Systèmes d'exploitation

Notes de cours

sergiu.ivanov@univ-evry.fr

2022/2023

Contents

1 Sergiu Ivanov **2**

2 Le cours **2**
2.1 Résumé . . . 2
2.2 Déroulé . . . 2
2.3 Évaluation . . . 2

3 CM$_1$: systèmes d'exploitation **2**
3.1 Systèmes d'exploitation : rôles et fonctions . . . 2
3.2 Organisation par couches . . . 3
3.3 Exemples . . . 3
3.4 Machine abstraite . . . 4
3.5 Interruptions . . . 4
3.6 Entrées/sorties synchrones et asynchrones . . . 4

4 CM$_2$: systèmes d'exploitation **4**
4.1 Entrées/sorties asynchrones . . . 4
4.2 Typologie des systèmes d'exploitation . . . 5
4.3 Historique des systèmes d'exploitation . . . 5

5 CM$_3$: systèmes de tâches et parallélisme maximal **5**
5.1 Gestion des processus . . . 5
5.2 Systèmes de tâches . . . 5
5.3 Détérminisme . . . 5
5.4 Parallélisme maximal . . . 5

6 CM$_4$: parallélisme maximal + gestion des processus **6**
6.1 Rappel : systèmes de tâches et parallélisme maximal . . . 6
6.2 Exemple d'application de l'algorithme de parallélisme maximal . . . 6
6.2.1 Vérification du déterminisme . . . 6
6.2.2 Construction du système de parallélisme maximal . . . 7
6.3 Gestion de processus . . . 7

7 CM$_5$: travail sur le projet **8**

8 CM$_6$: discussions diverses **8**
8.1 Organisation de la matière . . . 8
8.2 Virtualisation : machines virtuelles + conteneurs . . . 8
8.3 Processus de démarrage . . . 9
8.4 Conteneurs . . . 9
8.5 Runtime / langages de programmation . . . 9
8.6 Systèmes de fichiers . . . 10

8.7 Sujets pertinents mais non traités par manque de temps 10

1 Sergiu Ivanov

Maître de conférences au laboratoire IBISC. Domaines de recherche : biologie théorique, informatique théorique, auto-assemblage en ADN.

Ma page web: `https://www.ibisc.univ-evry.fr/~sivanov/`

2 Le cours

2.1 Résumé

6 CM + 6 TD d'introduction aux sujets suivants :

- systèmes d'exploitation, surtout entrées/sorties,
- programmation parallèle et concurrente.

2.2 Déroulé

Introduction + interruptions	CM_1		
		TD_1	Consigne TD_1
Suite + entrées/sorties	CM_2		
		TD_2	Consigne TD_2
Systèmes de tâches	CM_3		
		TD_3	Consigne TD_3
Parallélisme maximal	CM_4		
		TD_4	Consigne TD_3
Projet	*CM_5*		
		TD_5	*Projet*
À définir	CM_6		
		TD_6	**Soutenances**

2.3 Évaluation

1. Projet : rendu code + présentation
2. Examen final : QCM

3 CM_1 : systèmes d'exploitation

3.1 Systèmes d'exploitation : rôles et fonctions

Qu'est-ce qu'un système d'exploitation ?

Un système d'exploitation est un ensemble de logiciels qui effectuent des fonctions essentielles pour l'utilisation de l'ordinateur :

- gestion des tâches (activités),
- gestion du matériel (code brut & la machine),
- gestion des données (fichiers),
- gestion de la mémoire.

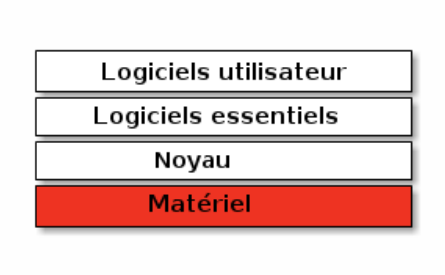
L'interface graphique (ou autre) ne fait pas partie d'un système d'exploitation, parce qu'elle n'est pas essentielle. Exemples : serveurs, certains ordinateurs embarqués.

