OI201 - Systèmes d'Exploitation

Résumé Théorique

12 septembre 2024

Guilherme Nunes Trofino
2022-2024

Table des matières

1. Introduction

Repository Hello ! My name is Guilherme Nunes Trofino and this is my LaTeX notebook of OI201 - Systèmes d'Exploitation that can be found in my GitHub repository : https://github.com/tr0fin0/classes_ensta.

Disclaimer This notebook is made so it may help others in this subject and is not intend to be used to cheat on tests so use it by your on risk.

Suggestions If you may find something on this document that does not seam correct please reach me by e-mail : guitrofino@gmail.com.

1.1. Information Matier

Référence Dans cette matière le but sera de comprendre comment une Système d'Exploitation marche.

1.2. Histoire

Les premières ordinateurs faisaient du **Batch Processing** à cause de la simplicité des systèmes de l'époque.

Définition 1.1. Ordinateurs que n'exécutent qu'un seul programme à la fois font du **Batch Processing**. Dans ce cas le programme n'a pas besoin de partage les resources de l'ordinateur. Il pourrait acceder :

1. toute la mémoire ;
2. tout le processeur ;
3. tous les périphériques ;

Comme ça n'est plus le cas actuellement on considère souvent le suivant :

Phrase. un système d'exploitation n'est utile que lorsqu'il y a plusieurs programmes

2. Architecture

Souvent, pour comprendre comment l'ordinateur marche, c'est nécessaire de connaître sa structure physique nommé :

2.1. Architecture Matérielle

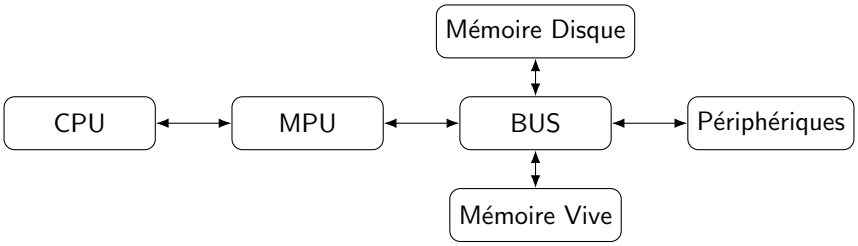
Définition 2.1. Décrit l'agencement interne de composants électroniques ainsi que leurs interactions.

Phrase. représente tous les ressources physiques disponibles et accessibles de l'ordinateur

Comment une ordinateur a plusieurs parties et le but de ce cours n'est pas de connaître tout on va considère une version très simplifiée d'un ordinateur.

2.1.1. Architecture Von Neumann

Définition 2.2. Architecture en que la mémoire est unique et utilisée pour stocker les instructions et les données des programmes.



On considère :

1. **CPU** : Control Process Unit ;
2. **MPU** : Memory Protect Unit ;
3. **BUS** : Connections entre les partis ;
4. **Mémoire** :
   (a) `Mémoire Disque` : géré pour le Système de Fichiers :
      i. HD ;
      ii. SSD ;
   (b) `Mémoire Vive` :
      i. RAM ;
5. **Périphériques** :
   (a) `Entrée / Sortie`, géré pour le Système de Fenêtres ou Shell :
      i. Clavier ;
      ii. Souris ;
   (b) `Réseau`, géré pour la Pile de Réseau ;

Remarque. La Mémoire peut être interprété comme un Périphérique.

3. Hardware

Après avoir pris connaissance des partis principales d'un ordinateur on precise quelques informations nécessaires pour mieux comprendre :

3.1. CPU

Partie centrale de tous les ordinateurs :

> **Définition 3.1.** Control Process Unit, responsable pour l'exécution des instructions.

À fin de comprendre comment les instructions sont interprété on les divise dans les étapes suivants :

1. **Fetch**

   > **Définition 3.2.** Récupération du mot d'instruction pointé par le pointeur d'instruction.

   En anglais, le Pointeur d'Instruction est appelé **program counter**, PC.

2. **Decode**

   > **Définition 3.3.** Activation desc composants et chemins correspondant à l'instruction.

3. **Execute**

   > **Définition 3.4.** Traitement de l'instruction par les composants activés.

4. **Memory Access**

   > **Définition 3.5.** Lecture ou écriture des données de la mémoire, transitant par le bus

5. **Writeback**

   > **Définition 3.6.** Écriture des registres incluant la mise à jour de PC.

Chaque étape a plusieurs informations supplémentaires importants qui sont en dehors de ce cours.

