

# LINFO1252 - LSINC1252

## Systèmes Informatiques



### Leçon 1 : Introduction

*Pr. Etienne Rivière*

[etienne.riviere@uclouvain.be](mailto:etienne.riviere@uclouvain.be)



École polytechnique de Louvain

# Système informatique ?

- Combinaison de matériel ...
  - Processeur
  - Mémoire
  - Périphériques d'entrée/sortie
  - Périphériques de stockage
- ... et d'un Système d'Exploitation (SE)
  - Lien fondamental entre les applications et le matériel
  - Fournit au programmeur et à l'utilisateur des mécanismes et outils pour tirer partie du matériel *simplement et efficacement*

# Objectifs de ce cours

- *Utiliser et comprendre les systèmes informatiques*
  - Et particulièrement UNIX & GNU/Linux
- ✓ En tant que programmeur : utilisation des services fournis par le SE
  - Interface programmatique (API) et outils systèmes
- ✓ Design et mise en œuvre des SE
  - Illustrer les compromis entre niveau d'abstraction et performance

# Thèmes abordés

- Structure des ordinateurs & architecture
- Services fournis par un SE
  - Processus et threads,
  - synchronisation et communication,
  - gestion de la mémoire,
  - gestion des fichiers
- Mise en œuvre d'un SE : sujets choisis
  - Structure et organisation des SE,
  - mise en œuvre de la synchronisation,
  - mise en œuvre des systèmes de fichiers,
  - ordonnancement (scheduling) des processus

# Organisation du cours

# Ressources en ligne indispensables

- Inscription sur Moodle
  - Code LINFO1252-LSINCI252
  - Informations sur les séances de chaque semaine
- Inscription au groupe Teams
  - Code “Cours-LINFO1252-2023-24”
- Exercices du cours sur Inginious
  - <https://inginious.info.ucl.ac.be/>
  - [LINFO1252] [LSINCI252] Systèmes Informatiques

# Matériel du cours

## ● **Syllabus en ligne**

- <https://sites.uclouvain.be/SystInfo>

### 1. Théorie

- Matière du cours
- Couvert dans les cours magistraux

### 2. Outils

- Apprentissage des outils système
- Approfondissement
- Contributions d'étudiants (en partie)

### 3. Exercices

- Mis à jour chaque semaine avec les sujets d'exercices
- Liens vers exercices Inginious

# Communication

- Interlocuteur principal :  
votre tuteur et votre assistant
- Forums sur Moodle pour les questions  
intéressant aussi les autres étudiants
- Canaux Teams
- Email pour les cas particuliers
  - ou pour une demande de rendez-vous
- SVP pas d'appel ou de message instantané sur  
Teams sans rendez-vous préalable !

# Equipe LINFOI 252

- Prof Etienne Rivière
- 4 assistants



Aurélien Buchet



François de Keersmacker



Maxime Piraux



Tom Rousseaux

- et 6 tuteurs/trices



Alexandre  
de Salle



Sacha  
Defrère



Martin  
Gyselinck



Jean-Baptiste  
Hontoir



Quentin  
Prieels



Raphaëlle  
Wats

# Organisation (LINFO1252)

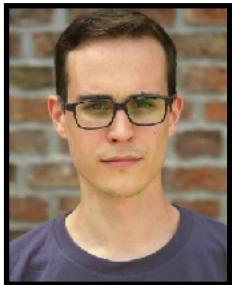
- Cours magistraux le jeudi 10h45 en SUDII
  - Exception en S1 et S2 : lundi 10h45
- Une séance d'exercices par semaine
  - Travaux dirigés en classe
  - Aide à l'avancement dans les exercices Inginious
  - Soutien sur la réalisation des projets
- La séance d'exercice est **obligatoire**
  - Mercredi 8h30-10h30 ou vendredi 8h30-10h30
  - Salles SUD, BARB ou MERC — consultez ADE
- Channels Teams pour les questions d'intérêt général

