# BUT Informatiques 2ème année
# Programmation Système
# R3.05
## Introduction

C. Raïevsky

IUT Valence
Université Grenoble Alpes

Département Informatique



## Objectifs de l'enseignement

Connaître le rôle d'un système d'exploitation *(Operating System, OS)*

- vis-à-vis des programmes,
- des fichiers,
- du réseau,
- du matériel.

Avoir une compréhension des mécanismes sous-jacents :

- lancement et exécution d'un programme ;
- permettant l'exécution parallèle de programmes ;
- permettant les accès mémoire.

Placer les multiples concepts reliés aux OS dans un cadre cohérent

Les informations de type "référence" sont accessibles en ligne.





## Autres objectifs de l'enseignement

Vous donner une meilleure compréhension que ChatGPT :

- des fonctions
- du rôle
- du fonctionnement

d'un système d'exploitation

Ne pas lire les 970 pages du livre d'Andrew Tanenbaum

Ne pas explorer les 20 millions de SLOC du noyau Linux





## Structure générale du cours

Themes abordés

- Fonction/Rôle d'un Système d'Exploitation (OS)
- Exécution - Processus - Thread
- Gestion des accès mémoire
- Ordonnancement - Programmation concurrente - Partage des ressources
- Communication inter-processus - Entrées-Sorties - Sockets

# Organisation

## Volumes horaires

- 11 séances de 1h de cours
- 1 DS de 2h
- 11 séances de TP de 1h30
  - Jean-Philippe Amouroux, Thalès Avionics
  - Thomas Roux, Conduent

## Évaluation

- TP notés
- DS final





# Disclaimer

Ne s'applique pas à tous les systèmes

## Ne concerne pas :

- Les systèmes embarqués
- Les micro-contrôleurs



## Biais fort vers les systèmes génériques

- Aspect software, vu depuis un programme
- Linux





# Fonctions d'un Système d'Exploitation

Fonctions essentielles – Utiles pour les programmeurs

**Rôle de chef d'orchestre**

## Gestion de l'exécution des programmes

- Lancement
- Basculement entre processus – Ordonnancement
- Adressage mémoire
- Certains aspects de la sécurité

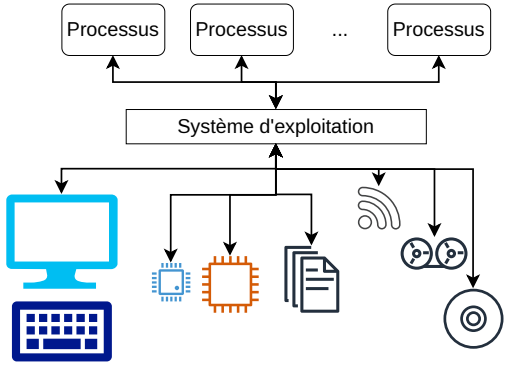




# Fonctions d'un Système d'Exploitation

Fonctions essentielles – Utiles pour les programmeurs

**Offrir au programmeur une interface uniforme et portable**




- Périphériques
- Système de fichier
- Réseau
- Horloges, Timers

# Fonctions secondaires

Visibles par l'utilisateur – Potentiellement déléguées à des programmes

Visibles par l'utilisateur – Potentiellement déléguées à des programmes

## Gestion des utilisateurs

- Sessions multiples
- Authentification

## Environnement graphique

- Gestionnaires de fenêtre (Gnome, KDE, etc.)
- Windows (en partie intégré au noyau)
- OSX

## Gestion de l'installation des applications

- Gestion de paquets Linux,
- App Store Apple,
- Play Store Android,
- etc.





# Gestion de l'exécution des processus

Chef d'orchestre

## Processus

"*A process is fundamentally a container that holds all the information needed to run a program.*" Tanenbaum

## L'OS est en charge de :

- Création
- Lancement
- Ordonnancement
- Destruction

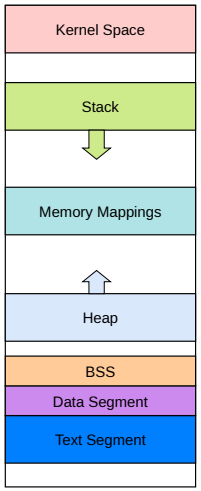




# Création d'un processus

## Étapes nécessaires à la création d'un processus :

- Création des structures de gestion internes,
- Allocation éventuelle de mémoire,
- Chargement du programme (code exécutable),
- Initialisation des différentes zones mémoire :
  - Constantes globales – Data Segment
  - Variables globales – BSS
  - Bibliothèques
  - Pile








# Lancement d'un processus

- Mise en attente du processus en cours (processus parent).
- Création du nouveau processus (processus enfant).
- Sauvegarde du contexte courant.
  - État du processus courant (structures noyau).
  - État du matériel (registres, etc.).
- Transfert du flot d'exécution au nouveau processus.

