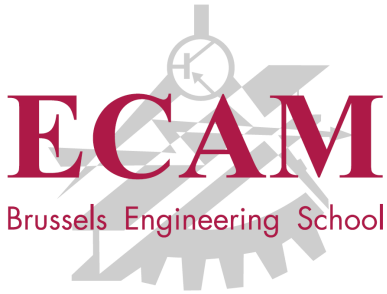


I403A Systèmes d'exploitation

Séance 15

Unix/Linux

Sébastien Combéfis, Virginie Van den Schrieck *jeudi 15 décembre 2016*

Objectifs

- Explorer l'histoire du système Unix

  Et comprendre l'évolution vers le systèmes Linux

- Examiner le modèle de processus de Linux

  Ordonnanceur, communication inter-processus...

- Examiner d'autres choix d'implémentation de Linux

  Gestion de la mémoire, système de fichiers et E/S...

Historique

- Compatibilité avec Unix désirée depuis le départ

  Existait déjà bien avant Unix

- Linus Torvalds étudiant d'une université finnoise (1991)

  Petit kernel pour les processeurs 80386, premier 32 bits

- Code source disponible sur Internet très rapidement

  Gratuitement et avec peu de contraintes pour la redistribution

Kernel, système et distribution

- Kernel Linux, développé à partir de zéro

  Logiciel développé par la communauté Linux

- Système Linux, regroupe de multiples composants

  Certains développés de zéro, d'autres provenant d'autres projets

- Distribution Linux, couche par dessus un système

  Ajout d'outils d'administration et de gestion, de logiciels...

Kernel Linux (1)

- Premier kernel Linux sorti en 1991 (version 0.01)
  - Processeurs 80386, pas de réseau, ni drivers de périphériques
  - Pages partagées (copy-on-write et espaces d'adresses protégés)
  - Un seul système de fichiers supportés, celui de Minix
- Version suivante sortie le 14 mars 1994 (Linux 1.0)
  - Ajout du réseau (TCP/IP et socket BSD)
  - Ajout drivers de périphériques (pour IP sur Ethernet)
  - Nouveau système de fichiers
  - Amélioration du support hardware (floppy, CD-ROM...)
  - IPC style System V (mémoire partagée, sémaphore...)

Kernel Linux (2)

- Développement de la version 1.1
  - Numéro de version mineure impaire pour développement
  - Numéro de version mineure pair pour stable
- Version 1.2 sortie en mars 1995
  - Support hardware (Bus PCI...)
  - Émulation de DOS (support du mode virtuel 8086)
  - Amélioration de IP (stats et firewall)
  - (Dé)Chargement dynamique de modules kernel
  - Dernier kernel limité uniquement aux PCs

Kernel Linux (3)

- Version 2.0 en juin 1996
  - Support de plusieurs architectures
  - Support du symmetric multiprocessing (SMP)
  - Amélioration performances système fichiers/mémoire virtuelle
  - Ajout des threads kernel interne
  - Ajout classe des processus temps-réel (POSIX-compatible)
- Version 2.2 sortie en 1999
  - Amélioration du réseau
  - Gestion plus fine des locus pour meilleures performances SMP

Kernel Linux (4)

- Version 2.4 et puis version 2.6
  - Meilleur support pour SMP
  - Système de fichier journalisé
  - Ordonnanceur des processus $\mathcal{O}(1)$ dans Linux 2.6
  - Kernel 2.6 préemptif, donc pour application en mode kernel
- Version 3.0 sortie en juillet 2011
  - Version majeure pour fêter les 20 ans de Linux
  - Support de la virtualisation
  - CompletelyFair Scheduler (CFS)

Système Linux

- Intégration de codes provenant d'autres projets
  - Système d'exploitation BSD (Berkeley)
  - X Window System (MIT, Massachusetts Institute of Technology)
  - GNU project (FSF, Free Software Foundation)
- GNU C compiler (GCC) directement utilisé dans Linux
- Filesystem Hierarchy Standard (FHS)

  Organisation globale d'un système de fichiers standard

Distribution Linux

- Plus besoin de compiler Linux pour l'installer sur sa machine

  Packages standards, précompilés pour faciliter l'installation

- Système Linux + utilitaires de gestion

  Par exemple, installation de paquets (format RPM très répandu)

- Première distribution en mai 1992 (SLS, Softlanding Linux System)

  Puis sortie du Slackware (gestion de packages très rudimentaire)

