

# Systemes d'exploitation

Introduction

**Pilot Systems** - Gaël LE MIGNOT

INSIA SRT - 2007

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Présentation du cours . . . . .	3
1.1.1	Objectif du cours . . . . .	3
1.1.2	Contenu du cours . . . . .	3
1.2	Rappels historiques . . . . .	3
1.2.1	Historique générale . . . . .	3
1.2.2	Histoire de Multics et d'Unix . . . . .	4
1.2.3	Quelques dates importantes . . . . .	4
1.3	Architecture générale du matériel . . . . .	4
1.3.1	La machine de Turing . . . . .	4
1.3.2	L'architecture classique . . . . .	5
1.3.3	Le fonctionnement d'un CPU . . . . .	5
1.3.4	Les niveaux de protection . . . . .	5
1.3.5	Les périphériques . . . . .	5
1.3.6	La communication avec les périphériques . . . . .	5
1.4	Rôles d'un système d'exploitation . . . . .	6
1.5	Catégorie de systèmes d'exploitation . . . . .	7
1.5.1	Types d'environnement . . . . .	7
1.5.2	Architectures des systèmes . . . . .	7

# Table des figures

1.1	Traitement par lots . . . . .	4
1.2	Exemple de bus et de bridges . . . . .	6
1.3	Noyau monolithique . . . . .	8
1.4	Micro-noyau à serveur unique . . . . .	9
1.5	Micro-noyau à serveurs multiples . . . . .	10

# Chapitre 1

## Introduction

### 1.1 Présentation du cours

#### 1.1.1 Objectif du cours

- Un administrateur système doit souvent optimiser, dimensionner, choisir et/ou trouver les points de blocage d'un système ;
- Il est nécessaire de comprendre le fonctionnement interne d'un système d'exploitation ;
- L'objectif n'est pas de développer un système, mais bien d'en comprendre le fonctionnement.

#### 1.1.2 Contenu du cours

- Une présentation des différents principes, architectures et composants d'un système d'exploitation ;
- Une présentation rapide de certains algorithmes fondamentaux ;
- Le plus souvent, les exemples viendront du monde Unix, mais pas uniquement.

### 1.2 Rappels historiques

#### 1.2.1 Historique générale

##### Première génération (1945-55)

C'est l'époque des lampes à vides et des cartes perforées : les ordinateurs sont extrêmement volumineux, coûtent extrêmement cher, et n'effectuent qu'un seul traitement à la fois. Ils nécessitent une intervention manuelle entre chaque traitement. Aucun système d'exploitation au sens usuel du terme n'est présent.

##### Seconde génération (1955-65)

L'apparition des transistors a permis d'augmenter considérablement la puissance des ordinateurs, mais ils restent très onéreux. Le mode d'opération est le traitement par lot : les traitements sont effectués à la suite. Vu le coût élevé des machines dotées d'une puissance suffisante pour effectuer les traitements, elles sont en général dotées en amont et en avalant de machines moins puissantes, dédiées aux entrées-sorties (impression, ...). Un opérateur collecte les tâches à effectuer sur une ou plusieurs bandes au niveau d'un gestionnaire d'entrée, puis transfère manuellement la bande à l'unité de traitement. Celui-ci effectue son traitement, puis un opérateur transmet le résultat au gestionnaire de sortie.

##### Troisième génération (1965-1980)

Les premiers circuits intégrés permettent l'apparition de nouvelles machines, beaucoup plus puissantes. C'est l'époque des *main-frame*, et l'apparition de la *multiprogrammation* : plusieurs programmes sont exécutés en parallèle. L'utilisation se fait en général par le biais de terminaux, plusieurs utilisateurs utilisant la même machine en même temps.

##### Quatrième génération (1980-maintenant)

Les microprocesseurs deviennent suffisamment petits, puissants et bon marché pour permettre le développement de l'informatique grand public.

