Système d'exploitation

Wassim SAIDANE

Mise à jour 19/01/2021

Table des matières

1 Présentation des systèmes d'exploitation **1**
1.1 Un ordinateur 1
1.2 Un OS 1

2 Principe des appels systèmes et interface POSIX **5**

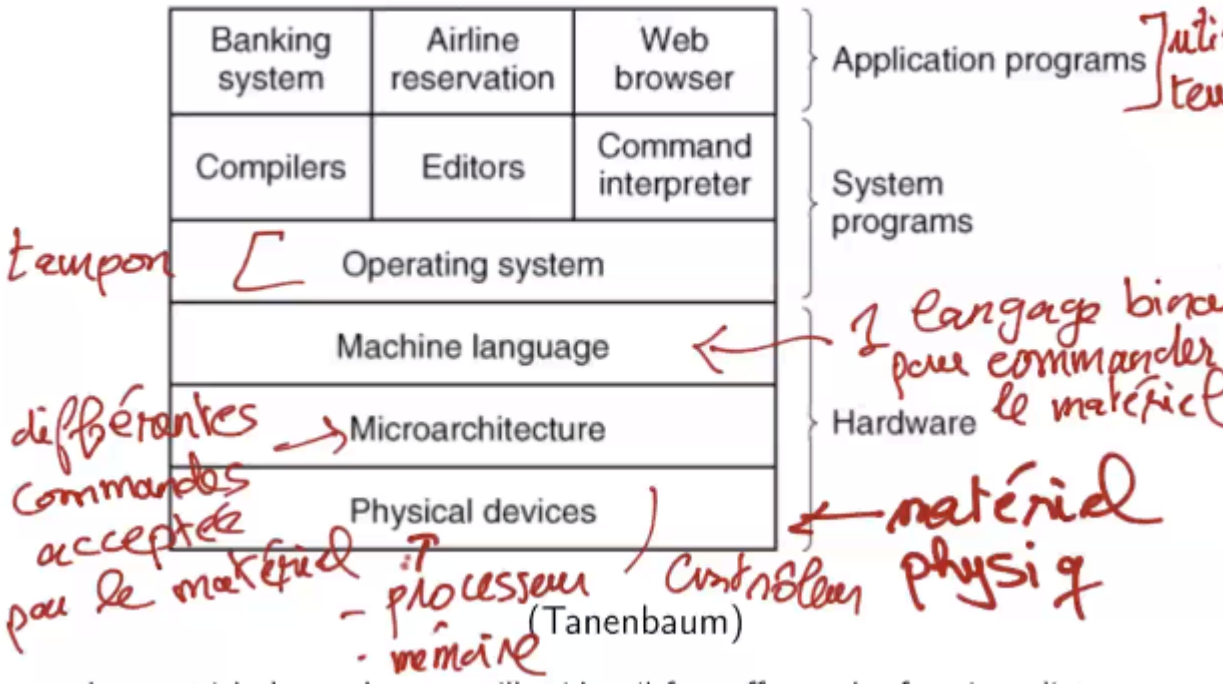
3 Gestion de la mémoire **11**

4 Gestion des processus **11**

5 Gestion des fichiers **11**

1 Présentation des systèmes d'exploitation

1.1 Un ordinateur



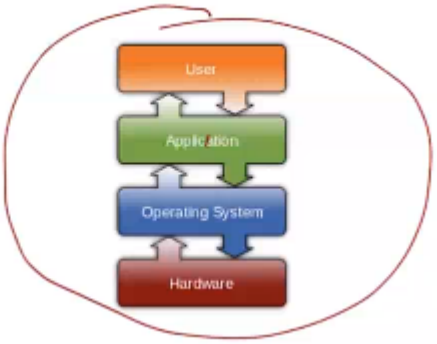
Le matériel n'est qu'une coquille vide : il faut affecter les fonctionnalités

1.2 Un OS

Le logiciel qui cache la complexité de l'architecture matérielle et la rend facilement exploitable.
Du point du vue utilisateur on ne veut pas savoir toutes les informations sur le matériel utilisé.
Un OS donne aux développeurs une base (/bin) et rend les choses simples, uniformes et cohérente (/media)
OS = machine virtuelle plus facile à programmer et gestionnaire de ressources.
Cette machine virteulle contrôle les accés aux ressources, uniformise les accès et si possible simplifie l'accés.
Le système d'exploitation est une machine virtuelle car on n'a pas à se soucier de comment invoquer les commandes des périphiques.
Il est également un un gestionnaire de ressources : Gestion d'accès aux différents péripériques.

