M1 SAR 2017-2018
Programmation Noyau Linux

Kevin Mambu

February 8, 2018

Contents

1 Présentation

Il s'agit d'une UE très exigeante sur la maîtrise d'un noyau et d'oeuvres avancées du C.
L'objectif de l'UE est de maîtriser à terme le développement au sein d'un Noyau Linux, le déboguage avec des outils adaptés et la programmation par patch et modules.

- Examen final : 40%
- Projet & Exercices : 60%

Deux choses à noter :

- Ce cours serait un très obn cours de génie logiciel : plusieurs milliers de contributeurs, un code très stable et viable car codé de manière très codifiée.
- Même sans développer dans le noyau Linux plus tard : il est important de connaître et de comprendre le système d'exploitation et les outils & bonnes pratiques.

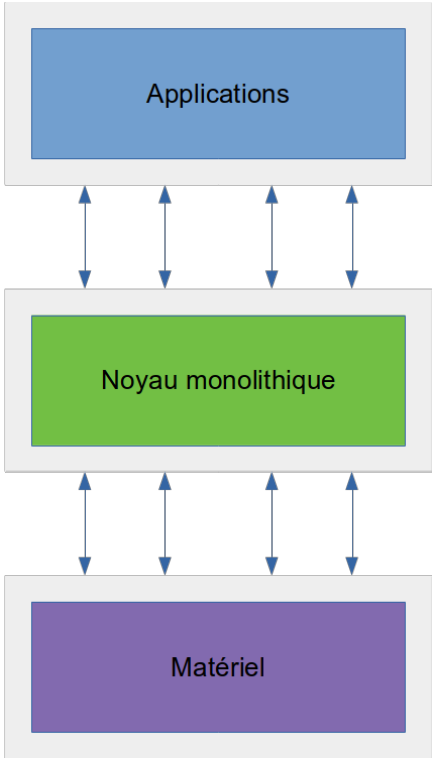
1.1 L'origine de "Unix"

Le nom UNIX fait réference au système d'exploitation MULTICS (MULTiplexed Information and Computing System) issu d'une collaboration entre le MIT, Bell & General Electric. Plusieurs rumeurs tounre autour de l'origine du mot.

2 Systèmes d'Exploitation - Rappels

Qu'est ce qu'un système d'exploitation, en essence? $\Rightarrow$ Un gestionnaire d'interruption. Il y a quatres types de Noyau:

2.1 Noyaux monolithiques



Un noyau compilé statiquement, sur un seul binaire (UNICSV6 par exemple).
WARN : un noyau monolithique ne veut pas dire un code monolithique Le noyau fournit aux applications utilisateurs une abstraction concernant les abstractions du matériel. Communiquer de l'application au matériel se passe par le biais d'interruptions ⇒ mapper différentes interruptions à différentes actions.

Un OS se distingue par deux modes d'exécution différents : le mode système et le mode utilisateur. Le premier possède des avantages tels que :

- Zones mémoire restreintes
- Instructions privilégiées
- Pile spécifique
- Accès à certains registres

Intérêts :

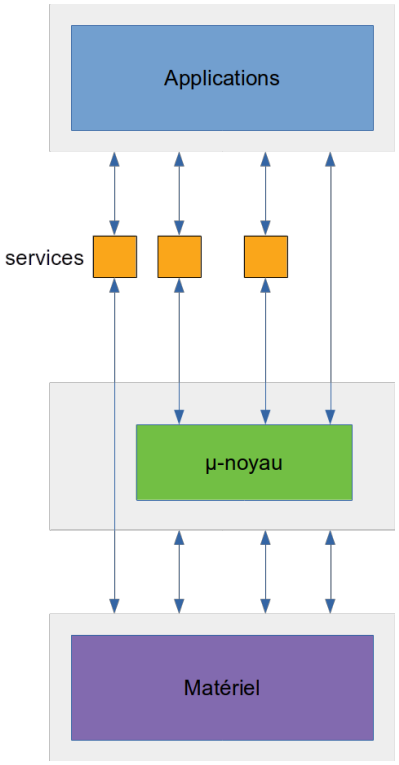
- Facile à optimiser car sur la même codebase ⇒ performant
- Facile à sécuriser

Inconvénients :

- Empreinte mémoire assez importante quand non-dédiée
- Beaucoup de fonctionnalités ⇒ Beaucoup de failles potentielles

Utilisation du noyau monolithique ⇒ embarqué, applications RT ou spécifique.