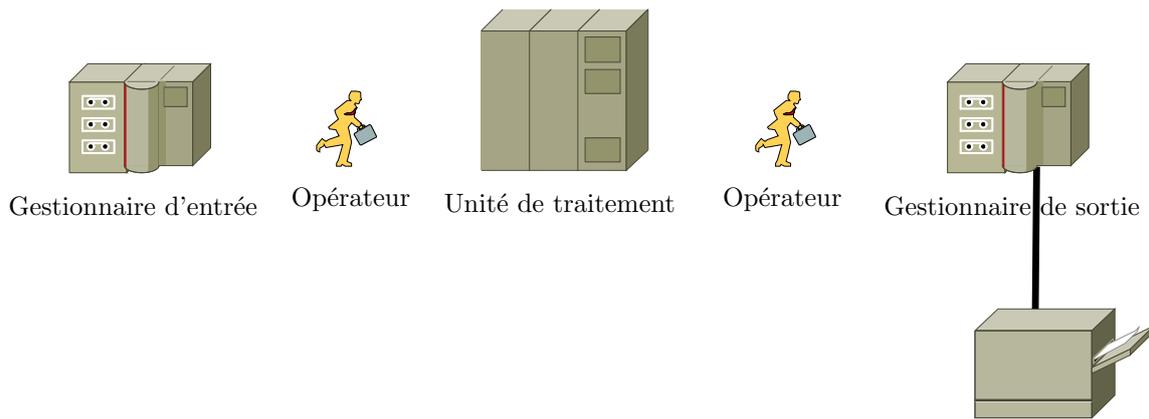


FIG. 1.1 – Traitement par lots

## 1.2.2 Histoire de Multics et d'Unix

- Multics (*MULTIplexed Information and Computing Service*) fut lancé en 1965, par un consortium composé des Bells Labs, le MIT et la General Electric, pour fournir aux utilisateurs un service informatique sous une forme comparable à la distribution d'électricité : des "centrales" de serveurs fournissent le service, et les utilisateurs payent à la consommation.
- Malgré sa complexité et son démarrage difficile, Multics supportait des milliers d'utilisateurs sur une machine comparable à un 386, et a été utilisé jusqu'en 2000.
- Unix fut alors créé par les Bells Labs d'AT&T, afin d'être une version moins ambitieuse et plus réaliste de Multics.
- Unix fut repris par de nombreuses entités (entreprises, universités, particuliers) et a donné le jour à deux grandes familles : les Unix System V et les Unix BSD.
- Le projet GNU, lancé en 1984 par Richard Stallman, visait tout d'abord à fournir un système d'exploitation conforme aux 4 libertés fondamentales du Logiciel Libre, mais aussi à s'inspirer d'Unix, tout en dépassant les limites (d'où le nom, GNU is Not Unix).

## 1.2.3 Quelques dates importantes

- 1947 Prototype du transistor ;
- 1965 Multics ;
- 1966 IBM 360 ;
- 1969 Première version d'Unix ;
- 1971 Premier microprocesseur commercial, le 4004 ;
- 1981 Premier IBM PC ;
- 1984 Premier Macintosh ;
- 1984 X-Window System ;
- 1984 Lancement du projet GNU.

## 1.3 Architecture générale du matériel

### 1.3.1 La machine de Turing

La machine de Turing est le modèle à la base des ordinateurs. Elle se compose de trois parties :

- Une bande (magnétique en général) de taille infinie ;
- Une tête de lecture/écriture positionnée sur la bande ;
- Un mécanisme (automate fini déterministe), qui, suivant la valeur lue sur la bande, et de son état interne, peut modifier la valeur de la bande, déplacer la tête et/ou modifier son état.

Le problème de la machine de Turing est qu'elle ne résout qu'un seul problème : le mécanisme est au niveau matériel, et les données uniquement sont magnétiques. Pour résoudre un autre problème, il faut la modifier physiquement.

L'étape suivante consiste à créer une machine de Turing capable de lire son programme depuis la bande magnétique. Une telle machine capable d'effectuer le traitement de n'importe quelle autre machine de Turing

est dite “universelle”.

### 1.3.2 L’architecture classique

L’architecture classique des ordinateurs actuels est dite de Von Neumann, du nom de l’inventeur de ce modèle. Elle est une évolution de la machine de Turing, et se compose elle aussi de trois parties :

- Une mémoire centrale (*RAM*) permet d’accéder arbitrairement à des données ;
- Un *CPU* qui exécute des instructions, se trouvant dans cette mémoire, et pouvant lire et écrire cette mémoire ;
- Un (ou plusieurs) *bus* permet de communiquer avec les périphérique.