3.2. MPU

La CPU est responsable pour la réalisation d'instructions et d'autres composants électroniques protègent la CPU des erreurs. La **MPU** est responsable pour protéger la mémoire :

> **Définition 3.7.** Memory Protect Unit, positionné entre le CPU et le BUS, fait la gestion des registres spéciaux en guardant :
>
> 1. Adresse de début ;
> 2. Adresse de fin ;
> 3. Permission ;

Tout accès mémoire en dehors de ces plages provoque une interruption.

> **Remarque.** Il faut modifier le paramétrage de la MPU quand on change de thread, seulement le noyau peut y acceder car le changement des registres de protection d'une MPU est privilégié.

3.3. MMU

La **MMU** est responsable pour estoquer les informations adresses de mémoire :

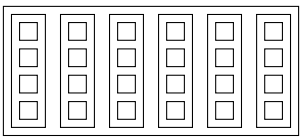
> **Définition 3.8.** Memory Management Unit sera responsable pour faire les conversions nécessaires entre adresses d'hardware et adresses utilises en software en les stockant dans une table.

Généralement n'est pas possible d'utiliser des régions continu de mémoire pour enregistrer les données d'une programme alors les mécanises suivant sont utilisés :

1. **Fragmentation**

   > **Définition 3.9.** Quand la mémoire des plusieurs programmes sont divisés en plusieurs morceaux la mémoire est considère **Fragmenté**.

2. **Pagination** :

   > **Définition 3.10.** Division de la mémoire en zones de taille fixe qui s'appellent de **pages**.
   >
   > 
   >
   > > **Remarque.** Couramment la taille de la page est de 4ko.
   >
   > Quand il y a de perte de mémoire quand toute la page n'est pas utilisée on considère qu'il y a **fragmentation interne** de la mémoire.

Dans le deux cas la mémoire est divise, donc le MMU doit faire la gestion des adresses pour que les programmes arrivent à bien exécuter les programmes. Quand on fait **Fragmentation** on considère qu'on aura **Mémoire Virtuelle** :

> **Définition 3.11.** Décomposition des adresses physiques en adresses virtuelles pour faciliter la distribution d'hardware et la compréhension de software.
>
> > **Remarque.** Décomposition des adresses virtuelles en **numéro de page**, bits plus significatifs, et un **déplacement**, bits moins significatifs.

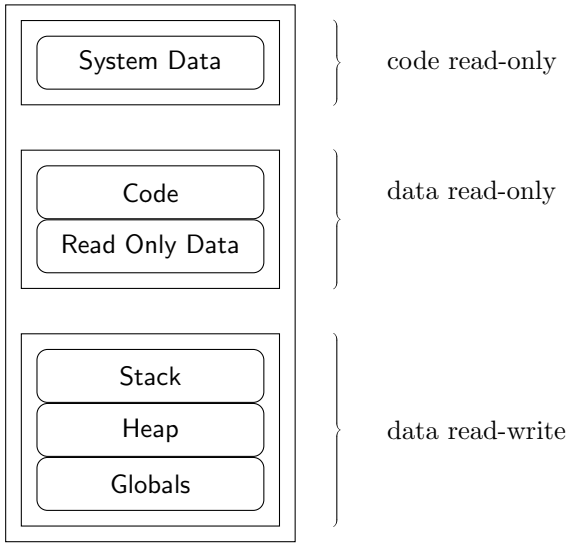
Si la translation d'adresse échoue soit il n'y a pas de page correspondant à une adresse virtuelle dans la table des pages, page non mappée, ou soit il y a une page mais accessible avec des droits différents. Ainsi une interruption est générée : défaut de page, et quand la mémoire virtuelle est utilisée seulement pour la protection mémoire la solution sera : tuer ou redémarrer la tâche.

> **Définition 3.12.** Ensemble des adresses accessibles par des Threads sera **l'Espace d'Adressage**.

3.4. Mémoire

La communication entre le CPU et la Mémoire doit avant passer pour la MPU et pour la MMU. Comme acceder la Mémoire prendre beaucoup plus de temps que exécuter une instruction il faut être sur que les accès mémoire sont bons.

Définition 3.13. Memory is the part of the computer that holds data and instructions for processing, normally divided in :



Où :

1. **code read-only** : contient les instructions ;
2. **data read-only** : contient constantes, chaîne des caractères ;
3. **data read-write** : divisé en :
   (a) `stack` (pile) : variables locales d'un thread ;
   (b) `heap` (tas) : allocation dynamique de mémoire ;
   (c) `globals` : allocation statique de mémoire ;

Remarque. Although closely associated with the CPU, memory is separate from it.