De façon plus approfondie, une interface est l'ensemble de moyens dont un ordinateur est muni pour communiquer avec son environnement. Tout ordinateur doit avoir une interface pour être utile. Ces interfaces ne prennent pas nécessairement la forme de l'interface graphique ou en ligne de commande. Si on reprend l'exemple classique du serveur, son interface avec l'environnement est sa carte réseau.

La transformation du code est le travail du compilateur. Un compilateur est un simple logiciel et ne fait partie du système d'exploitation.

3.2 Organisation par couches

Les couches que l'on retrouve typiquement dans un ordinateur de bureau sont :



- les logiciels utilisateur,
- les logiciels essentiels,
- noyau,
- matériel.

Le noyau est la couche la plus proche du matériel et est une partie essentielle du système d'exploitation. Un noyau inclut typiquement: un gestionnaire de mémoire, des pilotes, un gestionnaire de fichiers.

Les logiciels essentiels sont typiquement perçu comme faisant partie du système d'exploitation, mais ne faisant pas partie du noyau. Ces logiciels peuvent être vus comme des clients de noyau qui communiquent avec lui à travers une interface logicielle - une API.

Le nombre de fonctionnalités que comprend le noyau varie entre différents systèmes d'exploitation. Les noyaux majeurs pour les systèmes d'exploitation de bureau populaires aujourd'hui sont monolithiques – ils assurent beaucoup de fonctions, mais modulables – il est habituellement possible de rajouter ou d'enlever des parties. Exemples : pilotes, modules.

3.3 Exemples

- Windows (11, Server)
- iOS
- MacOS
- (Kali, Debian) Linux
- Android
- Unix (FreeBSD, NetBSD, OpenBSD, etc.)
- MS-DOS
- Symbian OS
- etc.

Cette liste n'est ni exhaustive, ni homogène. Il existe donc beaucoup de systèmes d'exploitation qui n'y apparaissent pas. De l'autre côté, Windows, Linux et Unix sont en réalité des familles ou des lignées de systèmes d'exploitation, alors qu'Android, MacOS, MS-DOS sont des systèmes d'exploitation individuels.

Techniquement, Linux est un noyau semblable aux noyaux des systèmes Unix. Lorsque l'on dit « installer Linux », on imagine habituellement une distribution : un noyau accompagné d'un certain ensemble de logiciels essentiels et logiciels utilisateur. C'est ce qui distingue les distributions de Linux entre elles.

3.4 Machine abstraite

La machine abstraite est une image que le système d'exploitation présente aux logiciels dans le but de faire abstraction des détails de l'architecture matérielle, par exemple : le nombre de processeurs, la taille de la mémoire vive, etc.

Slides du chapitre 2 « Mécanismes d'exécution et de communication »[1] : 3–7.

3.5 Interruptions

L'interruption est le signal matériel envoyé par le contrôleur d'un périphérique (ordinateur auxiliaire dans la figure) au processeur central (ordinateur principal dans la figure) pour signaler un évènement externe. Ce signal entraîne la suspension de l'exécution en cours du processeur central afin de lancer une autre procédure : le traitant d'interruption.

Les interruptions sont très présentes dans les ordinateurs de bureau. Une interruption est déclenchées pour tout évènement extérieur au processeur central. Exemples : appuis des touches du clavier, mouvement de la souris, l'horloge. Par conséquent, le traitant d'interruption doit être une procédure très courte qui la plupart du temps ne traite pas directement l'évènement, mais programme son traitement à la prochaine disponibilité du processeur central.

Slides du chapitre 2 : 9, 13–15, 17–18, 12.

3.6 Entrées/sorties synchrones et asynchrones

Les interruptions sont un outil de bas niveau pour organiser les entrées/sorties asynchrones. Lorsque les entrées/sorties sont asynchrones, le système d'exploitation autorise le processeur central à exécuter d'autres tâches en parallèle avec l'exécution des opérations d'entrée/sortie. À l'opposé, les entrées/sorties synchrones impliquent une attente active de la fin de chaque opération d'entrée/sortie.