### 1.3.3 Le fonctionnement d’un CPU

- Le CPU possède des *registres*, variables internes, en nombre réduit (d’une dizaine à une centaine) ;
- Le CPU charge, décode et exécute l’instruction se trouvant à l’adresse mémoire pointée par un registre particulier, *IP* (*Instruction Pointer*) ;
- Il passe ensuite à l’instruction suivante ;
- Les instructions peuvent modifier la valeur d’*IP*, afin d’effectuer un *saut* (par exemple pour une boucle ou une condition).

### 1.3.4 Les niveaux de protection

- Ce qui est fait au niveau logiciel peut être défait au niveau logiciel ;
- Pour permettre la multiprogrammation, le matériel doit définir des mécanismes de protection ;
- En général, on se contente de deux niveaux : un mode *kernel* et un mode *user*.

### 1.3.5 Les périphériques

Il en existe de beaucoup de types : écran, imprimante, disque dur, réseau, son, graveur, ..., mais sur le principe, presque tous sont destinés à émettre ou recevoir des données. Les détails techniques changent énormément suivant les périphériques.

Par exemple, pour lire un secteur sur une disquette, il faut :

1. Mettre en marche le moteur ;
2. Attendre que le moteur soit à sa vitesse optimale ;
3. Déplacer les têtes vers le cylindre considéré ;
4. Vérifier que les têtes sont en face du bon cylindre, sinon, effectuer une opération spéciale (recalibrer les têtes) ;
5. Attendre que le bon secteur passe sous la tête de lecture ;
6. Lire les données, en connaissant la taille d’un secteur, du préambule et de l’espacement entre les secteurs ;
7. Vérifier que les données sont correctes, via les différents mécanismes de contrôle et correction d’erreur ;
8. Si les données ne sont pas correctes, réessayer.

Il est évident qu’aucun programmeur ne souhaite effectuer ça à chaque lecture sur une disquette, surtout que le mécanisme est sensiblement différent pour un disque dur ou une mémoire flash USB.

### 1.3.6 La communication avec les périphériques

#### Les bus et les bridges

Tout d’abord quelques définitions :

**Bus** Un bus est un canal de communication entre différentes parties de la machine. En général, tout ce qui est écrit sur un bus est vu par tous les composants partageant ce bus ;

**Bridge** Un bridge est un pont entre deux bus, qui va retransmettre des données d’un bus à l’autre, éventuellement en filtrant. Par exemple, un contrôleur USB est un bridge, qui lit le bus PCI, et retransmet une partie des informations (celles qui concernent les périphériques USB) sur le bus USB.

Dans le cadre d’un ordinateur récent, il existe un grand nombre de bus inter-connectés, par exemple, on peut avoir un bus PCI sur lequel est connecté un contrôleur USB lui-même gérant un bus USB capable de communiquer avec différents périphériques USB.