## Destruction d'un processus

- Mise à jour du statut du processus
- Libération des ressources
  - Structures de données noyau
  - Pages mémoire
  - Fichiers ouverts
- Suppression des références au processus dans le noyau
- Mise à jour des relations parent-enfant
- Envoie des éventuels signaux
- Transfert du flot d'exécution à un autre processus





## Ordonnancement des Processus

Problèmes liés à une exécution séquentielle

Système monotâche :

- Blocage du système complet à chaque entrée-sortie
- Discord et LOL mutuellement exclusifs → Inacceptable

⇒ Basculement entre les différents processus existants

À quel moment ? Quel processus choisir ?

- À intervalle régulier ? → sous utilisation du processeur
- Discipline a part entière → Ordonnancement
- Linux → Completely Fair Scheduler (CFS)





## Ordonnancement des Processus

Multi-tâche, Multi-programmation

Avantages

- Meilleurs utilisation des ressources
- Pas de blocage du système sur les I/O
- Parallèlisation des calculs intensifs
- etc. . .

Inconvénients

Conflits potentiels d'accès mémoire, disque et au matériel

Nécésité de :

- Protéger les espaces mémoire
- Gérer l'accès aux ressources ⇒ *Programmation Concurrente*





## Démo - Chef d'orchestre

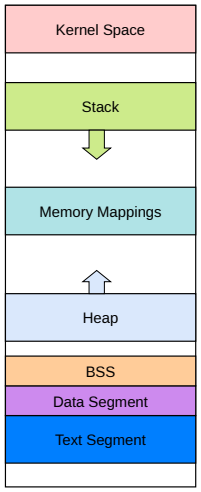
Exemple minimal de deux threads affichant des caractères en parallèle

*minimal.c*

## Mémoire d'un processus

Différentes zones mémoire :

- Constantes globales – Data Segment
- Variables globales – BSS
- Bibliothèques
- Pile








## Démo

Exemples d'adresses mémoire au démarrage

Exemple minimal d'affichage de l'adresse mémoire de deux variables
MemInit.c



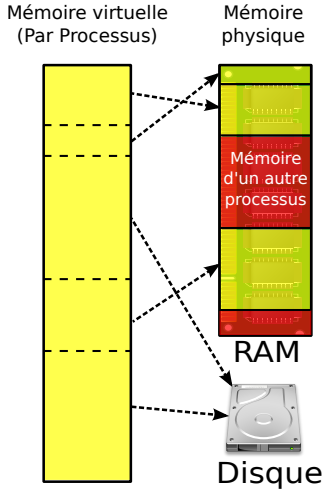


## Adressage mémoire

Mémoire Virtuelle

Chaque processus a un espace d'adressage **VIRTUEL**

- Évite les conflits d'adresses
- Facilite grandement la multi-programmation
- Permet de mettre en place des mécanismes de protection
- Nécessite un mécanisme de mise en correspondance

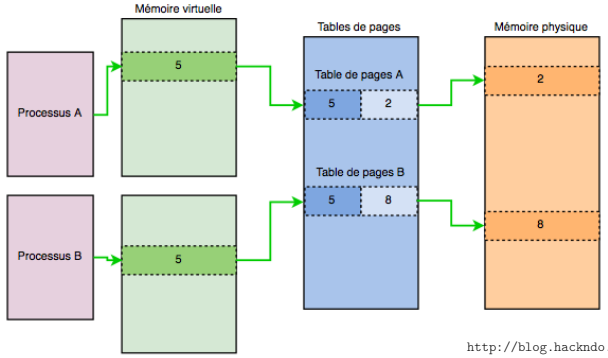







## Adressage Mémoire - Correspondance adresse virtuelle ↔ adresse physique

- Mémoire virtuelle et physique découpées en pages
- Chaque processus possède une **TABLE DE PAGE**
- qui contient la correspondance




http://blog.hackndo.com/gestion-de-la-memoire/

# Adressage Mémoire - Quelques chiffres

## Caractéristiques Générales

### Taille typique des pages : 4Ko

| Architecture | défaut | "Huge Pages" |
|---|---|---|
| x86_64 | 4Ko | 2Mo ou 1Go |
| ARM | 4Ko | 1Mo ou 16Mo |
| 32-bit x86 | 4Ko | 4Mo |
| Sparc | 8Ko | 64Ko, 4Mo, 256Mo, 2Go |

### Taille de l'espace d'adressage virtuel

- Systèmes 32bits : maximum 4Go
- Systèmes 64bits :
  - Maximum théorique : 18446744073709551616 octets
    $\rightsquigarrow$ 1 milliard de Go
  - Implémentation actuelle (x86_64 : 48bits) : 140737488355328 octets
    $\rightsquigarrow$ 256000 Go
  - Ça devrait suffire pour un moment...