Slides du chapitre 2 : 29–30.

4 CM_2 : systèmes d'exploitation

4.1 Entrées/sorties asynchrones

Rappel du CM et des TD précédents : les interruptions sont au cœur des entrées/sorties asynchrones.

Slides du chapitre 2 : 12, 21, 23–26, 28–30.

Les ordinateurs modernes sont constitués de nombreux contrôleurs (processeurs spécialisés) qui fonctionnent en parallèle avec le processeur central et de façon asynchrone. Les interruptions sont un mécanisme de communication qui permet aux contrôleurs de signaler un évènement externe au processeur central. Les *tampons* sont un mécanisme de communication entre ces processeurs.

Un tampon est une zone mémoire dédiée à l'échange de données avec un périphérique en entrée ou en sortie. Les tampons sont généralement dédiés à l'un des sens de communication.

[1] Les numéros de slides correspondent aux numéros affichés en bas à droite des slides.

Slides du chapitre 2 : 32–34.

Dans le cas des périphériques très lents par rapport au débit du processeur central – comme est le cas de l'imprimante, un tampon supplémentaire peut être alloué sur le disque pour éviter d'immobiliser la mémoire vive. Cela donne lieu à une cascade de tampons sur le chemin des données du logiciel qui les produit jusqu'au périphérique.

Slide 35 du chapitre 2.

4.2 Typologie des systèmes d'exploitation

Les systèmes d'exploitation peuvent être classés par application et donc par les caractéristiques attendues, pour lesquelles ils sont optimisés.

Slides du chapitre 1 : 19–29.

4.3 Historique des systèmes d'exploitation

Slides du chapitre 1 : 33–40.

5 CM_3 : systèmes de tâches et parallélisme maximal

5.1 Gestion des processus

Les gestion des processus est un autre pan majeur des systèmes d'exploitation, aux côtés des entrées/sorties, des systèmes de fichiers, etc. Dans ce cours, nous éraflons à peine la surface de ces sujets, et nous ferons même pas le tour de tous les aspects.

Slides du chapitre 3 : 3, 6–7.

Lorsque l'on parle de création d'un processus à partir d'un programme, il s'agit du code dans le langage machine que l'on retrouve dans un fichier exécutable. Ce fichier contient d'une part le programme, mais aussi les constantes, ainsi qu'une description de la disposition de mémoire attendue. Nous ne parlons pas dans cette matière de compilation ni d'édition des liens qui permettent de produire ces exécutables à partir du code source dans un langage haut niveau habituel : C, Java, Python, etc.

5.2 Systèmes de tâches

Les systèmes de tâches sont un cadre formel pour parler précisément des systèmes parallèles et concurrents.

Slides du chapitre 3 : 15–23.

L'opération de composition parallèle est souvent fournie par des bibliothèques de programmation parallèle sous le nom de « parallel map ».

Slide 24 du chapitre 3.

5.3 Détérminisme

Slides du chapitre 4 : 12, 14, 16–18, 20–23.

5.4 Parallélisme maximal

Slides du chapitre 4 : 24–29.

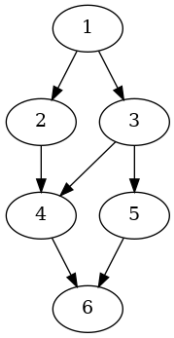
6 CM_4 : parallélisme maximal + gestion des processus

6.1 Rappel : systèmes de tâches et parallélisme maximal

Slides du chapitre 4 : 17–19, 21–22, 27–28.