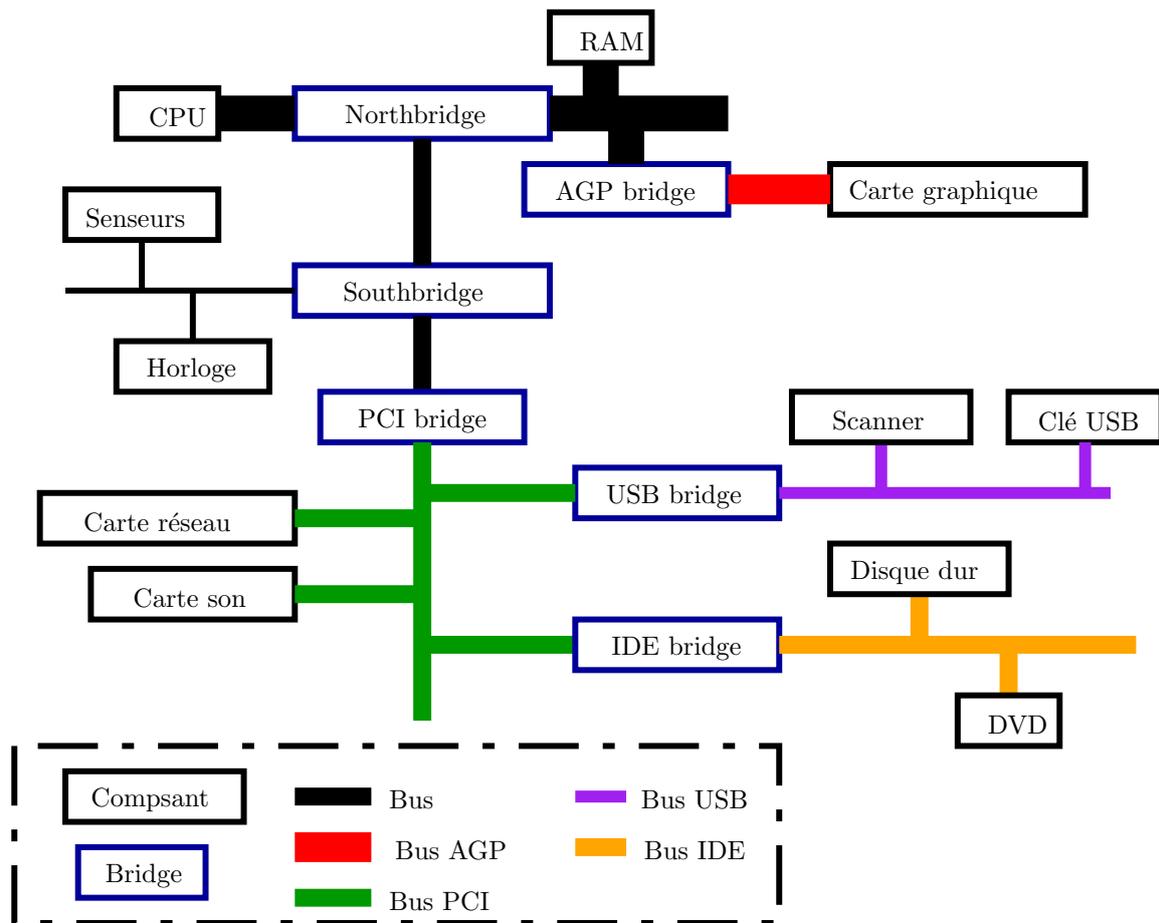


FIG. 1.2 – Exemple de bus et de bridges

### Les modes de communication

Il existe deux moyens de communiquer avec un périphérique :

- Le mode PIO (programmed IO) consiste à ajouter deux instructions spécifiques `in` et `out` dans le microprocesseur. Ces instructions prennent en paramètre un numéro de port, et un emplacement (en général un registre) où lire/écrire les données. Les données sont envoyées sur un bus, et chaque périphérique regarde si le numéro de port les intéresse.
- Le deuxième mode MMIO (memory-mapped IO) consiste à utiliser des adresses mémoires spécifiques, par exemple la plage située entre 640Ko et 1Mo pour les architectures x86. À chaque fois sur le processeur demande à lire ou à écrire des données dans ces plages spécifiques, un bridge spécial (le PCI bridge par exemple) les envoie sur le bus général des périphériques.

Le mode MMIO est plus agréable à utiliser, puisqu'il permet d'utiliser des pointeurs et des variables normales lorsqu'on écrit un programme, tandis que le mode PIO nécessite des instructions particulières. Cependant, le mode PIO est plus simple à réaliser au niveau du matériel.

Dans les deux cas, il est nécessaire de connaître l'adresse du périphérique (soit son numéro de port, soit l'adresse mémoire qui le concerne) afin de communiquer avec lui. Historiquement, les adresses étaient réglées via des cavaliers sur les cartes elles-mêmes, et configurées manuellement par l'utilisateur. De nos jours, des systèmes d'auto-détection et d'auto-configuration permettent d'éviter ces opérations manuelles (au prix d'erreurs parfois étranges).

## 1.4 Rôles d'un système d'exploitation

Il y a deux principales vision du rôle d'un système d'exploitation. La vision *top-down* consiste à dire que le système d'exploitation assure une abstraction vis à vis des détails du matériel, cachant toute la complexité que nous avons vu plus haut dans des appels simples (`open`, `read`, `write`). Dans cette vision, le système peut aussi

fournir des services qui ne sont pas directement liés au matériel, comme des protocoles réseaux, . . .