- Développement d'autres distributions

  RedHat, Debian très utilisées, distributions de Canonical et SuSE

Licence

- Le kernel Linux est distribué en licence GNU GPL 2.0
- Linux n'est pas un software dans le domaine public

  Le copyright est détenu par les contributeurs

- Mais Linux est un logiciel libre

  - Qui veut peut le copier, le modifier, l'utiliser comme il veut
  - Interdiction de redistribuer un dérivé sans fournir le code source

Principe de conception (1)

- Système multi-tâches à préemption, multi-utilisateurs

  Enrichi avec un ensemble d'outils compatibles Unix

- Système de fichiers et modèle réseau en accord avec Unix

- Machine multiprocesseur, Go de RAM, To de disque...

  ...mais aussi sur un système avec 16 Mo de RAM

- Principaux buts sont vitesse et efficacité

  Mais aussi standardisation comme POSIX (threads, temps-réel)

Composants

- Trois composants principaux en accord avec Unix

1. **Kernel**

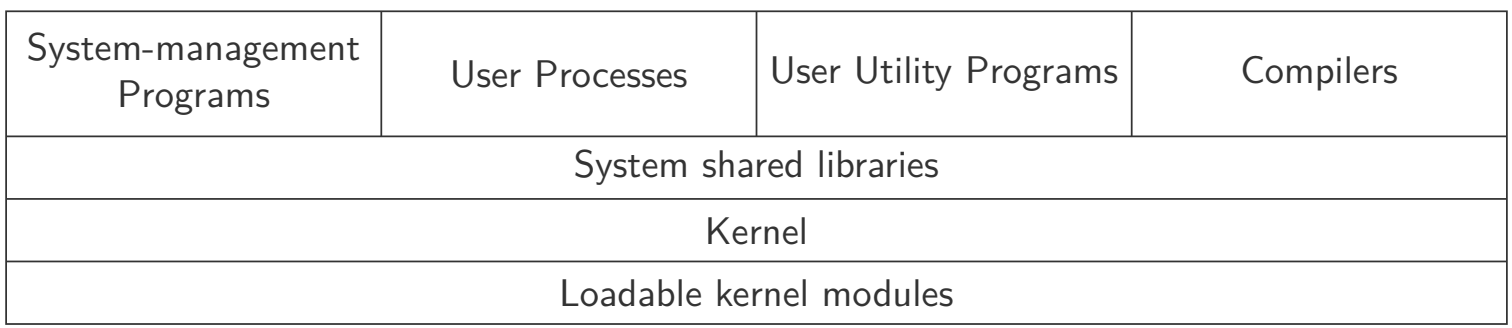
   Maintenir les abstractions du système

2. **Librairies systèmes**

   Fonctions pour interagir avec le kernel (`libc`...)

3. **Utilitaires systèmes**

   Tâches de gestion individuelle et spécialisée (démons...)



Principe de conception (2)

- Le kernel est un binaire monolithique unique
  - Pour des raisons de performance, en évitant les contexte switch
  - Communication en appelant une fonction C plutôt que IPC
- Code kernel complet dans l'espace d'adresses unique

  Ordonnancement, mémoire virtuelle, drivers, système de fichiers, réseau
- Programme utilitaire utilisateur le plus important est le shell

  Le plus commun est le bourne-Again shell (bash)

Première partie

Modules kernel

Modules kernel

- (Dé)Chargement de sections de code kernel à la demande

  Ces modules sont exécutés en mode kernel avec tous les privilèges

- Facilite le développement de drivers

  Permet aussi des drivers développés par des third-parties

- Kernel standard minimal, chargement au démarrage/au besoin

- Quatre composants pour le support des modules

  1. Gestion des modules
  2. (Dé)Chargement des modules
  3. Enregistrement des drivers
  4. Mécanisme de résolution des conflits

Gestion des modules

- Chargement du code binaire dans la mémoire du kernel

  Et vérification des références aux symboles du kernel...