Remarque. Memory stores program instructions or data for only as long as the program they pertain to is in operation.

3.4.1. Confinement Mémoire

On peut imaginer que comme les données ne sont pas continues dans la mémoire il peut avoir des access interdits. À fin d'éviter la corruption de la mémoire il faut la protéger :

Définition 3.14. Assurer que l'exécution d'un code ne puisse pas nuire au reste du système.

Assurer que l'exécution d'un code ne peut pas accéder, lire et écrire, aux zones mémoires du reste du système. Dans le cas des threads consiste à isoler l'exécution dans des espaces d'adressages séparés en visant :

1. améliorer la **sûreté** / disponibilité :
   (a) autres tâches continuent de tourner sans être impactées ;
   (b) détection d'une erreur permet de redémarrer la tâche fautive ;
2. faciliter la **mise au point** :

   (a) détecter les erreur au plus tôt ;

3. améliorer la **sécurité** :
   (a) limiter la prise de contrôle d'un attaquant à un seul composant ;
   (b) empêcher la lecture de secrets ;

On peut réaliser la séparation :

1. **physique** assure absence d'interférence entre programmes :
   (a) machines différents ;
   (b) mémoires et processeurs séparées sur une même carte ;
   (c) coeur d'exécution séparés sur une même puce ;

> **Remarque.** On considère les caractéristiques suivants :
>
> **Avantages** :
> (a) plus haut niveau de confinement ;
>
> **Inconvénients** :
> (a) coûte cher ;
> (b) pas mémoire partagée ;

2. **logicielle** assure absence d'interférence entre programmes sans support matériel :
   (a) techniques **dynamiques** : vérification pendant l'exécution ;
   (b) techniques **statiques** : vérification avant ou à la compilation ;

> **Remarque.** On considère les caractéristiques suivants :
>
> **Avantages** :
> (a) pas de modification au processeur ;
>
> **Inconvénients** :
> (a) surcoût à l'exécution ;

3. **matérielle** assure absence d'interférence entre les programmes sans support logicielle :
   (a) utilisation d'une MPU ;

4. Software

Ici on ajoute les principales informations de software nécessaires pour comprendre comment l'ordinateur s'assemble.

4.1. Taille d'Information

Il y a différents unités d'information utilises quand on étude les Systèmes d'Exploitation qui sont utilises dans différents contextes. D'entre les principales ont a les suivants :

1. **bit** : souvent utilisé pour la mémoire ;

   > **Définition 4.1.** Valeur binaire, soit 0 ou soit 1. La plus petite unité d'information.

2. **octet** : souvent utilisé pour la mémoire adressable ;

   > **Définition 4.2.** Généralement un ensemble de 8 bits, plus petite unité de mémoire adressable.
   >
   > > **Remarque.** Nomme **byte** en anglais.

3. **mot** : souvent utilisé pour les instructions ;

   > **Définition 4.3.** Ensemble de plusieurs `octets`. La unité de mémoire manipulée naturellement par un processeur : 8, 16, 32, 64 ou 128 bits.
   >
   > > **Remarque.** Nomme **word** en anglais.

Comment chaque unité est utilisé à un situation precise différent il faut toujours faire du **Padding**.

> **Définition 4.4. Padding** consiste à garantir que la taille des données soit compatible avec les algorithmes utilisés.

4.2. Endianness

Après savoir comment les différents tailles de données sont classifiées c'est important de comprendre comment les données sont estoques sur mémoire. C'est-à-dire de quel façon les bits seront distribuées dans la mémoire. Il y a deux principales façons de le faire :

1. **little endian** : utilisé dans RISC-V :

   > **Définition 4.5.** On commence par l'octet plus petit, LSB, droite à gauche ;

2. **big endian** :

   > **Définition 4.6.** On commence par l'octet plus grand, MSB, gauche à droite ;

Convention pour lire les octets dans un mot on considère :

Exemple 4.1. On considère le code suivante :

```
variable = {0x11, 0x22, 0x33, 0x44};

.word 0x11223344  # code little endian, plus commun
.word 0x44332211  # code big endian
```

On peut convertir les variables d'une codage à une autre avec l'algorithme.