# Séance d'exercices : qui et quand

Séance	Étudiants
Mercredi 8h30	Bac et Master FSA  Tous les autres cas (e.g. DAT*)
Vendredi 8h30	Bac et Master SINF

# Classes

- Chaque classe sous la responsabilité d'un assistant + une ou un tutrice/tuteur
  - Mercredi :



Maxime Piraux



Aurélien Buchet



Alexandre  
de Salle



Martin  
Gyselinck



Jean-Baptiste  
Hontoir

- Vendredi :



François de Keersmacker Tom Rousseaux



Tom Rousseaux



Sacha  
Defrère



Quentin  
Prieels



Raphaëlle  
Wats

# Matériel

- Vous utiliserez votre propre laptop
  - Contactez moi après le cours si vous n'en avez pas à votre disposition
- Il est nécessaire d'installer une distribution Linux, par exemple Ubuntu
  - Recommandé : dual-boot en natif
    - Faites une sauvegarde de vos données au préalable
  - Possible : machine virtuelle avec Virtual Box (gratuit)
  - Aide de l'équipe et KAP Linux pour l'installation (cf fin du cours)
  - Note: WSL (Windows Subsystem for Linux) n'est pas une option valable pour ce cours. L'installation d'un système GNU/Linux complet fait partie des acquis d'apprentissage. Aucun support sur WSL ne sera fourni par l'équipe



# Présentation Kot-à-Projet Louvain-li-Nux



# Évaluation (I)

## Session de **janvier** :

- Participation aux séances et exercices annoncés comme obligatoires (10%)
  - Pointage des présences en TD/TP
  - Essai > 0% avant l'échéance des exercices Inginious
- Evaluation continue et mini-projets (30%)
  - 3 mini-projets
  - Ensemble d'exercices Inginious
- Examen (60%)

## Session de **septembre** :

- La note de participation compte et ne peut pas être refaite (10%)
- Réalisation d'exercices et de mini-projet(s) de façon individuelle (30%)
- Examen (60%)

# Évaluation (2)

Les activités d'évaluation continue sont **toutes des activités certificatives** et les règles de déontologie en termes de **plagiat, triche**, etc. s'y appliquent **strictement**.

Les activités d'évaluation continue sont toutes **strictement individuelles** sauf mention explicite du professeur. Pour les activités autorisées en binôme, toute **collaboration** avec des personnes en dehors du binôme d'étudiant constitue **un cas de triche**.

L'utilisation d'**IA générative** est autorisée pour corriger la grammaire, l'orthographe et le style de texte préalablement écrit par un ·e étudiant ·e mais **interdite pour générer du texte à partir d'une instruction ou pour générer du code**.

Des solutions de **détection de plagiat** sont utilisés de manière **systématique**. (Y compris avec l'intégralité des réponses des années précédentes)

# Évaluation (2)

Des activités formatives sont susceptibles d'être considérés comme certificatives et prendront alors une partie ou tout du poids de l'examen dans la note si les circonstances le demandent.

L'examen peut utiliser tout ou partie des modalités d'évaluation suivantes en proportion variable. Cette proportion est annoncée lors de l'examen :

- restitution de connaissances sous forme de question de cours ouvertes.
- application de connaissances sous forme de problème.
- QCM et QRM appliquant le principe de "standard setting" : une réponse incorrecte à une question ne conduit pas à une pénalité, et la partie ne peut pas être quotée négativement, mais un seuil minimal (annoncé) de réponses correctes est nécessaire avant d'accumuler effectivement des points pour cette partie de l'examen.