- Utilisation de la liaison externe standard du langage C

- Chargement d'un module en deux phases

  - Réservation d'une zone continue de mémoire virtuelle kernel
  - Le module est ensuite passé au kernel, avec tables de symboles

- Déchargement automatique lorsqu'un module n'est plus utilisé

Enregistrement des drivers

- Appel d'une routine *startup* et d'une *cleanup*

  Garantit par le kernel au chargement et déchargement

- Le driver enregistre les fonctionnalités qu'il propose
  - **Drivers de périphérique**

    Périphériques caractères, blocs et interfaces réseaux
  - **Système de fichiers**

    Tout ce qui implémente les routines du VFS Linux
  - **Protocoles réseau**

    Protocole réseau complet tel TCP ou des règles pour Firewall
  - **Format binaire**

    Reconnaitre, charger et exécuter un nouveau type d'exécutable

Résolution des conflits

- Linux peut tourner sur n'importe quel PC

  Alors que Unix commerciaux dédicacés au hardware du vendeur

- Mécanisme central de gestion de conflits
  - Éviter clash d'un module lors d'accès aux ressources hardware
  - Empêcher des autoprobes d'interférer avec drivers existants
  - Résoudre conflits d'accès plusieurs drivers au même hardware

- Maintient d'une liste des ressources hardware allouées

  Un module doit réserver une ressource à l'avance

Deuxième partie

Gestion des processus

fork et exec

- Séparation de deux opérations habituellement combinées
  - Création d'un nouveau processus (fork)
  - Exécution d'un nouveau programme (exec)
- **Avantages**
  - Modèle très simple
  - Les deux opérations sont indépendantes
  - Pas besoin de décrire l'environnement d'un nouveau processus
- Les processus ont des propriétés rassemblées en trois groupes

  Identité, environnement et contexte

Identité du processus

- Un processus est identifié par quatre éléments
  - **PID** (*Process ID*)

    Identifie le processus auprès de l'OS
  - **Accréditation** (*Credentials*)

    Association d'un user ID et group ID pour déterminer ses droits
  - **Personnalité**

    Personnalité pouvant modifier certains appels systèmes
  - **Espace de noms**

    Vue spécifique du système de fichiers
- Le processus ne contrôle tout cela que de manière limitée

Environnement du processus

- Environnement hérité du processus parent

  À l'appel de `exec`, un nouvel environnement est créé

- Composé de deux vecteurs terminés par un `NULL`

  - **Arguments**

    Arguments de ligne de commande utilisés (et nom exécutable)

  - **Environnement**

    Liste de paires (nom, valeur) de variables d'environnement

- Par exemple `TERM` et `LANG`

  Type de terminal de l'utilisateur et langue

Contexte du processus

- Le contexte du processus représente l'état de son exécution
  - **Contexte d'ordonnancement**

    Information pour suspendre et reprendre un processus
  - **Comptabilité**

    Ressources actuellement consommées et depuis le démarrage
  - **Table des fichiers**, *les descripteurs de fichier*
  - **Contexte du système de fichiers**, *root, working et namespace*
  - **Table des gestionnaires de signaux**

    Actions à entreprendre pour des signaux pouvant être reçus
  - **Contexte mémoire virtuelle**

Processus et thread

- `fork` crée un nouveau processus et `clone` un nouveau thread

  Pas de différences pour Linux, processus et thread = tâches

- `clone` permet de partager des ressources avec le parent

`CLONE_FS`	Informations sur le système de fichiers
`CLONE_VM`	Espace mémoire
`CLONE_SIGHAND`	Gestionnaires de signaux
`CLONE_FILES`	Ensemble des fichiers ouverts

Ordonnancement

- Linux supporte le multitâche avec préemption, comme Unix
- L'ordonnanceur de processus décide quel processus s'exécute

  Compromis entre équité et performance

- Exécution et interruption de processus utilisateur

  Sans oublier les tâches kernel (demandes utilisateur et internes)

Ordonnancement des processus

- Tâches routinières en temps partagé

  Ordonnanceur équitable et préemptif

- Deux ordonnanceurs : $\mathcal{O}(1)$ (Linux 2.5) et CFS (Linux 2.6)
  - Deux niveaux de priorité : 0–99 temps réel, nice de -20 à 19
  - Processus reçoit une proportion du temps processeur

- CFS utilise la latence visée pour calculer les time slots

  Tout processus doit avoir été exécuté une fois endéans ce temps
  Et CFS se base aussi sur la granularité minimale

Ordonnancement temps-réel

- Deux ordonnanceurs temps-réel dans Linux (Posix.1b)

  Les deux classes requises sont FCFS et RR 

- Exécution du processus avec la plus haute priorité

  Pour la même priorité, choix de celui qui a la plus longue attente

- Ordonnanceur temps-réel soft sous Linux

  Garanties par rapport aux priorités relatives seulement

Synchronisation kernel (1)

- Sycnhronisation des opérations du kernel

  De manière différente que l'ordonnancement des processus

- Entrée en exécution mode kernel de deux manières
  - Un programme en cours fait appel explicite (appel système) ou implicite (défaut de page...)
  - Réception d'une interruption (d'un contrôleur de périphérique...)