4.3. Système d'Exploitation

On étude les Systèmes d'Exploitation et voici leur définition :

Définition 4.7. Au minimum un Système d'Exploitation est un ensemble composé de :

1. **noyau** 5.1 ;
2. **services** 5.4 ;
3. **librairies** ;

Parmi les utils nécessaires on considère :

1. **Système de Fenêtres** :

   Définition 4.8. Seulement la fenêtre sélectionnée doit recevoir les entrées clavier et souris ;

2. **Système de Fichiers** :

   Définition 4.9. Responsable pour organiser le partage de la Mémoire de Masse entre les différents fichiers.

   Remarque. Une mémoire de grande capacité, non volatile et qui peut être lue et écrite, entre autres, par un ordinateur est considère comme **Mémoire de Masse**.

   On considère aussi :

   (a) `Fichier` : Représente un ensemble de secteurs, adressés en continu.
   (b) `Répertoires` : Organisent les fichiers en une arborescence.

4.3.1. Virtualization

Quand on a besoin de utiliser une Système d'Exploitation dedans une autre OS il faut bien encapsuler le programme pour éviter conflits, pour ça on utilise la **Virtualization** :

Définition 4.10. Une encapsulation, sandbox, permet l'exécution d'un ou plusieurs programme de manière à limiter les dommages liés à leurs défauts ou vulnérabilités notamment limite la liste des privilèges accessible à ces programmes.

Mécanismes de sandboxing :

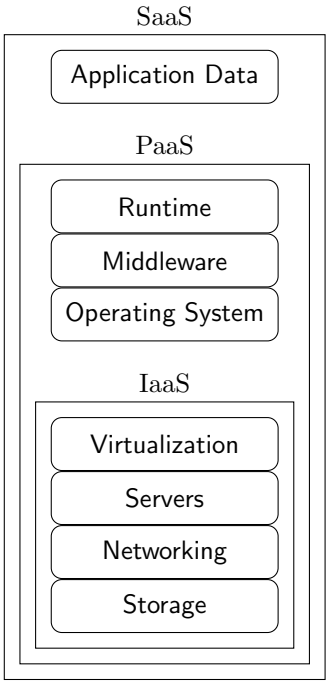
1. **seccomp** : filtre d'appel système
2. **containers** : OS-level virtualization

Généralement, si le mécanisme fait croire à une programme que celui-ci s'exécute tout seul sur la machine, on parle de virtualisation.

4.4. Cloud Service Models

Quand on va developer un nouveau produit il faut le bien classifier pour savoir comment le vendre :

Définition 4.11. They are sometimes referred to as Cloud Service Models or Cloud Computing Service Models are the different parts that compose a product, normally classified as :



Software as a Service : Google Applications

Platform as a Service : Google App Engine

Infrastructure as a Service : Google Compute Engine

En tout cas on considère les caractéristiques essentielles suivants :

1. **On Demand Self Service** : le consommateur accède au service selon ses besoins sans interaction avec un humain ;
2. **Broad Network Access** : les ressources doivent être accessibles à travers le réseau à l'aide de mécanismes standards ;
3. **Ressource Pooling** : les ressources sont dynamiquement ré-assignées en fonction des demandes des utilisateurs différents suivant un modèle multi-tenant ;
4. **Rapid Elasticity** : les ressources doivent être allouées et désallouées rapidement ;
5. **Mesured Service** : la consommation des ressources doit être mesurée et contrôlée à l'aide d'un mécanisme transparent ;

Finalement il y a différent façons de déployer ces produits, parmi eux on evidence :

1. **Cloud Privé** : l'infrastructure est provisionnée pour les besoins exclusifs d'une seule organisation qui comprend plusieurs entités ;
2. **Cloud Communautaire** : infrastructure est provisionnée pour les besoins exclusifs d'une communauté de clients ;
3. **Cloud Public** : l'infrastructure est ouverte à tout le monde
4. **Cloud Hybride** : combinaison des déploiements précédents par exemple utilisation d'un cloud public pour absorber des pics de charge.

5. Noyau

Concevoir une ordinateur demande beaucoup de couches de softwares pour faire la gestions d'utilisation de ses ressources sans conflit. La plus basique sera le **Noyau** :

Définition 5.1. Partie fondamentale du Système d'Exploitation, responsable privilégié pour gérer les ressources de l'ordinateur :

1. `allocation` : comment répartir la mémoire
2. `ordonnancement` : définir quel processus s'exécute à un instant ;

Remarque. Nomme **kernel** en anglais.

Quand on parle de division de ressources il faut considérer quelques définitions qui seront utilises dans le reste de ce document :

1. **Tâche** :

   Définition 5.2. Terme ambigu qui désigne souvent un Processus mais parfois de Thread. Il faut faire attention à son utilisation.

