosix. Le cluster se comporte de manière similaire à un système multi-processeurs symétrique, mais cette solution passe bien à l'échelle pour un milliers de noeuds, qui peuvent eux-mêmes être SMP.

La communauté openMosix est très active, contributant des applications additionnelles et partageant des informations avec tous les utilisations. openMosix et les applications l'entourant sont distribuées sous licence GPL.

openMosix 2.6 a été reécrit en totalité et réalisé de manière indépendante de la plateforme. Nous avons réalisé le port pour PowerPC, AMD Opteron et EM64T. Dans cette présentation, nous étudierons les différents composants tels qu'ils existent aujourd'hui : migration des processus, déplacement des régions virtuelles, répartition de la charge, partage de mémoire, interface /proc.

Nous verrons également quelques scénarios d'utilisation qui mettront à la fois en avant les meilleures utilisations et les pires utilisations des systèmes à image unique.

Présentation en anglais.

Mesures de l'impact du gestionnaire de mémoire virtuelle dans Linux, par Mel Gorman

Un préoccupation importante de la plupart des systèmes d'exploitation et l'ensemble des méthodes et mécanismes qu'il met en oeuvre pour gérer la mémoire physique et, dans beaucoup de cas, la façon dont cela s'articule avec la mémoire virtuelle. Par conséquent, le gestionnaire de mémoire virtuelle (VMM) joue un rôle critique par rapport à la performance du système étant donné que la plupart des sous-systèmes de l'OS doivent interagir avec lui et, dans certains cas, interagissent beaucoup.

De par son rôle critique, il est important de comprendre tous les aspects de l'interaction du VMM avec le système d'exploitation et avec les différents types de processus qui tournent sur la machine. Notre recherche se donne pour but de définir des métriques pour mesurer la performance d'un VMM et d'implémenter des outils pertinents spécifiquement pour Linux.

Cette présentation va commencer par introduire quelques unes des préoccupations de performance du VMM et de comment elles pourraient être mesurées. Nous présenterons ensuite certains des outils et des métriques que nous avons développés.

Présentation en anglais.

SOS: la réalisation étape par étape d'un OS de type Unix, par David Decotigny et Thomas Petazzoni

Nous discuterons de Sos, *encore un autre* système d'exploitation simple, un système d'exploitation dont

l'objectif est d'apprendre de manière didactique et étape par étape comment les OS sont implémentés. Chaque mois, un article paraît dans *GNU/Linux Magazine France*, présente un nouveau concept, le décrit, et présente rapidement son implémentation dans des systèmes d'exploitation répandus puis propose une approche simple mais fonctionnelle pour l'implémenter. Chaque article est fourni avec le code source associé, résultant en l'implémentation progressive de l'ensemble de l'OS. À la fin, Sos devrait avoir la majorité des fonctionnalités de base des systèmes de type Unix, telles que les threads utilisateur et noyau, les processus, la gestion de la mémoire virtuelle, le support pour des systèmes de fichiers, et un support réseau minimal. Techniquement parlant, Sos est écrit en C, et est prévu pour fonctionner sur des ordinateurs IA32 monoprocesseur. Il dérive de Kos (The Kid Operating System) en plusieurs aspects, en particulier par l'expérience acquise par les deux auteurs durant l'existence de Kos. Comme pour Kos, l'ambition principale de Sos n'est pas de devenir un nouveau "killer-OS", mais simplement de servir de base pour l'enseignement des systèmes d'exploitation ou comme source d'inspiration pour le développement de nouveaux systèmes de type Unix, Toy Lovelace étant le premier OS à s'être inspiré de SOS !

Présentation en anglais.

Toy Lovelace: une adaptation de SOS en Ada 95, par Xavier Grave

Toy Lovelace est un projet ayant pour but de démontrer la faisabilité d'un OS en Ada 95 et de s'amuser en le faisant.

Basé sur une traduction de SOS, Toy Lovelace devrait profiter des "bienfaits" de Ada 95 que sont la gestion des exceptions, la gestion transparente des taches, bonne lisibilité du code, typage fort ainsi que la possibilité de développement OO.

La traduction de SOS est en bonne voie et l'adjonction de parties du "run-time" Ada est bien avancée (programmation orientée objet, gestion des exceptions Ada, ...).

La présentation comportera une partie d'explication du comment la traduction s'effectue grâce aux comparaisons suivantes :

- code C vs. code Ada
- compilateur C vs compilateur Ada
- code generic vs MACROS

Il sera aussi expliqué la méthode type "jeu de piste" qui permêt d'embarquer le "run-time" Ada dans le noyau.

Présentation en anglais.

Merveilleux voyage à l'intérieur d'un OS, pour les débutants, par Thomas Petazzoni

Cette présentation a pour objectif de permettre à des personnes connaissant un peu l'utilisation des systèmes de type Unix de découvrir le fonctionnement interne de ces système, et en particulier de leurs noyaux. Les grands sous-systèmes d'un noyau monolithique comme Linux seront étudiés de la manière la plus accessible possible. Au terme de cette présentation, les participants devraient comprendre les grands mécanismes d'un noyau, comme la gestion des processus, l'ordonnancement, la gestion de la mémoire virtuelle, les systèmes de fichiers ou les pilotes de périphériques.

La présentation se veut résolument didactique et orientée vers les non-connaisseurs. Réalisée par un des auteurs des articles de la série SOS publiée dans GNU/Linux Magazine France, la conférence permettra au public de poser toutes les questions qu'il se pose au sujet du fonctionnement d'un OS.

La présentation prendra une forme un peu originale: elle n'aura pas lieu en amphithéatre, mais dans une salle de travaux dirigés, pour une conférence plus ouverte et interactive. Une feuille d'inscription sera mise à disposition dans la salle de TD réservée au thème OS.

Présentation en anglais ou français, selon le public.