- Framework pour ne pas violer l'intégrité des données partagées

  De plus, le kernel est préemptif depuis Linux 2.6

Synchronisation kernel (2)

- Utilisation de spinlocks et sémaphores
  - Monoprocesseur : activation/désactivation de la préemption
  - SMP : spinlock détenu que pour de courtes durées (sinon sémaphore)
- Chaque tâche a un compteur dans la structure `thread_info`

  Si > 0 *pour tâche en cours d'exécution, ne pas préempter*

- Activation/désactivation des interruptions

  Baisses des performances à cause suspension E/S

Multiprocessing symétrique

- Exécution de processus sur des processeurs différents

  Initialement kernel code sur un seul processeur à la fois

- Introduction du big kernel lock (BKL) en Linux 2.2

  Processus concurrents sur plusieurs processeurs dans le kernel

Troisième partie

Gestion de la mémoire

Gestion de la mémoire

- Deux composantes à la gestion de mémoire
- Allocation et libération de la mémoire physique

  Pages, groupes de pages, petits blocs de RAM
- Gestion de la mémoire virtuelle

  Mapping sur les processus en cours d'exécution

Gestion de la mémoire physique

- Séparation mémoire physique en quatre zones

  Zones spécifiques à l'architecture

`ZONE_DMA`	$<$ 16 Mo
`ZONE_DMA32`	
`ZONE_NORMAL`	16...896 Mo
`ZONE_HIGHMEM`	$>$ 896 Mo

- `ZONE_DMA` et `ZONE_DMA32` pour certains périphériques
- `ZONE_HIGHMEM` non-mappée dans l'espace mémoire kernel
- Pages libres dans chaque zone maintenue par le kernel

Allocateur de page

- Un allocateur de pages par zone mémoire
- Utilisation d'un système de type buddy
  - Unités adjacentes de mémoire allouable pairées
  - Deux unités pairées libérées sont fusionnées en une plus grande
  - Une unité peut aussi être découpée en deux unités plus petites
- Plus petite unité correspond à une page physique

Slab

- Slab utilisé pour allouer des structures de données kernel

  Une ou plusieurs pages physiques contigües

- La cache consiste en un ou plusieurs slabs

  Un cache unique par structure (process descriptor, inodes...)

- Trois états possibles pour les slabs
  - **Plein**, tous les objets du slab sont utilisés
  - **Vide**, tous les objets du slab sont libres
  - **Partiel**, objets utilisés et libres

Mémoire virtuelle

- Contrôle de l'espace d'adresses de chaque processus
- Création de pages de mémoire virtuelle, à la demande

  Chargement depuis le disque, et swap back vers le disque

- Deux vues de l'espace d'adresses d'un processus
  - **Vue logique**

    Ensemble de régions qui ne se chevauchent pas
  - **Vue physique**

    Stocke la localisation précise des pages virtuelles

Région de la mémoire virtuelle

- Plusieurs types de régions de mémoire virtuelle
- Stockage de l'origine des régions
  - Un fichier
  - Rien du tout (demand-zero memory)
- Réaction à l'écriture
  - **Privé**, copy-on-write à active
  - **Partagé**, écriture immédiate

Durée de vie de l'espace d'adresses virtuel

- Création d'un espace d'adresses virtuel dans deux situations
  - Lors d'`exec`, nouvel espace vide créé
  - Lors de `fork`, copie complète de l'espace existant du parent
- Cas particulier pour les pages privées, copy-on-write

Swapping et paging

- Sortir les pages du disque lorsqu'elles sont nécessaires

  Déplacement de pages de mémoire virtuelle individuelles

- Linux déplace uniquement des pages

  Unix swappe le contenu complet d'un processus

- Système de pagination en deux parties
  - **Algorithme de politique**, quelle page envoyer au disque
  - **Mécanisme de pagination**, fait le transfert des pages

- Âge associé aux pages pour politique de type LFU

Mémoire virtuelle kernel

- Zone de l'espace d'adresses virtuel réservé pour le kernel

  Marqué comme protégé, invisible et non modifiable en user mode

- Deux régions dans cette zone
  - Références aux tables de page pour pages physiques disponibles
  - Zone libre pour référer à d'autres zones en mémoire