La deuxième vision, *bottom-up*, consiste à voir le système d'exploitation comme le gestionnaire des différentes ressources (mémoire, CPU, disques, périphériques) dans un environnement multi-processus et multi-utilisateurs, répartissant les ressources entre les programmes et les utilisateurs, et appliquant la politique de sécurité décidée par l'administrateur.

Vu que la division exacte entre ce qui est effectué en espace utilisateur et ce qui est effectué en espace noyau (la division au niveau matériel que l'on a vu plus haut) est parfois changeante (la gestion de l'USB sous MacOS X est en grande partie faite en espace utilisateur, par exemple), et que pour certains systèmes (comme le DOS) ils ne sont même pas utilisés, on évite en général de confondre "noyau" (ce qui est exécuté avec des droits particuliers vis à vis du matériel) et "système d'exploitation" (le ou les programmes qui gèrent l'abstraction vis à vis du matériel et répartissent les ressources).

## 1.5 Catégorie de systèmes d'exploitation

### 1.5.1 Types d'environnement

La première façon de classer les systèmes d'exploitation est de regarder le type d'environnement pour lequel le système est conçu. Les types d'environnement généralement reconnus sont les suivants :

**Traitement par lot** Ce type d'environnement est peu utilisé de nos jours, mais peut encore exister pour des besoins spécifiques.

**Serveurs et main frames** Les gros serveurs équipés de multiples processeurs, de grandes quantités de mémoire et de périphériques à très haut débits sont utilisés couramment dans les environnements de production, et ont des problématiques spécifiques : gestion d'un grand nombre de ressource, tolérance de faille, sécurité.

**Machines de bureaux** Les machines de bureaux, qui incluent aussi les ordinateurs portables et parfois de petits serveurs, sont les machines les plus répandus ; elles ont des problématiques spécifiques de réactivité, de multimédia et d'auto-détection du matériel.

**Embarqué et temps-réel** Enfin, la dernière catégorie est l'informatique embarquée, de l'enregistreur de DVD au GPS des voitures, des montres aux équipements de contrôle des centrales nucléaires. Les problématiques sont souvent la faible puissance du matériel, ainsi que la nécessité de répondre aux problèmes dans un délai fixe.

### 1.5.2 Architectures des systèmes

La deuxième façon de classer les systèmes est de regarder la structure interne des systèmes.

#### Les systèmes sans protection

Systèmes les plus simples (comme MS-DOS, mais encore utilisés parfois dans l'embarqué) n'utilisent aucune protection matérielle ; les programmes utilisateurs fonctionnent comme le noyau, et doivent souvent gérer un grand nombre de choses par eux-mêmes.

#### Les systèmes à noyau monolithique

Il s'agit de la première génération de systèmes d'exploitation avec protection, où la quasi-totalité du système réside dans un seul programme, le noyau, ayant tous les privilèges.

Le noyau doit contenir tous les gestionnaires de périphériques, les systèmes de fichiers, les couches réseaux (comme TCP/IP) et peut être très conséquent (le code source de Linux 2.6.22 fait 300 Mo, pour plus de 5 millions de lignes de code).

Historiquement, le noyau était compilé de manière statique, et il fallait recompiler le noyau pour ajouter un périphérique. Dans les systèmes plus récents, un mécanisme de modules permet de charger, dynamiquement, des modules dans le noyau. Ce mécanisme est particulièrement nécessaire avec les technologies comme l'USB qui permettent de changer les périphériques à chaud.

Mais les modules restent tous dans le même espace d'exécution protégé, et un bug dans le driver de la carte son pour corrompre des données dans le driver du disque dur. Ou pire.

