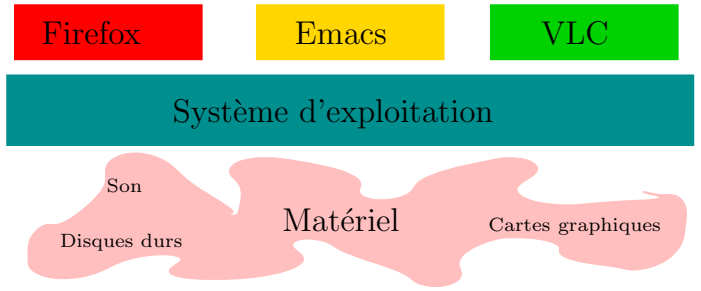
Premier chapitre

Introduction

Qu'est ce qu'un système d'exploitation

Logiciel entre les applications et le matériel

- Offre une interface unifiée aux applications
- Gère (et protège) les ressources

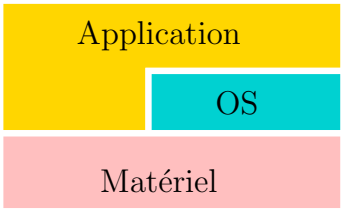


Évolution des systèmes d'exploitation : Étape 0

À l'origine, pas de système d'exploitation

Une machine, un utilisateur, un logiciel

- Tout au plus une bibliothèque de services standards
  Exemples : premières machines, systèmes embarqués (voitures, ascenseurs)
- Complexité conceptuelle réduite . . .
- . . . mais complexité technique accrue : on refait tout à chaque fois (⇒ cher)



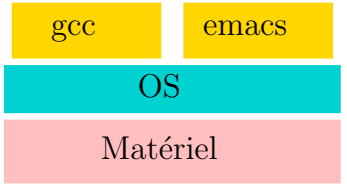
- Surtout, cela pose un problème d'efficacité :
  Si le programme est bloqué (disque, réseau, humain), la machine est inutilisée

Évolution des systèmes d'exploitation : Étape 1

Maximiser le temps d'utilisation du matériel

Une machine, plusieurs logiciels

- Problème fondamental de la précédente solution :
  Si le programme est bloqué (disque, réseau, humain), la machine est inutilisée
- Solution : **plusieurs processus**. Si l'un est bloqué, on exécute les autres.

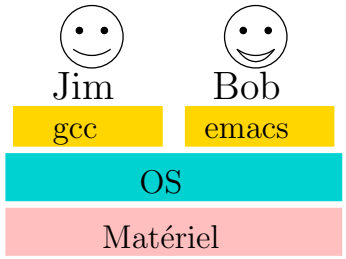


- Mais que se passe-t-il si un processus :
  - Fait une boucle infinie
  - Écrit dans la mémoire des autres processus
- Le système doit donc fournir une certaine protection :
  ⇒ interposition entre les applications et le matériel

Évolution des systèmes d'exploitation : Étape 2

Une machine, plusieurs logiciels, plusieurs utilisateurs

- La solution précédente est chère (en 1970) : il faut une machine par utilisateur.
- Solution : **partage de la machine entre utilisateurs**
  (les utilisateurs tapent moins vite que l'ordinateur ne compile)

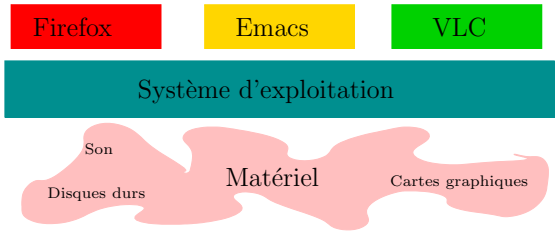


- Mais que se passe-t-il si les utilisateurs sont :
  - Trop gourmands ou trop nombreux
  - Mal intentionnés
- Le système doit protéger et gérer les ressources

Rôle de l'OS : intermédiaire matériel ↔ applications

Deux fonctions complémentaires

- Adaptation d'interface : offre une interface plus pratique que le matériel
  - Dissimule les détails de mise en œuvre (abstraction)
  - Dissimule les limitations physiques (taille mémoire) et le partage des ressources
- Gestion des ressources (mémoire, processeur, disque, réseau, affichage, etc.)
  - Alloue les ressources aux applications qui le demandent
  - Partage les ressources entre les applications
  - Protège les applications les unes des autres ; empêche l'usage abusif de ressources