# Cours d'aujourd'hui

- Deux parties
- I. Une vue “haut niveau” de ce qu’est et de comment fonctionne un système informatique
    - Objectif : comprendre le rôle et la place du système d’exploitation
    - Nous verrons en détail la mise en œuvre des mécanismes mentionnés au cours du quadrimestre
  2. Introduction à l’utilisation du shell (ligne de commande)

# **Partie I : Le système informatique et le rôle du système d'exploitation**

# Syllabus

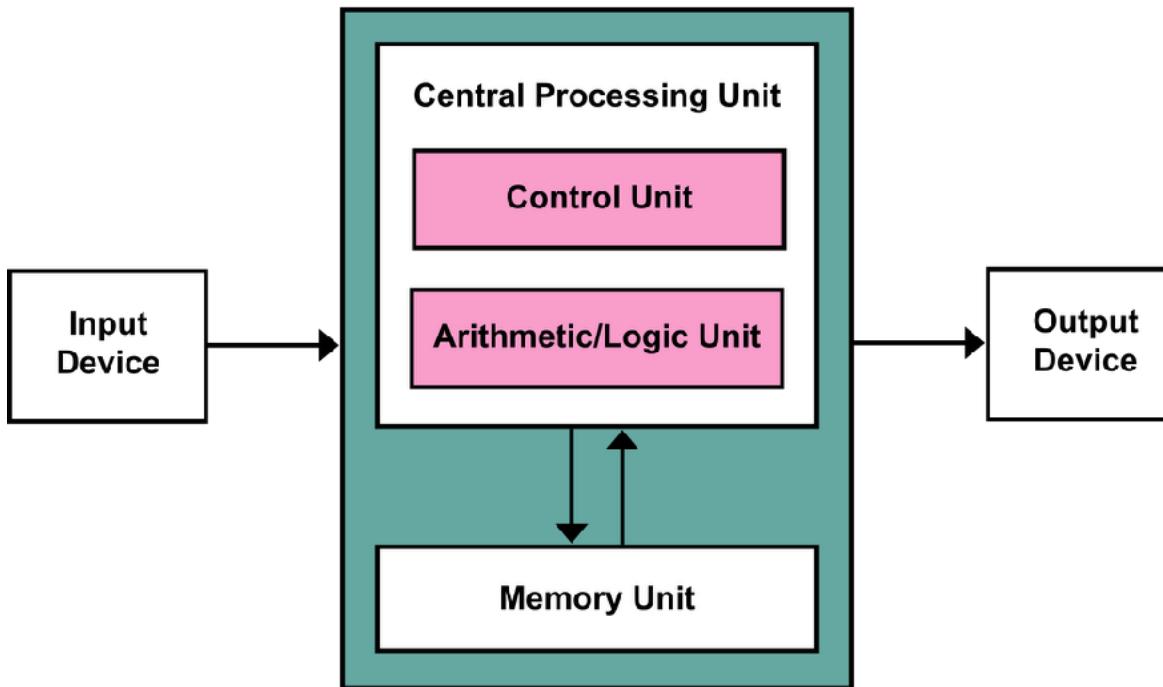
- Cette partie correspond à la partie “Introduction” de la Théorie
- <https://sites.uclouvain.be/SystInfo/notes/Theorie/intro.html>
- Illustrations supplémentaires en cours : à reproduire dans vos notes !

# Système informatique : fondamentaux

- Composants
  - Un (des) processeur(s) — CPU en anglais
  - Mémoire principale
  - Dispositifs d'entrée/sortie (y.c. de stockage)
- Fonctionnement d'un processeur
  - Exécution d'*instructions* câblées en matériel
    - Lire / écrire en mémoire vers / depuis des registres
    - Opérations (calculs, comparaisons) sur ces registres
- Un processeur est défini par son *jeu d'instructions*
  - x86\_64 (PC, anciens Mac) ;  
ARM A64 (Raspberry PI, iPhone, nouveaux Mac M1/M2)

# Architecture de von Neumann

- Mémoire principale utilisée pour stocker les instructions ET les données



By Kapooh - Own work, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=25789639>