6.2 Exemple d'application de l'algorithme de parallélisme maximal

Prenons le système de tâches affiché sur la slide 28 du chapitre 4 :



Voici les domaines de lecture et d'écriture de ses tâches :

T_i	L_i	E_i
1	M_1	M_4
2	M_3, M_4	M_1
3	M_3, M_4	M_5
4	M_4	M_2
5	M_5	M_5
6	M_1, M_2	M_4

6.2.1 Vérification du déterminisme

L'algorithme de parallélisme maximal débute nécessairement avec un système de tâches déterminé. Pour nous assurer que le système de tâches de départ est déterminé, nous prenons chaque paire de tâches qui *ne sont pas connectées par un chemin* et vérifions les conditions de Bernstein sur leurs domaines de lecture et d'écriture :

- T_2 et T_3 : OK,
- T_2 et T_5 : OK,
- T_4 et T_5 : OK.

Le système de départ est donc déterminé.

Il est des fois pratique d'étudier le tableau qui décrit les domaines de lecture et d'écriture auxquels appartient chacune des variables (cellules mémoire) du système de tâches, comme indiqué sur la slide 28. Par exemple, on peut y voir que la variable M_1 appartient aux domaines de lecture des tâches 1 et 6, et au domaine d'écriture de la tâche 2. Cela nous permet de déduire plus facilement que les paires de tâches 1 et 2, ainsi 6 et 2 sont interférentes.

6.2.2 Construction du système de parallélisme maximal

Construisons d'abord la matrice des chemins à partir de la matrice d'adjacence du graphe du système de départ :

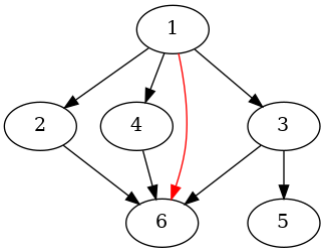
	1	2	3	4	5	6
1		×	×	×	×	×
2				×		×
3				×	×	×
4						×
5						×
6						

Cette matrice est formellement appelée *la clôture transitive de la matrice d'adjacence*.

Parcourons maintenant à l'aide de cette matrice chaque chemin dans le graphe de précédence du système de départ et vérifierons si la suppression de ce chemin est possible en appliquant les conditions de Bernstein sur les domaines de lecture et d'écriture des tâches connectées par ce chemin :

	1	2	3	4	5	6
1		×	×	×		×
2						×
3					×	×
4						×
5						
6						

Nous avons pu supprimer les chemins (les contraintes de précédence) suivants : $1 \to 5$, $2 \to 4$, $3 \to 4$, $5 \to 6$. Cela nous donne ce système de parallélisme maximal :



La flèche $1 \to 6$ dans ce graphe est redondante dans le sens où ces tâches ont déjà une contrainte de précédence imposée par le chemin $1 \to 4 \to 6$.

Une avancée que nous avons pu obtenir en construisant le système de parallélisme maximal est la réduction du nombre d'étages ou d'étapes de calcul nécessaires pour exécuter l'ensemble de tâches. Le système de départ imposait 4 étages d'exécution : $\{1\}$, $\{2, 3\}$, $\{4, 5\}$, $\{6\}$. Le système de parallélisme maximal n'en impose que 3 : $\{1\}$, $\{2, 3, 4\}$, $\{5, 6\}$. On peut donc obtenir un gain en terme de temps de calcul si on dispose d'une capacité de parallélisation suffisante.

Notez que le nombre total de pas de calcul ne change pas suite à une parallélisation plus forte. En revanche, si on dispose d'une capacité de parallélisation suffisante, nous pouvons exécuter plus de pas de calcul en parallèle et donc gagner du temps en sollicitant plus de ressources de calcul en parallèle.

6.3 Gestion de processus

Slides du chapitre 3 : 6–14.

Slides « Gestionnaire de processus. Introduction au document technique ».

Document « Gestionnaire de processus » : sections 1–4.

7 CM_5 : travail sur le projet

8 CM_6 : discussions diverses

8.1 Organisation de la matière

La date de soutenance des projets a été reportée au 14 avril, et la date de rendu du code de projet au 17 avril. Les soutenances le 14 avril consisterons en des exposés de 10 minutes par groupe, suivi par quelques questions. Aucun rapport écrit n'est demandé.