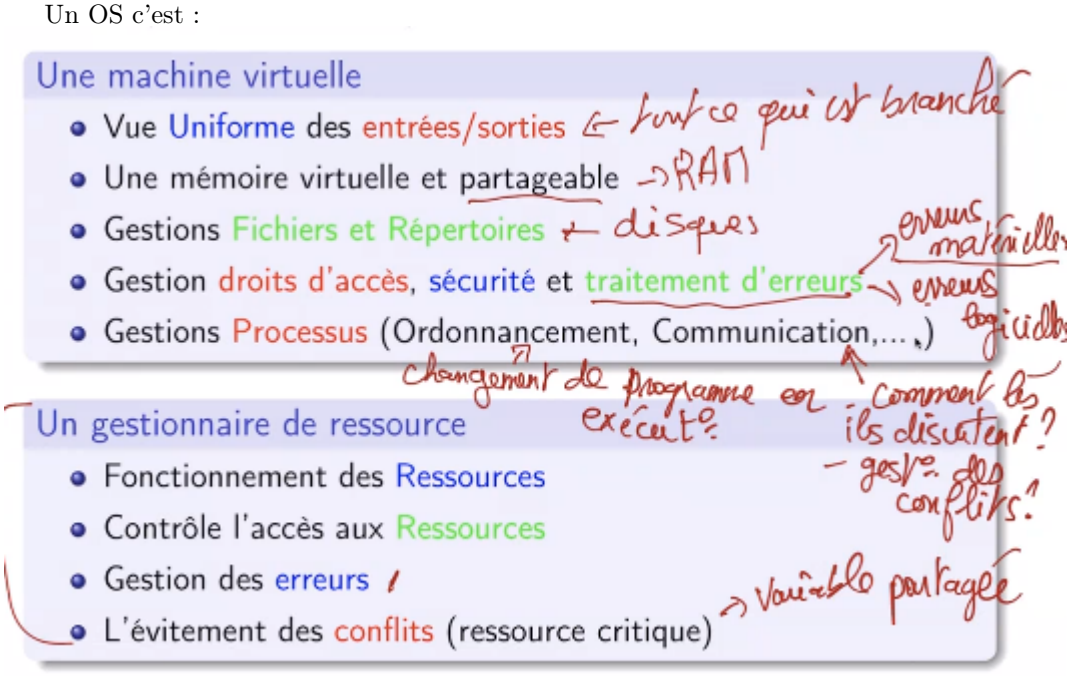
Le système d'exploitation se charge

- Des fichiers ← Système de gestion des fichiers
- Des processus ← lancer, quitter les programmes
- De la mémoire ← allocation dynamique / gestion de la mémoire d'1 processus, ...
- Des E/S ← disques, clavier, souris, ...
- Des utilisateurs ← différents accès possibles



Un OS ce n'est pas :
- L'interpète de commandes
- L'interface graphique
- Les utilitaires
- Le BIOS

Un OS c'est :



Différents types d'OS pour différentes fonctionnalités :

- Mono utilisateur : votre téléphone par exemple
- Temps réel (Nucléaire, Chimie) : réactif
- Général (Linux, Windows, Android, Mac OS,...) Multi-tâches et Multi-utilisateurs

Les programmes en exécution sont encapsulés dans des processus :
- Gestion multi-programmation
- Gestion des communications entre programmes
- Gestion des attentes de réponse des programmes
- Droits associés aux restrictions fixés aux utilisateurs

Les programmes en exécution sont encapsulés dans des processus
- Gestion multi-programmation
- Gestion des communications entre programmes
- Gestion des attentes de réponse des programmes
- Droits associés aux restrictions fixés aux utilisateurs

(Systèmes de) Fichiers
- Les données stockées dans des objetsappelés fichiers et on s'abstrait des disque.
- Gestion des droits d'accès aux fichiers.