# Représentation des données

- Information manipulée sous forme binaire (1 et 0) — bits
  - 4 bits : *nibble*, (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F) en hexadécimal
  - 8 bits : *octet* ; 32 bits : *mot* ; 64 bits : *long mot*
- Instructions stockées sous forme binaire
  - Premiers bits codent identifiant de l'instruction à exécuter
  - Suivi des arguments ou *opérandes*
- Processus “Fetch/Decode/Execute”
  - Compteur de programme : registre incrémenté pour savoir quelle instruction exécuter à la suite de l'instruction courante
  - Des instructions de contrôle peuvent positionner le compteur de programme selon le résultat d'un test : permet `while`, `if`, etc.

# Fonctionnement d'un système informatique

- Opérations d'entrée/sortie se déroulent de manière concurrente (*en même temps*) que l'exécution d'instructions par le processeur
- Contrôleurs de périphériques spécifiques à chaque type de périphérique (clavier, souris, écran, etc.)
- Chaque contrôleur possède une mémoire dédiée (un *buffer*)
- Le processeur doit déplacer des données depuis/vers la mémoire principale depuis/vers ces buffers dédiés
- Opérations d'entrée/sortie entre le contrôleur et le périphérique

# Fonctionnement d'un système informatique

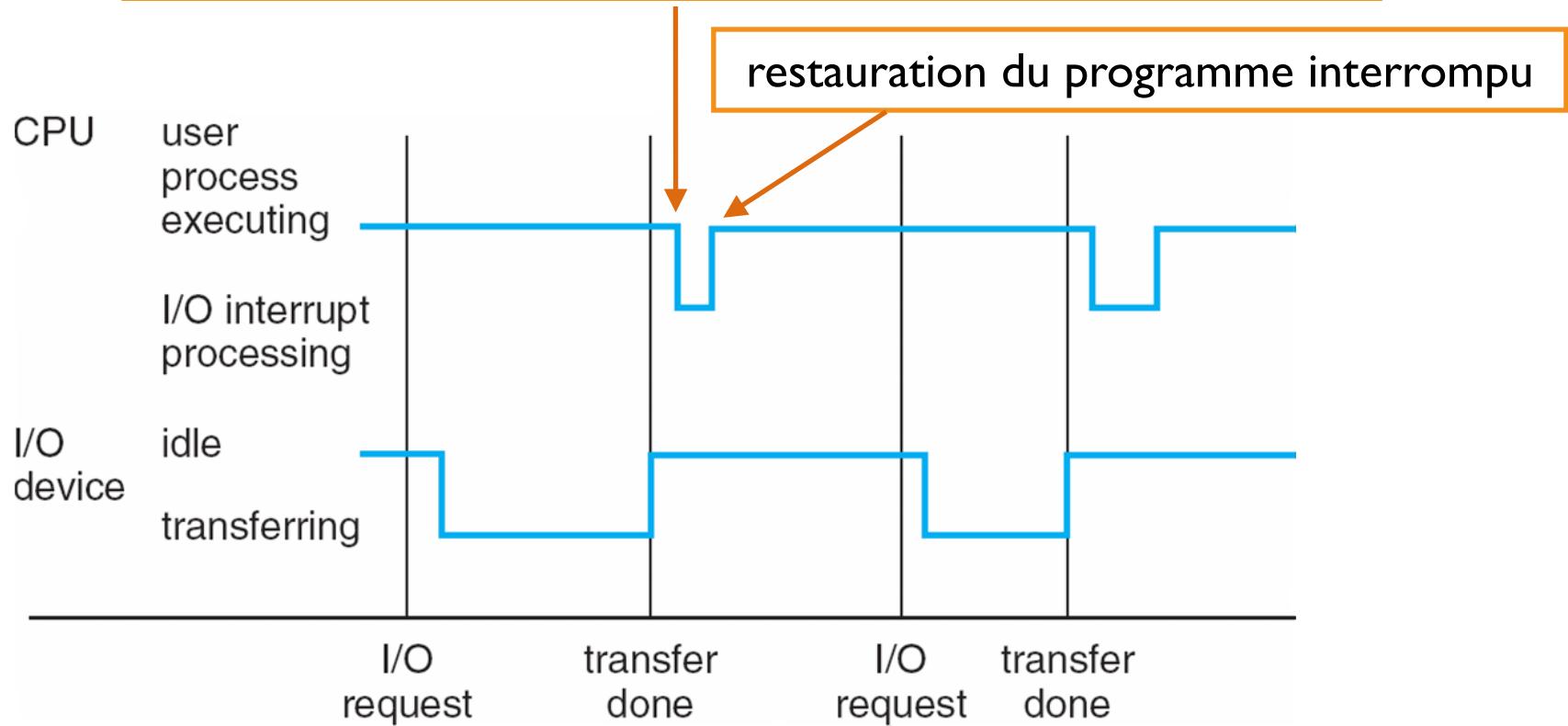
- Par défaut, le processeur suit un ‘fil’ continu d’instructions
  - Comment réagir à des stimuli externes ?
    - Frappe du clavier, mouvement de la souris, paquet réseau
  - → par exemple, récupération de l’identifiant de la touche depuis la mémoire du contrôleur du clavier
- Le contrôleur de périphérique annonce au processeur la fin d’une opération d’entrée/sortie en générant une ***interruption***
  - Signal électrique à destination du processeur