Fonctions du système d'exploitation

	Organe physique	Entité logique
Gestion de l'activité		
Déroulement de l'exécution (processus)	Processeur	Processus
Événements matériels		
Gestion de l'information		
Accès dynamique (exécution, mémoire)	Mémoire principale	Mémoire virtuelle
Conservation permanente	Disque	Fichier
Partage		
Gestion des communications		
Interface avec utilisateur	Écran clavier, souris	Fenêtre
Impression et périphériques	Imprimante	
Réseau	Réseau	
Protection		
de l'information	Tous	Toutes
des ressources		

Premier chapitre

Introduction

UNIX, POSIX et les autres

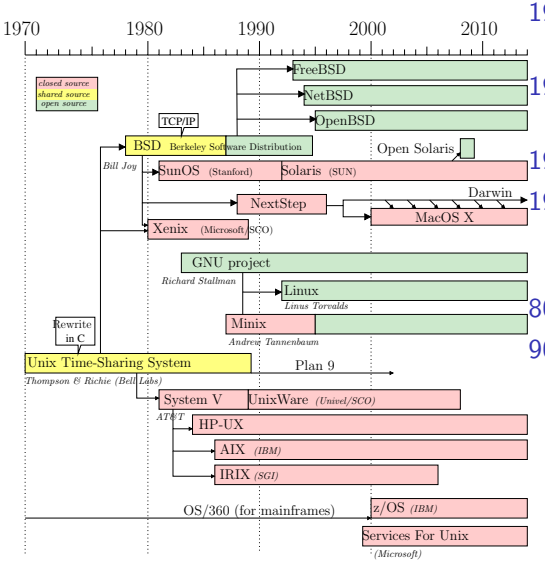
Qu'est ce qu'UNIX ? Pourquoi étudier UNIX ?

- Famille de systèmes d'exploitation, nombreuses implémentations
- Impact historique primordial, souvent copié
- Concepts de base simple, interface *historique* simple
- Des versions sont *libres*, tradition de code ouvert

Et pourquoi pas Windows ?

- En temps limité, il faut bien choisir ; je connais mieux les systèmes UNIX
- Même s'il y a de grosses différences, les concepts sont comparables
- D'autres cours à TELECOM Nancy utilisent Windows
  occasion d'étudier ce système ; le cours de RS permet d'aller plus vite
- Système plus *transparent* que Windows : plus facile de comprendre ce qui se passe sous le capot

Historique



1965 MULTICS : projet de système ambitieux (Bell Labs)

1969 Bell Labs se retire de MULTICS, Début de UNICS

1970 Unix projet Bell Labs officiel

1973 Réécriture en C
Distribution source aux universités
Commercialisation par AT&T

80-90 Unix War : BSD vs. System V

90-10 Effort de normalisation :

- Posix1(88) processus, signaux, tubes
- Posix1.b(93) semaphores, shared memory
- Posix1.c (95) threads
- Posix2 (92) interpreteur
- Posix :2001, Posix :2004, Posix :2008 Mises à jour

Premier chapitre

Introduction

Notion d'interface

Un service est caractérisé par son interface

- L'interface est l'ensemble des fonctions accessibles aux utilisateurs du service
- Chaque fonction est définie par :
  - son format : la description de son mode d'utilisation (syntaxe)
  - sa spécification : la description de son effet (sémantique)
- Ces descriptions doivent être :
  précises, complètes (y compris les cas d'erreur), non ambigües.