   On utilisera tache pour décrire un parti d'un programme qui veut utiliser les ressources de l'ordinateur.

2. **Processus** :

   Définition 5.3. Séquence d'exécution indépendante qui peut avoir des différents états d'exécution avec ses permissions et espace d'adressage séparé.

   Remarque. En général une instance de l'exécution d'un programme est considère comme Processus.

   Phrase. un processus peut être vu comme une **machine virtuelle** disposant de :

   (a) `thread` : CPU virtuel ;
   (b) `espace d'adressage` : mémoire virtuelle ;
   (c) `périphériques virtuels` : permettent d'acceder des périphériques réels ;
   (d) `hyperviseur` : noyau virtuelle ;

   Processus est beaucoup utilise et nécessaire pour les Threads. En cas de besoin, jeter un coup d'oeil à la Section de Threads.

Chaque fonction du **Noyau** sera fait pour une partie différent. On divise les fonctions plus important à la sequence.

5.1. Service

On considère qu'il y aura plusieurs **Services** dans un **Noyau** :

Définition 5.4. Programme qui organise le partage de ressources de l'ordinateur.

Parmi on met en evidence :

1. **Pilote Périphérique** :

> **Définition 5.5.** Service responsable pour organiser le partage des périphériques de l'ordinateur en communicant avec le **noyau** 5.1.
>
> > **Remarque.** Nomme **driver** en anglais.

5.2. Ordonnanceur

On considère que comme l'ordinateur a besoin de faire la gestion de plusieurs tâches il faut qu'il puisse choisir quoi faire à un instant donne et pour ça on a le **Ordonnanceur** :

> **Définition 5.6.** Responsable pour choisir l'ordre d'exécution des processus sur les processeurs, c'est à dire l'attribution de travaux à des resources à partir de :
>
> 1. **Politique d'Ordonnancement** ;
> 2. **Algorithme d'Ordonnancement** ;
>
> > **Remarque.** En anglais, l'Ordonnanceur est appelé **scheduler**.

On ne considère pas l'allocation statique de mémoire comme ordonnancement.

5.2.1. Politique d'Ordonnancement

> **Définition 5.7.** Contraintes permettant de décider à tout instant quel travail exécuter par les travaux prêts en considérant les aspects suivants :
>
> 1. **Coût de Préemption**, coût pour interrompre un travail :
>   (a) `élevé` :;
>   (b) `faible` :;
> 2. **Coût de Migration**, coût pour changer d'une ressource à une autre :
>   (a) `élevé` :;
>   (b) `faible` :;
> 3. **Critère d'Optimisation**, peut varier d'application à application :
>   (a) `débit` : maximiser le nombre de tâches faites ;
>   (b) `latence` : minimiser les temps de réponse à des évènements ;
>   (c) `équité` : faire en sorte que toutes les t6aches aient accès aux ressources ;

5.2.2. Algorithme d'Ordonnancement

Représente la Politique d'Ordonnancements désiré pour le noyau. On peut classifier de la façon suivante :

1. **Statique** : plan d'ordonnancement est décide avant l'exécution ;
   (a) McNaughton ;
2. **Dynamique** : plan d'ordonnancement est décide pendant l'exécution ;
   (a) `Non Préemptif` : les tâches ne peuvent pas être interrompre ;
      i. FIFO : First In First Out, or PEPS : Premier Entré Premier Sorti ;

> **Remarque.** File d'Attente, en anglais **queue**, implémente FIFO.

On note que c'est optimal s'il y a une seule ressource et il n'y a pas besoin d'aucune connaissance sur les travaux. Par contre, pas de prise en compte de priorités et pas de parallélisation du CPU.

ii. LIFO : Last In First Out, or DEPS : Dernier Entré Premier Sorti ;

> **Remarque.** Pile, en anglais **stack**, implémente LIFO.

iii. SJF : Shortest Job First ;

iv. EDF : Earliest Deadline First ;

(b) `Préemptif` : les tâches peuvent être interrompre ;

i. Priorité Statique : priorité des travaux ne change pas ;

A. Priorité Fixe ;

B. Round-Robin ;

ii. Priorité Dynamique : priorité des travaux peut changer ;

A. EDF ;

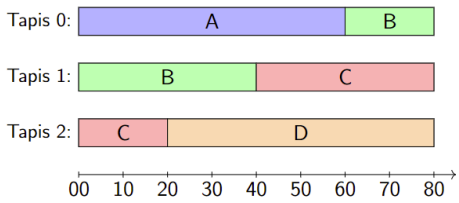
B. Multi-Level Feedback ;

McNaughton

Définition 5.8. Algorithme d'Ordonnancement a McNaughton

> **Exemple 5.1.** 4 étudiants veulent faire 60 abdominaux chacun, 1 par seconde, en y passant un minimum de temps. Il n'y a que 3 tapis. Comment le faire ?