# Traitement d'une interruption

- Le processeur interrompt le fil d'exécution d'instructions courant et transfert le contrôle du processeur à une routine de traitement
  - L'étape (compteur de programme) à laquelle le fil d'exécution est interrompu est sauvegardée, ainsi que l'état (les registres) du processeur
- La routine de traitement détermine la source de l'interruption ...
  - Polling : interroge tous les contrôleurs un à un, ou
  - Vectored : plusieurs lignes d'interruption différentes (poss. partagées)
- ... puis positionne le compteur de programme à la première instruction du segment de code associé à cette source d'interruption
- À la fin du traitement, on restaure l'état du processeur et on reprend le processus interrompu en restaurant le compteur de programme

# Opérations d'entrée/sortie

traitement de l'interruption : récupération de l'identifiant de la touche depuis le *buffer* du contrôleur de périphérique

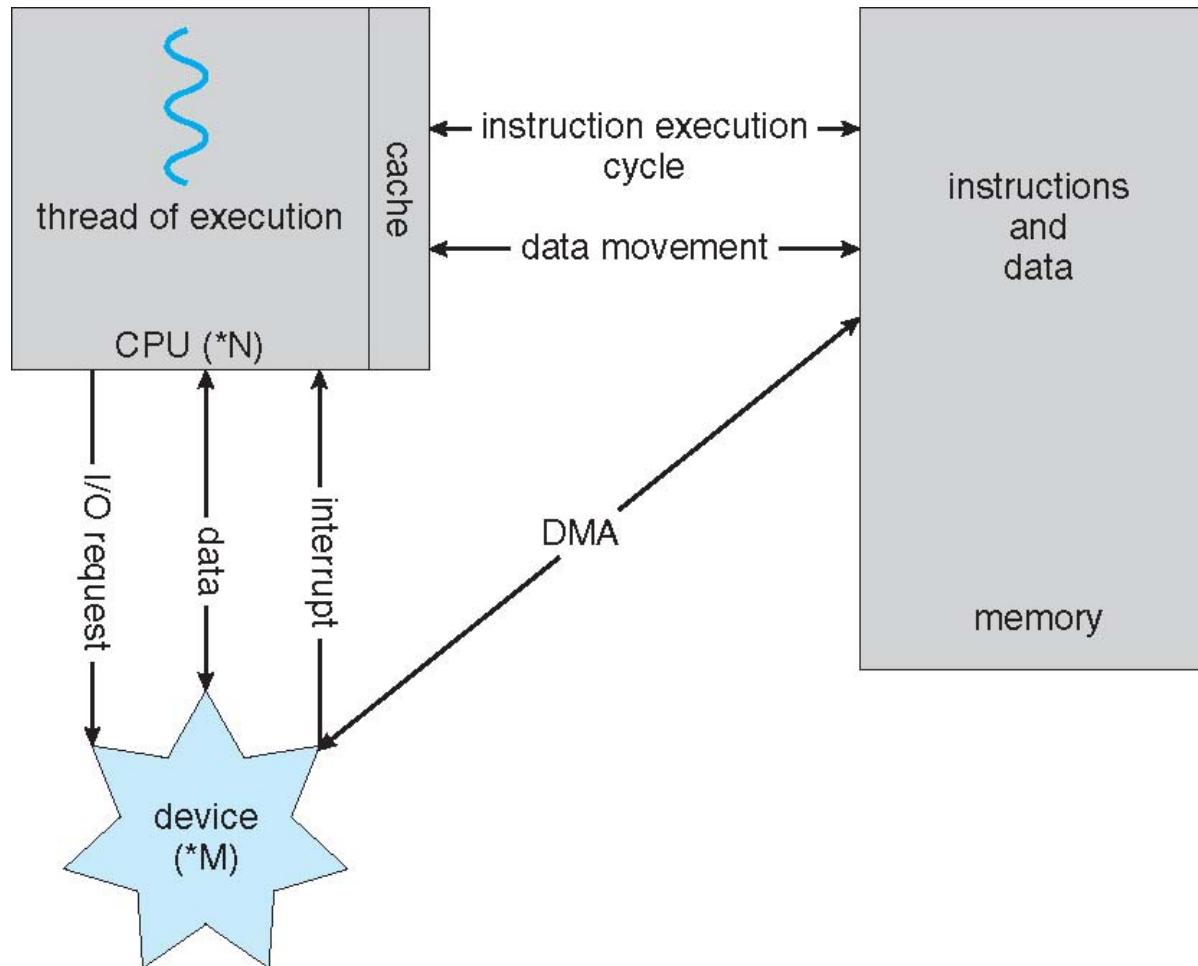


credit: Adam Silberschatz

# Accès direct à la mémoire

- Une interruption pour chaque appui d'une touche de clavier ?
  - toutes les ~150-200 millisecondes : OK ! 😎
- Une interruption pour chaque octet lu depuis un disque dur ?
  - 125 Mo (méga-octets =  $10^6$  octets) par seconde
  - $125 * 10^6 = 125$  millions d'interruptions par seconde