Mémoire
- Gestion de la mémoire d'exéctuin des programmes par espaces d'adressage.
- Mémoire transformée en mémoire virtuelle.

Un ensemble de fonctions systèmes avex des super-droits et une sémentique d'appel particulière appelée appels systèmes.
- Linterpréteur Shell est l'exemple type utilisant beaucoup les appels systèmes.

2 Principe des appels systèmes et interface POSIX

Attention : 1 seule instruction par temps CPU

1. Proposer des services pour accéder au matériel: fonctions systèmes
   - read, open, fork, dup, etc.
2. Traiter les erreurs matérielles des processus
   - division par zero, seg fault, etc.
3. Traiter les interruptions matérielles
   - erreur de lecture disque, ecran, souris, clavier, etc.
4. Entretien global: accés au processeur, allocation de mémoire, etc.

Problématiques

1. Protection matérielle.
2. Certaines instructions sont réservées (pour la protection par exemple) ou accés à certaines parties de la mémoire.

Mémoire virtuelle et 2 modes d'exécution: utilisateur et noyau

Mémoire virtuelle

1. Les adresses mémoire des programmes ne peuvent référencer les adresses physiques.
2. Les processus ont des espaces d'adressage virtuel
3. Lors du chargement les adresses virtuelles sont traduites en adresses physiques (changement de contexte)
   - Un circuit Memory Management Unit fait la conversion à l'aide de registres
   - Une table de conversion pour chaque processus

Modes d'Exécution

Utilisateur

1. Processus peut accéder uniquement à son espace d'adressage et à un sous-ensemble du jeu d'instructions.
   $\Longrightarrow$ pas de corruption du système
2. L'accés à l'espace noyau est protégé et on y accède par une instruction protégée.

Noyau

1. Accès à tous les espaces : noyau et utilisateur
   - Code et données du SE accessible seulement en mode noyau: les segments mémoire sont inclus seulement lors du passage en mode mémoire.
2. Accés à toutes les instructions protégées (qui ne peuvent exécutées qu'en mode noyau)
   - Instructions de modification segments de mémoire: un processus ne peut pas modifier ses droits d'accés à la mémoire.
   - Accés aux périphériques: E/S, réseaux, allocation mémoire, etc.

Noyau?

Machine virtuelle

1. Vue uniforme des E/S
2. Gestion de la mémoire et des processus, réseau
3. Système de fichiers

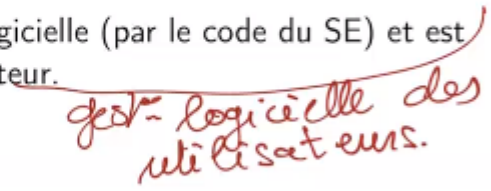
Gestionnaire de ressources

1. Fonctionnement des ressources (processeur, délais, ...)
2. Contrôle d'accés aux ressources (Allocation CPU, disque, mémoire, canal de communication réseau, ...)
3. Gestion des erreurs
4. Gestion des conflits

Modes d'exécution

Mode noyau ≠ mode root

1. Mode noyau = gestion par le matériel via des interruptions (matérielle et logicielle)
2. Mode root = gestion logicielle (par le code du SE) et est souvent en mode utilisateur.



Mode noyau par le matériel

1. non connaissance lors de la compilation des segments de mémoire où se trouvent les fonctions systèmes.
2. Raisons: maintenabilité et portabillité du SE

Standard POSIX: Portable Operating System Interface

1. Est une interface de programmation système.
   - Un ensemble de fonctions disponibles sur tous les SE *IX et pratiquement implémentées par tous.
   - Un ensemble de types : time_t, size_t, dev_t, ...
2. Beaucoup de fonctions libc sont des wrappers: font juste appel à la fonction système (ex: time, E/S, etc.)
   - Stocker les arguments dans les bons registres
   - Invoquer l'appel système
   - Interpréter la valeur de retour et si possible positionner la variable `errno`.