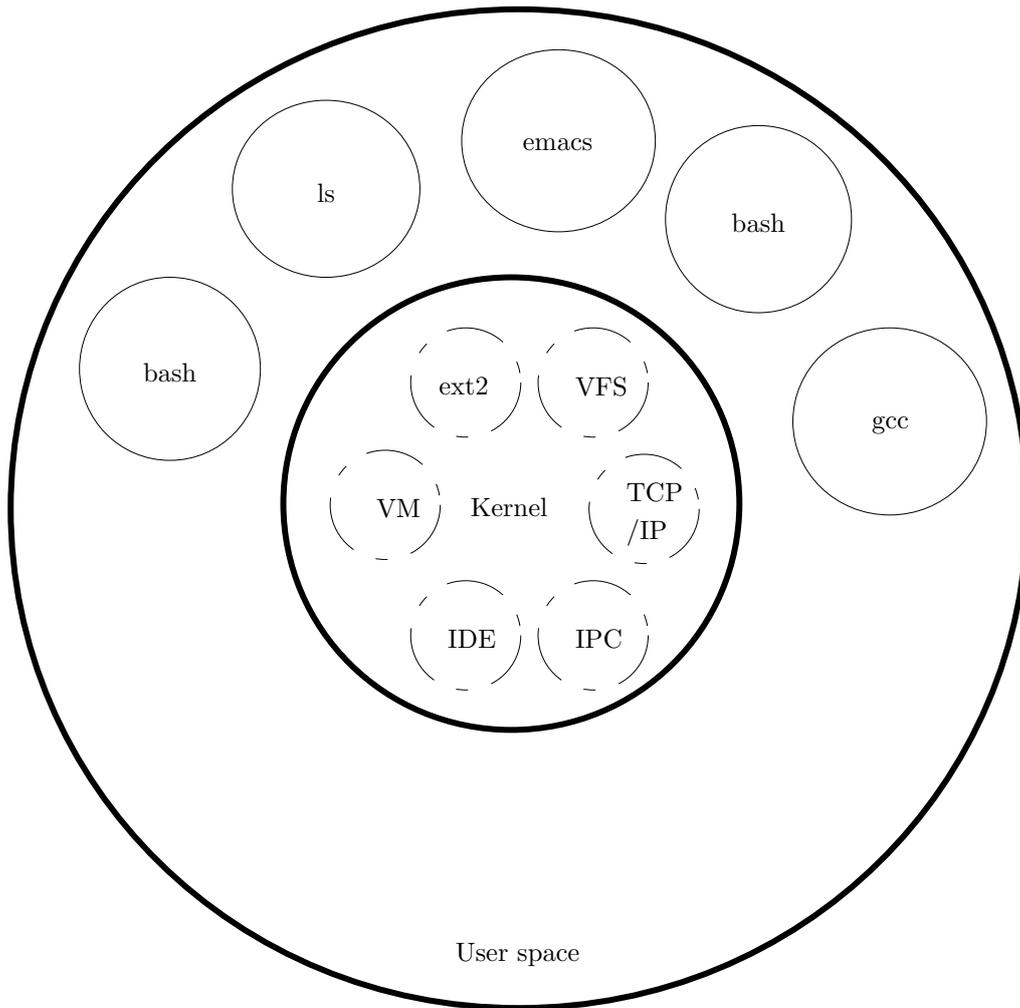


FIG. 1.3 – Noyau monolithique

### Les systèmes à micro-noyau

Afin de pallier à ces problèmes, le concept de micro-noyau a été développé dans les courants des années 1980. Il consiste à vouloir réduire la taille du noyau le plus possible, et à mettre en espace utilisateur tout ce qui peut y être mis (par exemple, un système de fichier ou une pile réseau ne nécessite pas de privilège vis à vis du matériel).

La première étape consiste à créer un seul serveur, fonctionnant au-dessus du micro-noyau. Des portages du noyau Linux en espace utilisateur au-dessus d'un micro-noyau ont par exemple été réalisés, ce sont les projets MkLinux et L4Linux.

La deuxième étape consiste à découper ce serveur unique en une multitude de serveurs, chacun s'occupant d'une tâche et d'une seule, et communiquant entre eux. Un exemple de projet utilisant cette approche est le GNU Hurd.

Jusqu'à quel point peut-on mettre le code en espace utilisateur ? Là-dessus, deux réponses sont possibles. La première réponse, des micro-noyaux de première génération comme Mach, consiste à laisser en espace noyau tout ce qui a besoin d'accès au matériel (les pilotes de périphériques, l'ordonnanceur, la gestion de la mémoire) et de mettre en espace utilisateur ce qui est de plus haut niveau (les systèmes de fichiers, les couches réseaux). La deuxième réponse, des noyaux de deuxième génération comme L4, consiste à découper la politique de l'implémentation de la politique : le micro-noyau ne fournit que des mécanismes permettant d'accéder au matériel, mais les décisions (quelle page mémoire envoyer sur le swap, à quel processus donner le CPU) sont prises en espace utilisateur.