  - Temps moyen entre deux interruptions = 8 ns ...
  - Avec un processeur à 3 GHz, 24 'tics' d'horloge entre deux interruptions — insuffisant pour traiter chaque interruption ! 😱
- ✓ Accès direct à la mémoire (DMA) : permet le transfert direct entre le contrôleur de périphérique et la mémoire principale
  - Une interruption générée pour signaler la fin du transfert d'un *bloc*

# Système informatique complet



credit: Adam Silberschatz

# Le rôle du système d'exploitation

- Programmer directement au dessus du matériel, gérer les interruptions, les divers gestionnaires de périphériques, etc. ? 😱
  - Système d'exploitation = intermédiaire entre matériel et applications
- 3 rôles principaux
  - ✓ Rendre l'utilisation et le développement d'applications plus simple et plus universel (portable d'une machine à une autre)
  - ✓ Permettre une utilisation plus efficace des ressources
  - ✓ Assurer l'intégrité des données et des programmes entre eux (e.g., un programme crash mais pas le système)

# Virtualisation

- Le système d'exploitation assure ces rôles en *virtualisant* les ressources matérielles
  - Malgré leur hétérogénéité
  - Sur des systèmes équipés de ressources différentes (en nature et en volume)
- Représentations abstraites des ressources
  - Utilisation à travers d'API
  - Universelles (même entre différents SE)