2.2 Micro-noyaux



Idée : Réduire le noyau à un set de fonctionnalités fondamentales, et mettre les autres services en mode utilisateurs. On réduit la surface d'attaque du noyau en dispatchant dans le mode utilisateur (exemple : filesystems)

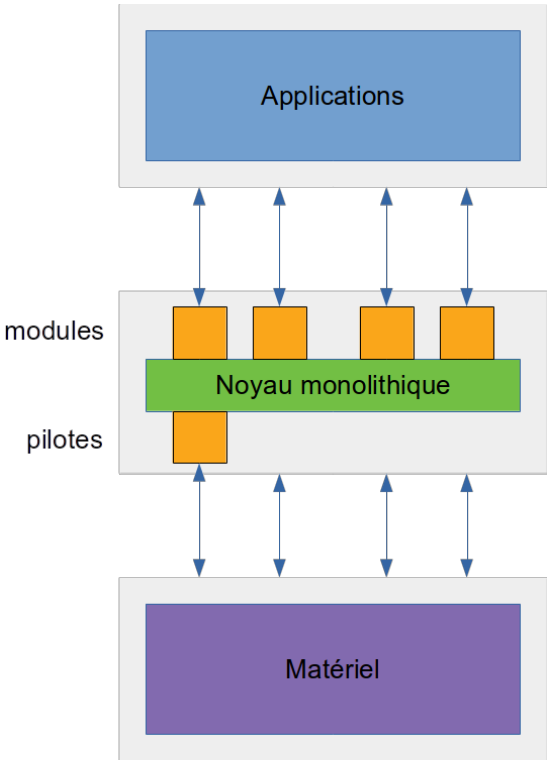
Intérêts :

- Empreinte mémoire très réduite
- Surface d'attaque très restreinte
- Fonctionnalités en mode utilisateur ⇒ execution en mode utilisateur

Inconvénients :

- Beaucoup d'appels systèmes ⇒ impact sur les performances
- Codebase dispatchée ⇒ plus difficile à tenir à jour

2.3 Noyaux modulaires monolithiques



Avoir un noyau monolithique auquel on peut charger des modules. Intérêts :

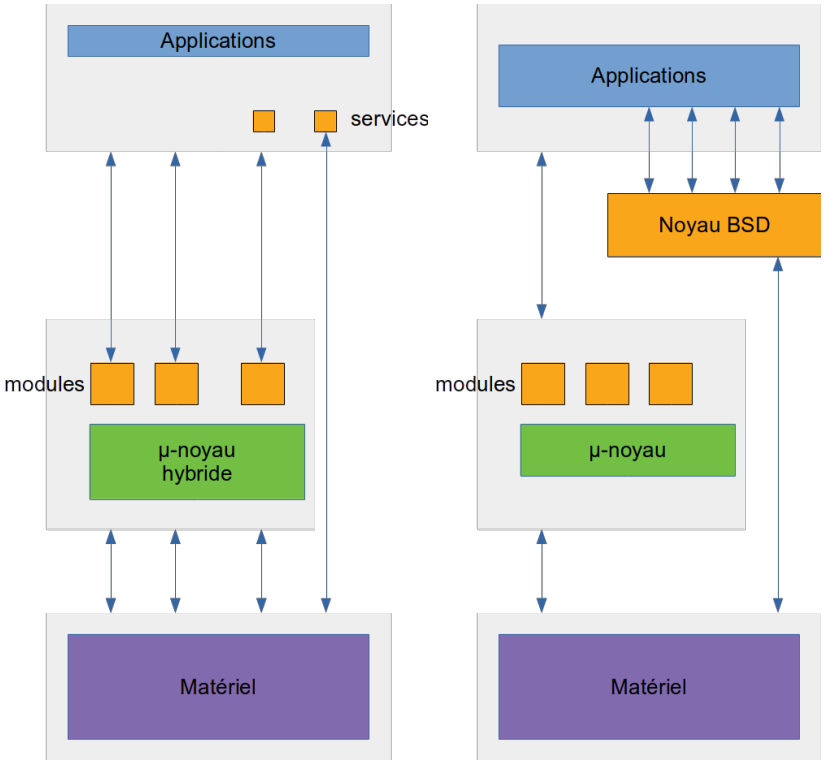
- Interaction directe avec le matériel optimisé
- Possibilité d'opti pour une architecture
- facile d'extension et de portabilité
- Modules directement en mode privilégié