Principe de base : séparation entre interface et mode de réalisation

Avantages :

- Facilite la portabilité
  - Transport d'une application utilisant le service sur une autre réalisation
  - Passage d'un utilisateur sur une autre réalisation du système
- Les réalisations sont interchangeables facilement

Dans POSIX

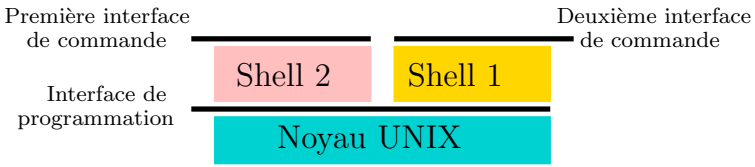
- C'est l'interface qui est normalisée, pas l'implémentation

Interfaces d'un système d'exploitation

En général, deux interfaces

- Interface de programmation (**A**pplication **P**rogramming **I**nterface)
  - Utilisable à partir des programmes s'exécutant sous le système
  - Composée d'un ensemble d'appels systèmes (procédures spéciales)
- Interface de commande ou interface de l'utilisateur
  - Utilisable par un usager humain, sous forme textuelle ou graphique
  - Composée d'un ensemble de commandes
    - Textuelles (exemple en UNIX : `rm *.o`)
    - Graphiques (exemple : déplacer l'icone d'un fichier vers la corbeille)

Exemple sous UNIX :



Exemple d'usage des interfaces d'UNIX

- Interface de programmation
  Ci-contre : programme copiant un fichier dans un autre
  (`read` et `write` : appels système)

```
while (nb_lus = read(fich1, buf, BUFFSIZE )) {
  if ((nb_lus == -1) && (errno != EINTR)) {
    break; /* erreur */
  } else if (nb_lus > 0) {
    pos_buff = buf;
    a_ecrire = nb_lus;
    while (nb_ecrits =
            write(fich2,pos_buff,a_ecrire)) {
      if ((nb_ecrits == -1) && (errno != EINTR))
        break;  /* erreur */
      else if (a_ecrire == 0)
        break;  /* fini */
      else if (nb_ecrits > 0) {
        pos_buff += nb_ecrits;
        a_ecrire -= nb_ecrits;
} } } }
```

- Interface de commande
  Commande shell :
  `$ cp fich1 fich2`
  recopie fich1 dans fich2

- Documentation
  `man 1 <nom>` : documentation des commandes (par défaut)
  `man 2 <nom>` : documentation des appels système
  `man 3 <nom>` : documentation des fonctions
  d'autres sections, mais plus rares. Voir `man 7 man` ;)

Découpage en couches du système

Niveau utilisateur

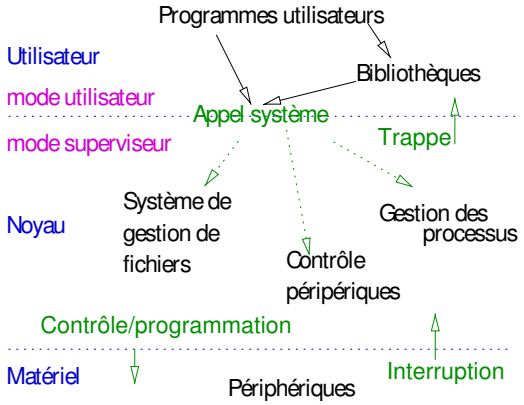
- Applications utilisateur
- Logiciels de base
- Bibliothèque système
- Appels de fonction

Niveau noyau

- Gestion des processus
- Système de fichier
- Gestion de la mémoire

Niveau matériel

- Firmware, contrôleur, SOC



OS = bibliothèque système + noyau

(d'où le nom GNU/Linux)

- Ce cours : bibliothèque système
- RSA : le noyau

Premier chapitre

Introduction

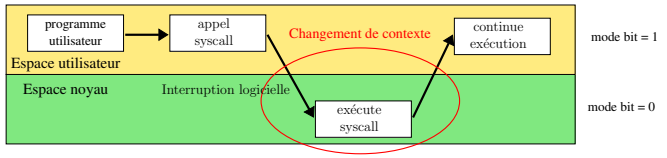
Protection des ressources (rapide introduction – cf. RSA)

Méthodes d'isolation des programmes (et utilisateurs) dangereux

- Préemption : ne donner aux applications que ce qu'on peut leur reprendre
- Interposition : pas d'accès direct, vérification de la validité de chaque accès
- Mode privilégié : certaines instructions machines réservées à l'OS