# Exemple de virtualisation (I)

- Virtualisation du processeur : **processus**
- Illusion pour le programmeur d'avoir un processeur dédié pour son application
  - exécution d'une suite d'instructions sans interruption
- Plusieurs processus co-existent au même moment et au sein d'un même système
- Le SE partage la ressource processeur entre les processus en utilisant le *partage de temps*
- Mécanismes support pour, par exemple, mettre en suspens un processus qui attendrait une entrée clavier puis le “réveiller” par la suite

# Exemple de virtualisation (2)

- Virtualisation de la mémoire
  - Plusieurs processus en mémoire : tous utilisent la mémoire physique présente sur la machine
  - Comment s'assurer que les processus ne puissent pas lire/écrire dans l'espace mémoire des autres processus (propriété d'isolation) ?
- Mémoire virtuelle : fournit l'illusion que chaque processus dispose de son espace mémoire propre, de grande taille et de structure fixe
  - Par défaut, pas de visibilité de la mémoire des *autres* processus
  - Le système d'exploitation gère la correspondance entre des *adresses virtuelles* utilisées par chaque programme et des *adresses physiques* en mémoire principale

# La mise en œuvre d'un SE : compromis abstraction/coût

- Des abstractions de plus haut niveau facilitent la vie du programmeur et de l'utilisateur
  - Exemple de la mémoire virtuelle ;
  - Pourtant, le principe de mémoire virtuelle semblait irréaliste encore longtemps après son invention (fin 1950s)
- Rapport coût / bénéfice
  - ☹ Sur le modèle de processeur (naïf) que nous avons utilisé, chaque opération d'accès mémoire devrait être transformé en plusieurs instructions pour mettre en œuvre la translation adresse virtuelle-adresse physique !
  - 😊 L'adoption de la mémoire virtuelle à coût raisonnable a été possible grâce à l'ajout de fonctionnalités aux processeurs, permettant cette translation au niveau matériel

# Séparation entre mécanisme et politique

- Principe important de mise en œuvre des SE
- Exemple : la virtualisation du processeur via les processus
  - I. Un mécanisme permet le partage de temps
    - changement de contexte : sauvegarde de l'état du processeur (compteur de programme, registres) et restauration ultérieure
  - 2. Une *politique* arbitre entre les processus pouvant s'exécuter et le(s) processeur(s) disponibles
    - politique d'ordonnancement (scheduling)
- On peut définir des politiques d'ordonnancement différentes selon les contextes, mais sur la base du même mécanisme

# Interactions entre les applications et le SE

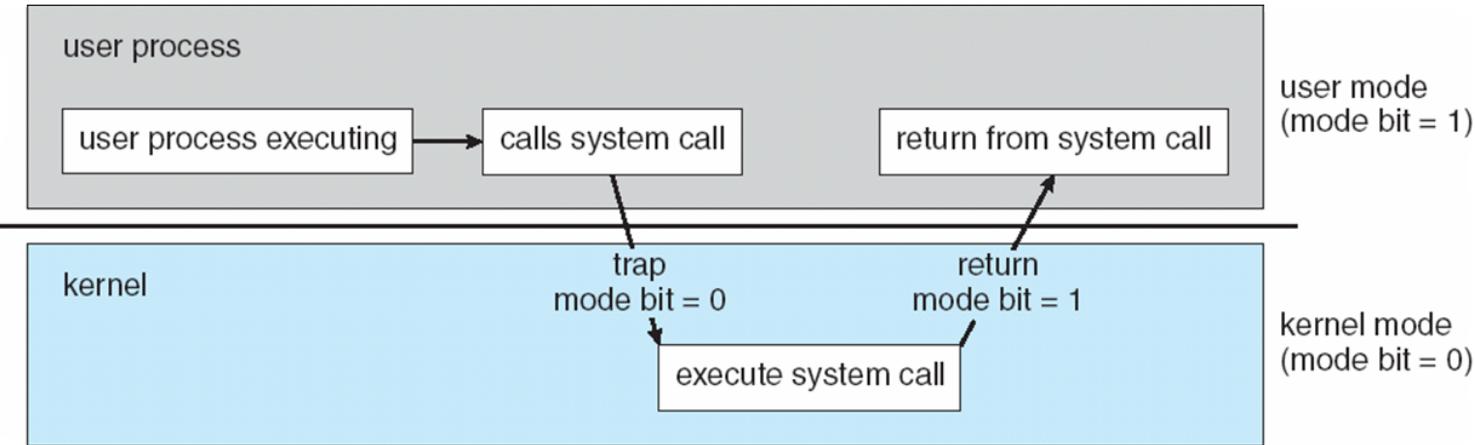
- Fonctionnalités du SE disponibles via une API
- Les fonctions du SE nécessitent un accès global au système, par exemple
  - Accès à toute la mémoire
  - Configuration des gestionnaires de périphériques
  - Configuration des interruptions
- Pour mettre en œuvre le rôle d'isolation, les processus clients (programmes utilisateurs) ne doivent pas avoir accès à tout cela !