> **Résolution.** Il y a 240 secondes de travail à faire, 60 abdominaux chaque étudiant. Donc le temps minimum pour tout faire est de 80 secondes en considérant que les tapis sont utilisés au même temps. À la fin on aura :



> **Remarque.** On considère les caractéristiques suivants :
>
> **Avantages** :
>
> 1. optimal sur multi-processeur ;
>
> **Inconvénients** :
>
> 1. faire des migrations ;
> 2. connaître tous les travaux ;

Round-Robin

Définition 5.9. As the term is generally used, time slices, also known as time quanta, are assigned to each process in equal portions and in circular order, handling all processes without priority, also known as cyclic executive.

> **Exemple 5.2.** Vous êtes seul et devez nourrir trois bébés. Lorsqu'un bébé attend sa cuillerée trop longtemps, il pleure. Dans quel ordre nourrissez vous les bébés ?

Remarque. On considère les caractéristiques suivants :

Avantages :

1. simple ;
2. easy to implement ;

Inconvénients :

1. multiples préemptions prennent du temps CPU ;

Multi-Level Feedback Queue

Définition 5.10. Amélioration de l'algorithme du Round-Robin avec plusieurs files qui change à partir de deux situations :

1. **remonte** dans les priorités, quand un thread est :
   (a) bloqué ;
   (b) attends depuis trop longtemps ;
2. **descend** dans les priorités, quand un thread est :
   (a) utilise tout son quota de temps ;

Priorité Fixe

Définition 5.11. Un **Système Temps-Réel** est un système qui a la capacité de répondre à des évènements asynchrones issus du monde extérieur dans des délais pré-déterminés.

Phrase. temps-réel est différent de rapide

Généralement le système sera divise en deux types de tâches qui seront découper en différents threads :

1. **Périodiques** déclenchement sur timer avec durée fixe ;
2. **Sporadiques** déclenchement sur interruption avec une durée minimale ;

Définition 5.12. Algorithme d'Ordonnancement a Priorité Fixe

Théorème 5.1. Si on peut ordonnancer les tâches en priorité fixe, on saure le faire en affectant les priorités dans l'ordre inverse des périodes.

Théorème 5.2. Si la charge du système est $< 69\%$, alors les tâches sont ordonnançables en priorité fixe.

Théorème 5.3. En monoprocesseur, tout système de tâche périodique faisable, charge processeur $< 100\%$ est faisable avec l'algorithme EDF.

Remarque. EDF reste optimal sur des tâches non périodiques. Connaissance des durées de calcul des tâches non nécessaire.

Phrase. Il n'existe pas d'ordonnancement optimal en multi-processeur.

5.3. Sécurité

Quand on considère un Système d'Exploitation il faut toujours considère comment protéger des attaques externes et des conflits pour garantir l'intégrité du système :

> **Définition 5.13.** Découpage des fonctionnalités en processus indépendants en laissant les services : parties sensibles, minimales et incontournables.

On utilise des **Principes de Sécurité de Saltzer et Schroeder** pour construire et évaluer la sécurité d'un OS :

1. **Open Design** :

   > **Définition 5.14.** ne pas se reposer sur le fait que les attaquants ignoreront certains détails.

2. **Psychological Acceptability** :

   > **Définition 5.15.** si la sécurité d'un système le rend trop pénible à utiliser, les utilisateurs contourneront les protections.

3. **Work Factor**

   > **Définition 5.16.** Concevoir le système en évaluant les ressources d'un attaquant.

4. **Compromise Recording** :

   > **Définition 5.17.** On peut parfois reporter son attention sur la détection d'une compromission plutôt que sur sa prévention.

Pendant la construction d'OS il faut faire la séparation des privilèges en isolant les compromission de chaque partie du système.

> **Remarque.** Il faut minimiser les mécanismes communs pour minimiser l'impact d'une compromission.