L'usage de ChatGPT dans la préparation des projets n'est pas interdit, notamment parce qu'une telle interdiction ne pourra pas être imposée. En revanche, vous êtes tenu·e·s à comprendre entièrement votre rendu. C'est une question de responsabilité que vous assumerez en tant que professionnel·le·s, en possession d'un diplôme universitaire. Les enseignants se réservent le droit de vous donner la note 0 au projet s'ils déterminent que votre soumission est issue directement de ChatGPT.

Le partiel se présentera sous la forme d'un QCM d'une durée de 1h ou 1h30. Les informations plus précises concernant sa durée seront diffusées plus tard.

8.2 Virtualisation : machines virtuelles + conteneurs

Une *machine virtuelle* (VM, virtual machine) = un logiciel qui émule un certain ensemble de matériels, et qui est habituellement utilisé pour faire tourner un système d'exploitation avec ses logiciels.

Le système qui fait tourner la machine virtuelle est habituellement appelé l'hôte. Le système qui est exécuté *dans* la machine virtuelle est habituellement appelé l'invité.

Le processus typique d'installation d'une VM commence par le démarrage depuis une image démarrable (bootable). Le terme « image » fait référence à un fichier qui contient l'image d'un disque entier, avec le système de fichier ISO 9660. Historiquement, ces images étaient distribués sur les CD, puis DVD, d'où l'usage du système ISO 9660, spécifiquement conçu pour les supports optiques. Une telle image démarrable (bootable) contient typiquement un système d'exploitation, des logiciels d'installation, des logiciels à installer, des pilotes, etc.

Les étapes typiques d'installation d'une VM :

1. Démarrer avec une image ISO bootable.
2. Installer le système d'exploitation sur le disque dur virtuel de la machine virtuelle.
3. Redémarrer la VM avec le nouveau disque dur virtuel.

Utilisations possibles d'une machine virtuelle :

- tester différentes configurations de réseau,
- tester différents systèmes d'exploitation,
- tester différentes architectures,
- isoler et étudier des pourriels (logiciels malveillants, malware),
- tester les performances d'un logiciel.

Avantage principal d'une machine virtuelle : émulation de tous les niveaux du matériel. Cela permet par exemple d'émuler une architecture RISC (e.g. ARM) sur un processeur CISC (e.g. Intel, AMD).

Problème : le coût de l'émulation de toute la pile matérielle est important, ça va beaucoup moins vite.

Pour pallier ce problème, la plupart d'outils de gestion de machines virtuels permettent une réutilisation directe de matériel dans les VM, afin de réduire la surcouche d'émulation, e.g. VirtualBox Guest Additions. Cela peut créer en revanche des failles d'isolation.

8.3 Processus de démarrage

Le démarrage d'un ordinateur – la séquence d'évènements qui se déroulent entre le moment où l'alimentation est branchée et le moment où l'interface d'accueil du système d'exploitation nous est affichée – est un processus long, intéressant, et complexe. Voici un survol des étapes :

1. Le processeur central fait l'inventaire des matériels connectés, d'où l'allumage successif des LED, par exemple.
2. Le processeur central exécute le BIOS/UEFI pour trouver le support (disque dur, clef USB, réseau, etc.) depuis lequel continuer le démarrage.
3. Le processeur exécute le démarreur (bootloader). Dans certains cas, on peut observer l'interface de choix de système d'exploitation, qui est affichée par le démarreur. Certains démarreurs comme GRUB, sont tellement complexes qu'ils possèdent un démarreur de démarreur.
4. Le démarreur démarre le noyau du système d'exploitation.
5. Le noyau fait l'inventaire des pilotes à charger pour le matériel disponible, lance la gestion des processus avec le mode non privilégie du processeur, etc.
6. Le noyau lance les premiers processus utilisateur, dont l'interface.

8.4 Conteneurs

Un *conteneur* est un dispositif d'isolation avancée de processus. L'isolation est typiquement réalisée à 2 niveaux :

1. le système de fichiers : le conteneur a une vision limitée sur le système de fichiers hôte (e.g. chroot),
2. les processus : les processus du conteneur ne savent pas l'existence des processus à l'extérieur du conteneur (e.g. cgroups).