Inconvénients :

- Pas très portable
- Modules tiers en mode privilégié ⇒ sécurité compromise!

Nb : il est possible de hard link des modules pour faire de la programmation monolithique.

2.4 Micro-noyaux modulaires

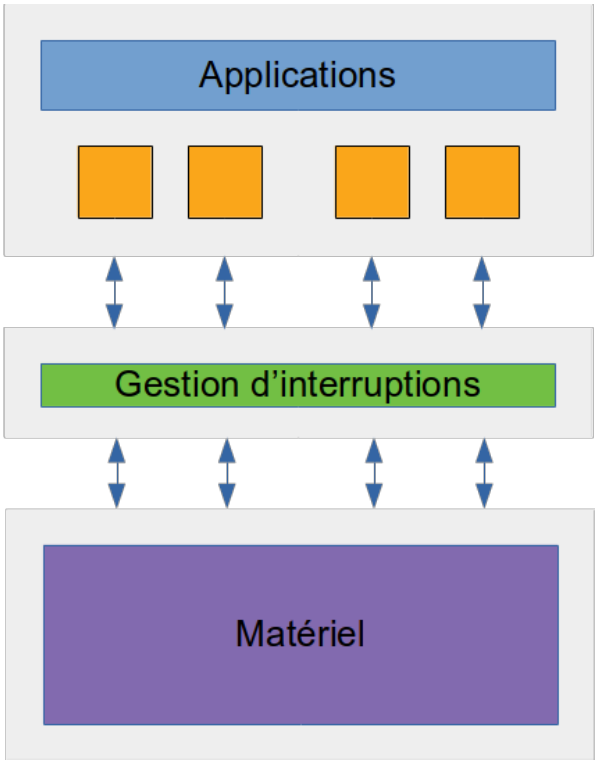


Possibilité d'avoir d'une part des services haut-niveau opérés en mode utilisateur et un micro-noyau modulaire. Cela permet de combiner les avantages de modularités des micro-noyaux et des noyaux monolithiques modulaires.

On pourrait dire que parce que glibc est l'API de choix des développeurs sous Linux, Linux est quelque part un micro-noyau modulaire.

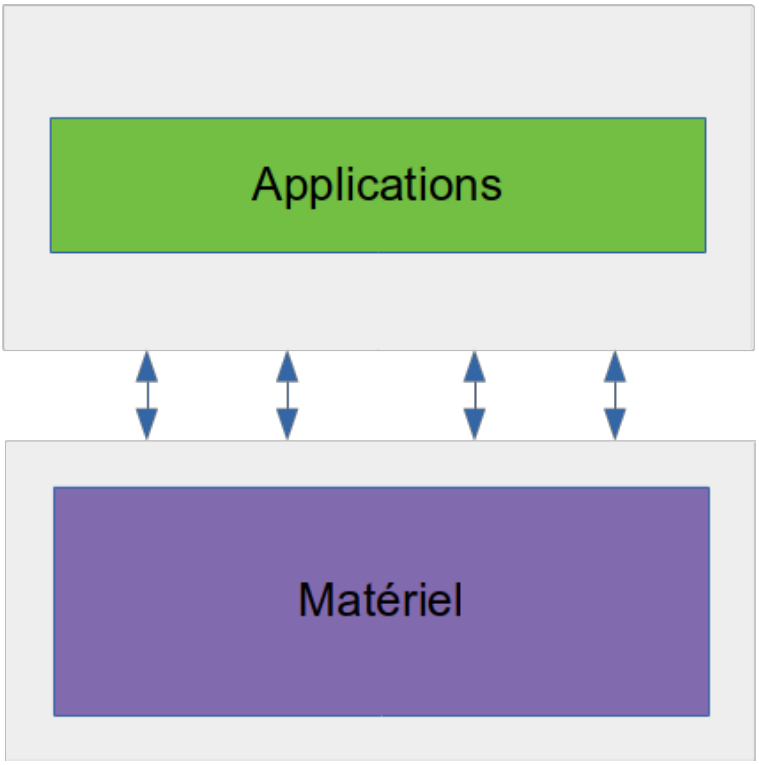
Pourquoi BSD : problème de licence. Si on utlisait GNU Linux, il y aurait obligation de laisser les codes sources sous GNU libres.