Les avantages des micro-noyaux sont nombreux : plus grande tolérance de panne, plus grande flexibilité, gestion de la sécurité plus fine (il est par exemple possible d'autoriser les utilisateurs à écrire leurs propres systèmes de fichiers). Les désavantages sont aussi présents : les coûts de communication entre les différentes

couches peuvent être problématique, et la conception d'un tel système demande plus de précautions et de réflexion.

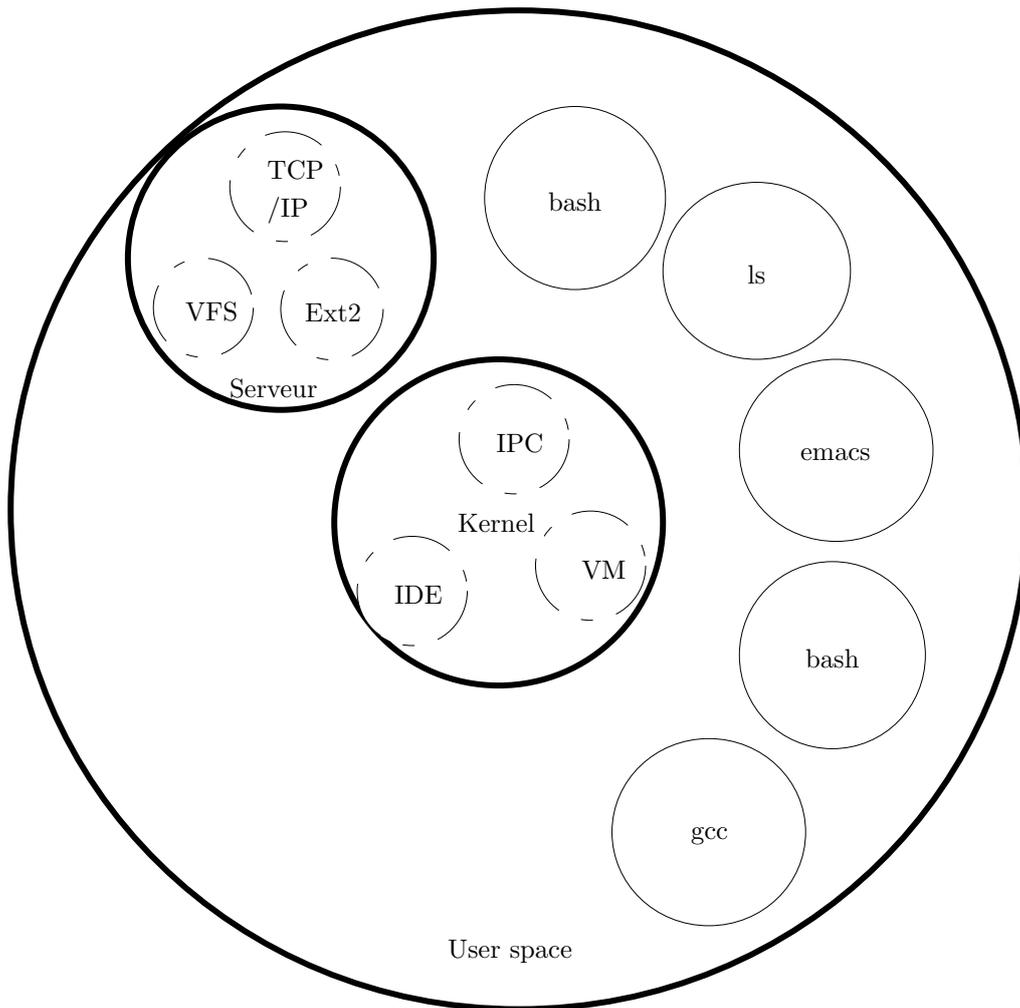


FIG. 1.4 – Micro-noyau à serveur unique

### Les autres types de systèmes

Il faut d'abord noter qu'il existe de nombreux hybrides entre les modèles présentés plus haut. Par exemple, MacOS X utilise un micro-noyau Mach au-dessus duquel se trouve un "serveur" FreeBSD en espace noyau, mais une gestion de l'USB presque totalement en espace utilisateur.