Docker est un exemple de logiciel pour la conception et exécution de conteneurs construit à base de chroot et cgroups. LXC en est un autre. Les FreeBDS jails ont été parmi les inspirations pour ces mécanismes d'isolation.

L'objectif des conteneurs est d'isoler les logiciels, tout en mutualisant le noyau du système d'exploitation hôte, afin de considérablement réduire le coût de l'isolation, notamment par rapport aux machines virtuelles.

Un exemple d'usage typique est la configuration d'une machine serveur qui fait tourner plusieurs services : serveur web, serveur E-mail, serveur Git, etc. Les conteneurs permettent de limiter la casse dans le cas où l'un de ces services est compromis, mais aussi de fluidifier la gestion des ressources qui leur sont allouées.

La mutualisation du noyau hôte impose le désavantage central : il n'est pas possible d'utiliser les conteneurs d'un autre système d'exploitation ou pour une autre architecture matérielle. Pour cela, l'usage d'une machine virtuelle s'impose. Il est possible néanmoins d'avoir des conteneurs d'une distribution de Linux différente, car toutes les distributions de Linux peuvent utiliser le même noyau.

8.5 Runtime / langages de programmation

Les programmes dans les langages de programmation de haut niveau (Python, Java, Racket, OCaml, etc.) tournent dans un environnement d'exécution, connu sous le nom de runtime en

anglais. L'environnement d'exécution est un environnement de virtualisation. Cela est particulièrement visible avec Java, qui est traduit vers un code binaire (bytecode) exécuté dans l'environnement appelée JVM : la Java Virtual Machine. La majorité des langages de haut niveau possèdent des environnements d'exécution offrant des services similaires : ramasse-miette (garbage collector), fils d'exécution (threads) légers, etc.

La richesse d'un runtime est l'un des critères qui distinguent un langage de haut niveau d'un langage de bas niveau. Le langage C par exemple n'as pas ou très peu d'environnement d'exécution spécifique.

8.6 Systèmes de fichiers

Le terme *système de fichiers* désigne

1. un format de stockage d'informations sur le disque,
2. une suite de logiciels pour manipuler ce stockage.

Imaginons que nous ayons 2 photos et 1 vidéo sur une clef USB. Quels sont les octets réellement stockés sur la clef ?

Bien qu'on pourrait imaginer stocker les trois données de façon contiguë sur la clef, cela ne répondrait pas à plusieurs demandes :

- comment savoir l'adresse à laquelle trouver la vidéo ?
- comment supprimer l'un des 3 fichiers et garder une trace de la suppression ?
- comment rajouter une nouvelle donnée, qui peut être de la taille plus petite ou plus grande par rapport à la taille de l'espace libéré à la suppression ?

Un système de fichiers répond à ces demandes en stockant en début du disque plusieurs métadonnées : l'arborescence, les noms des dossiers et des fichiers, les permissions, etc. Ces informations permettent d'avoir une vue structurée sur la partie principale du disque, qui contient les données elles-mêmes.

Lorsqu'un fichier est créé, le système de fichiers cherchera habituellement à le découper en quelques morceaux correspondant à la taille des régions libres (libérées lors des suppressions précédentes). Cela doit être fait sans engendrer trop de fragmentation – le phénomène où la plupart de fichiers sont stockés sous forme de trop de blocs non contigus.

La grande variété des systèmes de fichiers est engendrée par les usages différents et les optimisations qui s'en suivent. Typiquement, le système ISO 9660 utilisé sur les supports optiques, mais aussi pour stocker des informations qui varient peu dans le temps, est optimisé pour des lectures rapides et un stockage efficace en espace.

Le systèmes de fichiers intègrent également des fonctions de résilience : la récupération au moins partielle après des failles – coupure de l'électricité, crash du noyau, etc. Une solution souvent employée est la journalisation.

8.7 Sujets pertinents mais non traités par manque de temps

- licences libres,
- gestionnaires de paquets et DevOps,
- gestionnaires de fenêtres,
- *etc.*