Chaque parti du système sera responsable pour faire la séparation et protection :

1. **processus** : séparation des tâches en domaines de protection ;
2. **micro-noyau** : séparation des services en domaines de protections ;
3. **exo-noyau** : minimisation des noyau et services ;

5.3.1. Gestion de Privilèges

Au fur et à mesure que le système exécute ses fonctionnalités il faudrait avoir des utils pour gérer qui a accès à quoi. On se base toujours au **Principe du Moindre Privilège** :

> **Définition 5.18.** Principe du Moindre Privilège : autoriser seulement le nécessaire. Tout ce qui n'est pas explicitement autorisé est interdit.

Pour bien identifier et localiser la relation entre les fichiers et les processus on considère :

1. **processus** : associé à :
   (a) un user id ;
   (b) un groupe id ;
2. **fichier** : associé à :

(a) un utilisateur ;
(b) un groupe ;
(c) les permissions :
    i. Read ;
    ii. Write ;
    iii. eXecute ;

Un fichier central associe les utilisateurs au groupe.

Quand un fichier est ouvert, open, le noyau renvoie un numéro : le file descriptor qui sera utilisé pour les opérations sur le fichier.

Dans une système UNIX les permissions ne sont utilisées que pour l'ouverture du fichier. ensuite toutes les opérations sont faites en utilisant le file descriptor.

ACL

Quand on considère une ressource il faudrait implementer une contrôle d'accès un **Access Control List, ACL** :

> **Définition 5.19. Access Control List** pour les ressources, est une ensemble des listes de quels Processus peuvent faire quoi dans une Processeur.

Avec les caractéristiques suivants :

1. **Efficacité** :
   (a) nécessite de mécanismes supplémentaires ;
   (b) un espace de nommage, nommer la ressource à laquelle on veut accéder ;
   (c) un mécanisme d'identification ;
2. **Responsabilité**, savoir qui peut avoir accès à quoi :
   (a) la liste donne l'information directement ;
3. **Revocation**, supprimer l'accès :
   (a) supprimer de la liste ;

Capacités

Quand on considère une processus il faudrait implementer une liste de **Capacités** :

> **Définition 5.20. Capacités** pour les tâches, est une liste de ce que une Processus peut faire pour chaque autre ressource d'un ordinateur.

Avec les caractéristiques suivants :

1. **Efficacité** :
   (a) la capacité sert à nommer l'objet c'est un pointeur vers la ressource ;
2. **Responsabilité**, savoir qui peut avoir accès à quoi :
   (a) peut être plus difficile ;
3. **Revocation**, supprimer l'accès :
   (a) plus difficile : necessite de metre en place un proxy ;

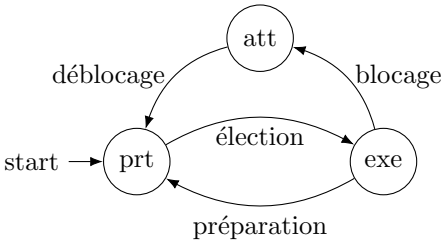
6. Multi-Tâche

Quand on considère une OS il y aura Multi-Tâches qui se déroulent en même temps. On considère qu'une **Tâche** sera :

Définition 6.1. Terme ambigu, souvent synonyme de Processus mais parfois de Thread. Il faut faire attention à son utilisation.

6.1. Thread

Définition 6.2. Séquence d'exécution indépendante qui peut avoir des statuts suivants :



Où : prt est **prêt**, att est **en attente** et exe est **en exécution**.

Remarque. L'Unité d'Ordonnancement est responsable pour ordonner les Threads.

6.1.1. Processus

Définition 6.3. L'Ensemble Threads et ses permissions, donc avec son espace d'adressage séparé.

Remarque. En général une instance de l'exécution d'un programme est considère comme Processus.

Phrase. un processus peut être vu comme une **machine virtuelle** disposant de :

1. `thread` : CPU virtuel ;
2. `espace d'adressage` : mémoire virtuelle ;
3. `périphériques virtuels` : permettent d'acceder des périphériques réels ;
4. `hyperviseur` : noyau virtuelle ;

6.1.2. Atomicité

Définition 6.4. `f` est **atomique** par rapport à `g` si `f` apparaît s'exécuter d'un seul coup sans être interrompue par l'exécution de `g`.

Deux fonctions `f` et `g` sont atomiques si l'exécution entrelacée de `f` et `g` est équivalente à une exécution séquentielle de `f` puis `g` ou de `g` puis `f`.

6.2. Changement de Contexte

Définition 6.5. Consiste à sauvegarder en mémoire la valeur de tous les registres du thread en cours d'exécution, par exemple les données sur la pile, puis à restaurer les registres d'un thread dont les valeurs ont été précédemment sauvées.