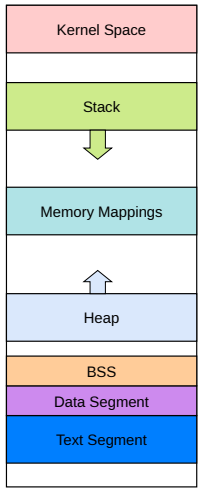




# Protection - Sécurité

### Accès restreint à :

- Text segment (ro)
- Data segment
- BSS segment
- Heap (Tas, free-store)
- Stack (Pile)
- Memory Mapping segment








# Protection - Sécurité

### Un processus ne peut accéder qu'à certaine parties de son espace d'adressage

- Text segment (ro)
- Data segment
- BSS segment
- Heap (Tas, free-store)
- Stack (Pile)
- Memory Mapping segment

### Tout accès en dehors de ces zones $\Rightarrow$ Segmentation Fault

Une exception : appel de fonction $\rightarrow$ agrandissement de la pile

### Cohérence assurée par les tables des pages

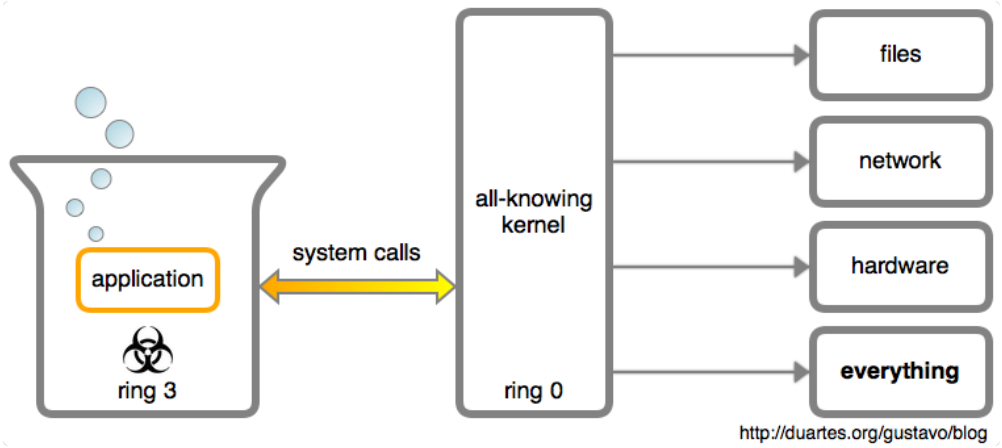
- Une page physique n'est associée qu'à un processus à la fois
- Exception : possibilité de partager explicitement des zones





# Abstraction du matériel – Fonction majeure de l'OS

**Offrir au programmeur une interface uniforme et portable du matériel**
Sans l'OS un programme n'a aucun moyen d'interagir avec l'extérieur.






Exception : systèmes embarqués

## Abstraction du matériel

### Types de matériels :

- Accès à des données locales (disque, mémoire, clavier,...)
- Accès réseau
- Horloges – Timers
- Contrôle spécifique

### Deux principaux types d'abstraction :

- Fichiers
- Adresses mémoire

⇒ Interface uniforme pour les programmes





## Classification en fonction du mode d'accès

En fonction du mode d'accès

### Accès à des données (fichiers) : en fonction du mode d'accès :

- Accès par blocs (disques, mémoire) → *Block Devices*
- Accès par flux (clavier, réseau) → *Character Devices*

### Memory-mapped Files

- Un fichier est mis en correspondance avec un espace mémoire
- Partage efficace de fichiers entre processus
- Très utilisé pour les bibliothèques partagés (libc, ...)
- Beaucoup plus efficace que les accès disque





## Memory-Mapped & Port-Mapped I/O

### Adresses Dédiées

- Communication via des adresses mémoires réservés
- Mémoire et/ou registres matériels mis en correspondance avec des adresses mémoire
- Transparent pour les programmes (hopefully)
- Toutes les instructions manipulant la mémoire sont disponibles

### Ports Dédiés

- Transfert de la valeur d'un registre CPU sur un port
- Très peu d'instructions disponibles

Memory-mapped I/O est préférable quand disponible





## À retenir

### Systèmes d'exploitation

- Logiciel permettant l'utilisation de la machine
- Abstraction du matériel pour les applications
- Chef d'orchestre des applications