2.5 Exo-noyau



Le noyau est réduit à l'interception et redirection d'interruptions. Les applications ont la charge de la gestion de la mémoire et des périphériques.

2.6 Uni-Kernel



Une seule application en exécution dans le noyau. En vogue dans le domaine des systèmes embarqués. Ces uni-kernels sont ensuite encapsulées en VM, et chaqu'une possédant son propre espace mémoire.

3 Linux : Une architecture en oignon

3.1 Explicatoin du schema

Chaque point représente chaque composante du noyau et chaqu'une de ses dépendances. ⇒ On parle là d'une structure en oignon, avec au centre le noyau et plus on s'éloigne, plus on monte en niveau d'abstration.
Même das le cas d'un μ-noyau, on parle d'un μ-noyau enrichi lorsqu'il est appairé à l'ensemble de ses services (ex : Mach(enrichi) = NoyauBSD + μ-noyau).
Pour comprendre cette structure, on va énumérer 5 services offerts par le noyau :

- La gestion du stockage.
- La gestion des processus (execution concurrente, multi-programmation).
- La gestion de la mémoire.
- La gestion du réseau : la pile IP est fournie par le noyau.
- L'interface avec l'utilisateur*.

De ce partitionnement, on peut modéliser un plan du noyau, une **Kernel Map**.

	Système	**Réseau**	**Stockage**	**Mem**	**Processus**
API Apps/Kern	Syscalls	Sockets	Fichiers	Accès Mem/mmap	Processus
Services Abstr.	Kobject/Submodule	Familles Prot.	VFS*	Vmalloc	Thread
Services	Modules	Protocoles	Filesystems	Kmalloc	Scheds
HW Abstraction	Generic HW	Device Res	Blocks	PFRA	Interrupt
API Kern/Hard	Driver Bus	Driver Carte Res	Driver Disk	Table des Pages	CPU-specific
Matériel	Bus (USB/PLIE)	Cartes Res.	Disk/Band	RAM/MMU/TLB	CPU

Abstraction d'une K-Map

Il y a des interactions entre colonnes mais on retrouve la même structure au sein d'un même colonne.
Vers le bas de la map, on va trouver les choses les plus bas-niveau, e.g. quels matériels vont être gérés par le noyau.
Nb : GMail omet progressivement la notion de répertoire au profit de tags et de métadonnées. Il existe d'aiileurs des propositions pour changer les arborescences de fichiers pour des bases de données.
Factoriser du code au niveau des drivers est très fastidieux, on a donc intérêt à laisser de la marge.
→ utilisation d'abstractions
PFRA : (Page Frame Reclaim Allocator) réclamation de mémoire
Tâche : Programme + Données à traiter.
Processus : Fils d'exécution, une famille de threads avec la particularité de partager le même tas.
VFS : Virtual Filesystem (DirEntry, Inode) (WARN : ne pas confondre le VFS, les codes génériques de tous les filesystems avec les RAMFS, les filesystems **réellement** virtuels)
NFS : TODO
Page Cache : Cache des processus, les processus sont chargés en mémoire.

Encapsulation de données : Bloc → Page → Page Frame
Dans les processus Intel : il est possible d'associer en Hardware des PIDs (utilisé pour le correctif Meltdown)