Principe Exécution Appels Systèmes (POSIX)

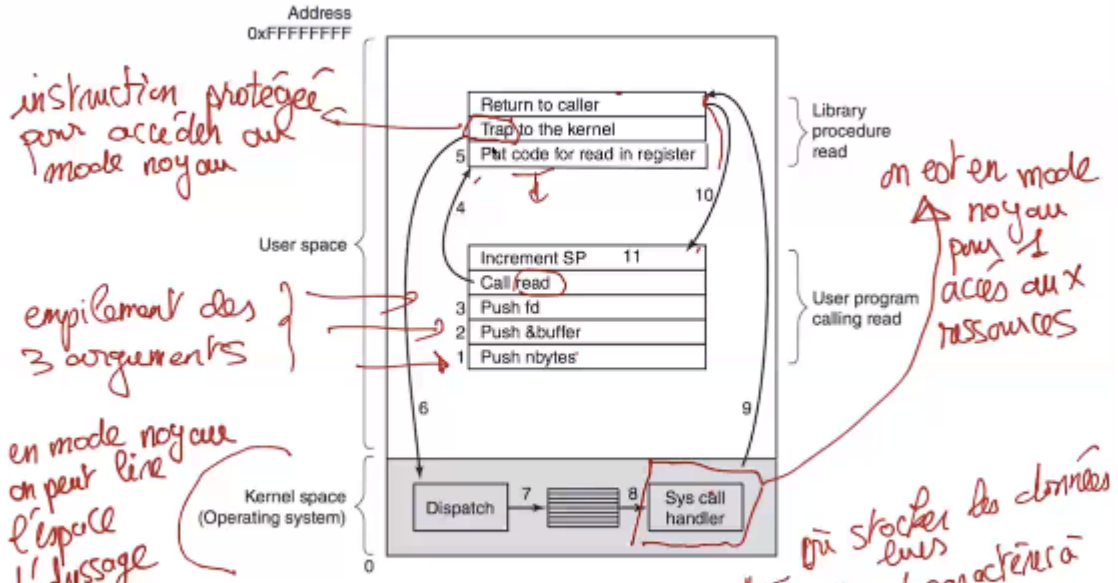


Figure 1-17. The 11 steps in making the system call read(fd, buffer, nbytes).

(Tanenbaum)

Principe Exécution Appels Systèmes (POSIX)

1. Lors de l'initialisation on installe les codes des appels systèmes dans une table Interrupt handler et à chaque fonction système on associe un numéro interrupt
2. Dans le code de l'appel système on a une instruction de passage en mode noyau (sous Linux: int) qui prend en arguments le numéro de la fonction système et les différents arguments de la fonction.
3. Depuis le mode noyau
    1. On appelle le gestionnaire d'exception trap handler: sauvegarde du contexte et transfert des données vers espace noyau.
    2. Ce dernier à son tour appelle la vraie fonction système (indexée par son numéro)
    3. Après calcul, transmission valeur de retour au trap
    4. transmission de la valeur de retour et des données et retour en mode utilisateur (encore instruction protégée) après restauration du contexte

Types d'Appels Systèmes

Appel bloquant. Le processus appelant ne pourra continuer son travail que lorsque l'appel système a terminé (lorsque les données demandées sont prêtes par exemple). Ex: appels système: open, read, write

Appel non bloquant. On fixe un délai δ. La main est redonnée automatiquement au processus appelant si au bout de δ temps l'appel système n'a pas terminé. Ex: read, write. Il existe des fonctions pour passer d'un mode bloquant à un mode non bloquant ou inversement.

Attente active. Le processus simule lui-même un mode bloquant sur un appel non bloquant. Ex: while (1) { r= read (...); if (r≥ 0) break;}

Gestion des erreurs

1. Une variable globale `errno` dans errno.h qui permet de transmettre les erreurs des fonctions systèmes aux codes utilisateurs
2. Un appel système qui réussit et alors le retour de la wrapper est un entier ≥ 0
3. Un appel système qui échoue et alors le retour de la wrapper est un entier < 0 et un positionnement de la variable `errno` : numéro de l'erreur

   Les fonctions `strerror(int)` et `perror(string)` pour avoir/afficher le texte associé à l'erreur : perror("ouverture") affiche

   "ouverture:"+message associé à `errno`

3 Gestion de la mémoire

4 Gestion des processus

5 Gestion des fichiers