Exemples de ressources protégées

- Processeur : préemption
  - Interruptions (matérielles) à intervalles réguliers → contrôle à l'OS
  - (l'OS choisit le programme devant s'exécuter ensuite)
- Mémoire : interposition (validité de tout accès mémoire vérifiée au préalable)
- Exécution : mode privilégié (espace noyau) ou normal (espace utilisateur)
  - Le CPU bloque certaines instructions assembleur (E/S) en mode utilisateur



Limites de la protection

Les systèmes réels ont des failles

Les systèmes ne protègent pas de tout

- `while true ; do mkdir toto; cd toto; done` (en shell)
- `:(){ :|: & };:`
- `while(1) { fork(); }` (en C)
- `while(1) { char *a=malloc(512); *a='1'; }` (en C)

Réponse classique de l'OS : gel (voire pire)
On suppose que les utilisateurs ne sont pas mal intentionnés (erreur ?)

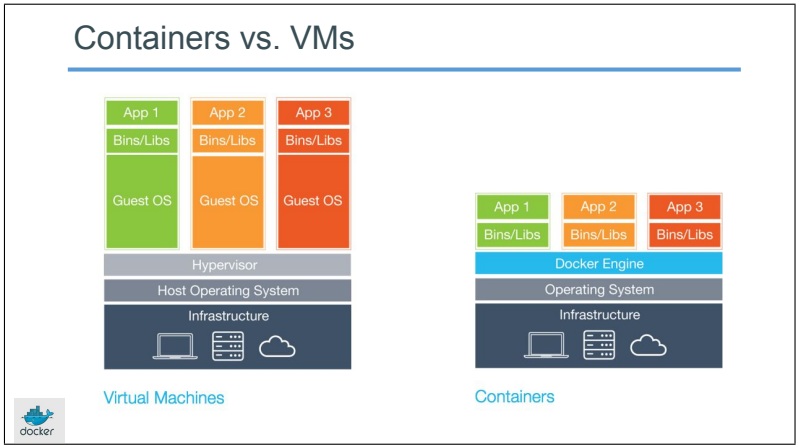
> *Unix was not designed to stop people from doing stupid things, because that would also stop them from doing clever things.*
> – Doug Gwyn

Deux types de solutions

Technique : mise en place de quotas

Sociale : "éduquer" les utilisateurs trop gourmands

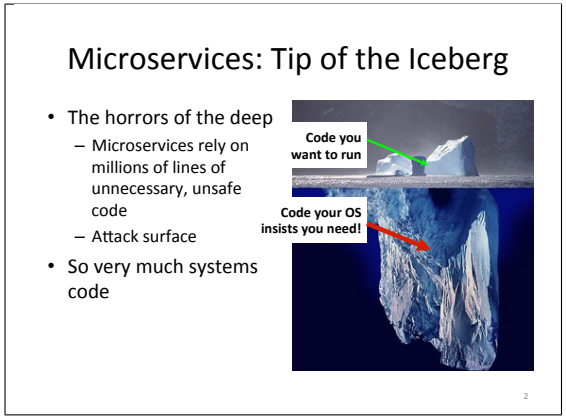
Évolution récente : virtualisation et conteneurs



- Virtualisation : un OS hôte (l'hyperviseur) permet d'héberger plusieurs systèmes *invités* en virtualisant le matériel ⇝ Xen, KVM, VMWare ESXi, etc.
- Conteneurs : l'OS hôte permet de distinguer plusieurs espaces utilisateurs isolés (mais qui partagent le même espace noyau) ⇝ Docker, LXC

Évolution récente : unikernels

- Noyau minimaliste, spécialisé, sans espace utilisateur (plus de changement de contexte), n'offrant qu'un seul service
- Retour à l'étape 0, mais le serveur physique est partagé par la virtualisation



(source)

Résumé du premier chapitre

- Le système d'exploitation
  - Un intermédiaire entre les applications et le matériel
- Rôles et fonctions du système d'exploitation
  - Offrir une interface du matériel unifiée et plus adaptée
  - Assurer la gestion et la protection des ressources
- Les différentes interfaces d'un système d'exploitation
  - Interface de programmation (API)
  - Interface de commande (shell, environnement graphique)
- Techniques classiques de protection des ressources
  - Préemption
  - Interposition
  - Mode d'exécution (protégé ou non)