Phrase. essentially suspending the process and then resuming it.

Remarque. Sert à implémenter plusieurs fils d'exécution, threads, indépendants sur un seul processeur.

Exemple 6.1. On suppose qu'on est en train de lire un livre. S'on est appelé pour faire un autre tâche plus important, on marque la page, fait la tâche et après on reprend là où on s'est arrêté pour continuer à lire.

S'il arrive un demande de changement de contexte pas immédiate le processus doit attendre qui le CPU prend connaissance de cette demande. Le CPU faire constamment du **Polling** pour découvrir s'il a besoin d'agir :

Définition 6.6. The process in which the CPU constantly checks the status of the device to see if it needs the CPU's attention.

6.3. Interruptions

Síl arrive un demande de changement de contexte immédiate le processus ira envoyer une **interruption**

Définition 6.7. Mécanisme permettant de signaler au processeur un évènement requérant son attention immédiate.

Remarque. Le **Interrupt Handler** c'est le service d'interruption qui est appellee comme réponse du processeur à une interruption.

Lors d'une interruption il faut :

1. **sauvegarder** les données anciennes :
   (a) pc ;
   (b) sp ;

   Généralement ce process est automatique et considère flags et d'autres registres utilises.
2. **recevoir** les nouveaux valeurs :
   (a) pc reçoit une adresse fixe qui dépend du numéro d'interruption ;
   (b) sp reçoit l'adresse d'une pile du noyau ;
3. **exécution** de la routine ;
4. **restauration** de tous les registres du thread interrompu ;

Quand il y a une interruption pendant le traitement d'une autre interruption les nouvelles interruption sont masquées, ça veut dire que sont mises en attentes.

Remarque. Masquage des interruptions introduit de la latence dans le traitement des interruptions. Ainsi les interrupt handlers peuvent découper le traitement pour les exécuter plus rapidement.

Sur Unix deux fonctions peuvent être utilises avec le changement de contexte :

1. **mmap()** :

   > **Définition 6.8.** appel système UNIX permettant de projeter un fichier en mémoire qui récupère automatique une donnée, sur le disque dur, de la partie du fichier lors des défauts de page.

2. **fork()** :

   > **Définition 6.9.** duplique un processus. initialement toute la mémoire est partagée entre les deux copies jusqu'à première modification qui modifiera l'accès aux pages partagées en lecture seule.

Les Interruptions peuvent arriver de différent parties de l'ordinateur comme :

1. **Interruptions Matérielles** :

   > **Définition 6.10.** Interruptions qui provient d'un périphérique matériel.

   Comme :

   (a) entrée clavier ;
   (b) arrivée d'un paquet réseau ;
   (c) fin de traitement d'une lecture ou écriture disque ;

   > **Remarque.** Programmée d'une Horloge est nomme **timer** en anglais.

2. **Interruptions Logicielles**

   > **Définition 6.11.** Interruptions qui provient du processeur lui-même.

   Comme :

   (a) division par zéro ;
   (b) accès à une zone mémoire interdit ou impossible ;
   (c) interruptions volontaires ;

   > **Remarque.** Interruptions Volontaires est nomme **trap** en anglais.

6.3.1. Préemption

> **Définition 6.12.** Après une interruption un contexte différent est restauré au lieu du courant.
>
> > **Remarque.** Interruptions sont le mécanisme matériel à la base de l'Ordonnancement Préemptif.

6.4. Synchronisation

> **Définition 6.13.** Mécanismes permettant de coordonner l'exécution de plusieurs threads en bloquant leur exécution à des points de programmes précis pour régler les problèmes de concurrence sur l'accès à une ressource, logicielle ou matérielle, partagée.

Remarque. Bloquer c'est à dire changer le statut de tâches à bloqué.

Remarque. Débloquer c'est à dire changer le statut des taches à prêt.

Pendant le changement entre threads la CPU restera inactif pour un des threads et donc perdra du temps d'exécution. Il faut copier les adresses de mémoire nécessaires que prend un peu de temps. On considère les méthodes de synchronisation suivants :

1. `mutex` :

   Définition 6.14. Utilisation de code qui permet de gérer des conflit pour ressources.

2. **Concurrence** :

   Définition 6.15. Quand les tâches accèdent simultanément à une même ressource.

3. **Parallélisme** :

   Définition 6.16. Quand il y a utilisation de plusieurs resources.