Les systèmes Unix eux, en général possèdent un noyau monolithique, mais externalisent la quasi-totalité de la gestion des aspects graphiques à une application en espace utilisateur, le serveur X. À l'opposé, Windows 2000 et ses successeurs revendiquent un micro-noyau, mais pourtant une partie de la couche graphique se trouve en espace noyau.

Il existe aussi d'autres types de systèmes, comme les exonoyaux et les systèmes à couches, mais leur étude sort du cadre de ce cours.

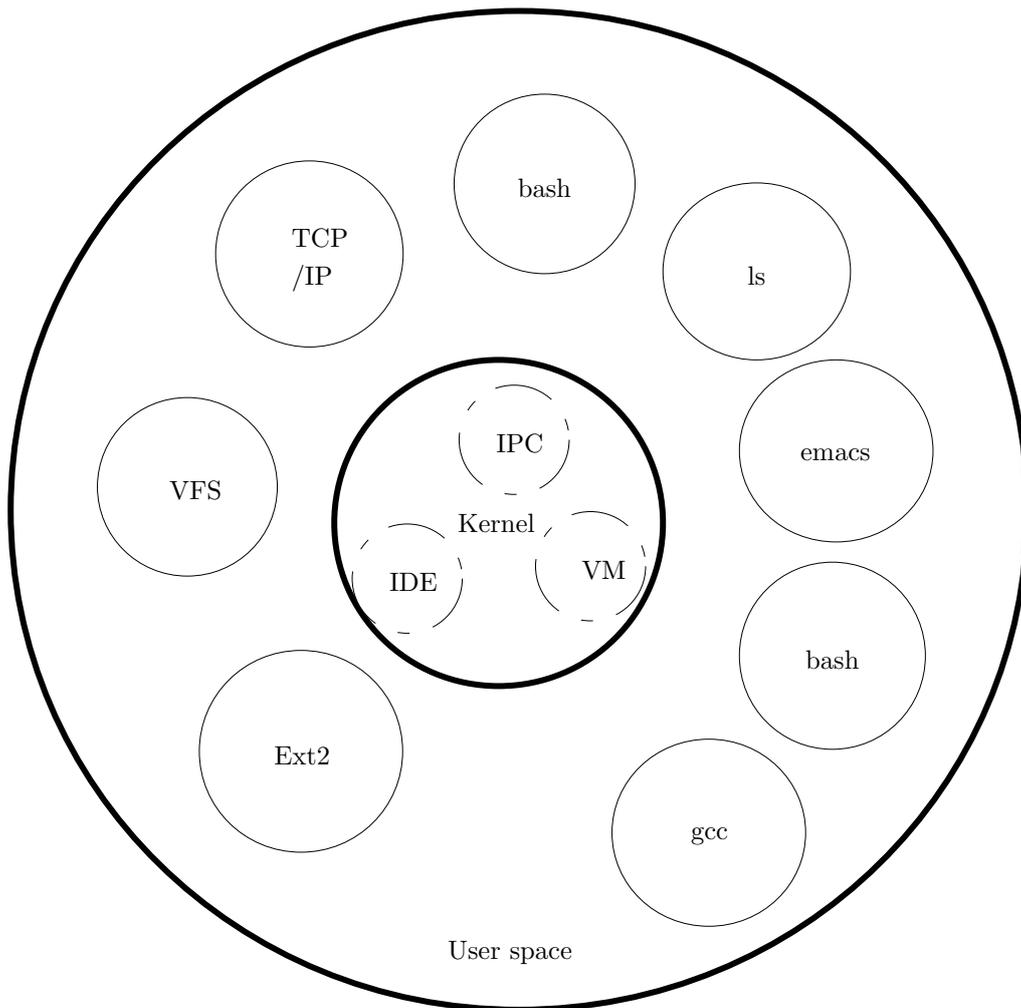


FIG. 1.5 – Micro-noyau à serveurs multiples

# Informations légales

Ce document est Copyright(c) Pilot Systems 2007. Il est disponible selon les termes de la GNU General Public License, version 3 ou supérieure.

La version PDF et le code source  $\text{\LaTeX}$  sont disponibles sur <http://insia.pilotsystems.net>.