# Modes d'exécution

- mode utilisateur : programme utilisant les abstractions fournies par le SE ; certaines instructions sont interdites
  - accès mémoire en dehors d'une zone autorisée — le fameux “segmentation fault” !
  - de manière générale, toutes les instructions permettant de changer la configuration matérielle du système, comme la configuration ou la désactivation des interruptions
- mode protégé : utilisé par le *noyau* du SE, toutes les instructions sont autorisées
- L'utilisation de fonctionnalités du SE par un processus utilisateur nécessite de passer d'un mode à l'autre
  - Utilisation d'un **appel système**

# Appel système == interruption

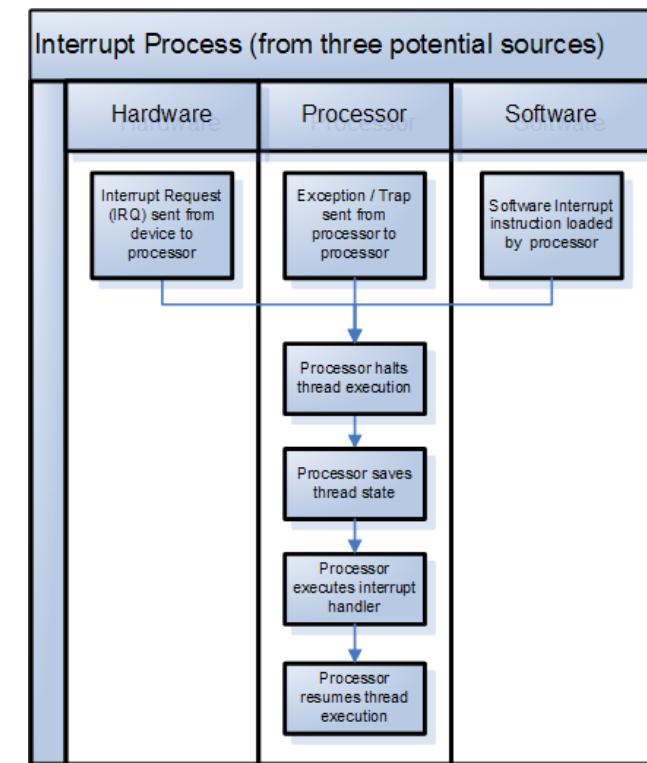
- Un appel système permet à un processus utilisateur d'invoquer une fonctionnalité du SE
- Génération d'une interruption logicielle (*trap*)
  - à l'aide d'une instruction processeur spécifique
- Le processeur interrompt le processus, passe en mode protégé, et branche vers le point d'entrée unique du noyau



credit: Adam Silberschatz

# En résumé ...

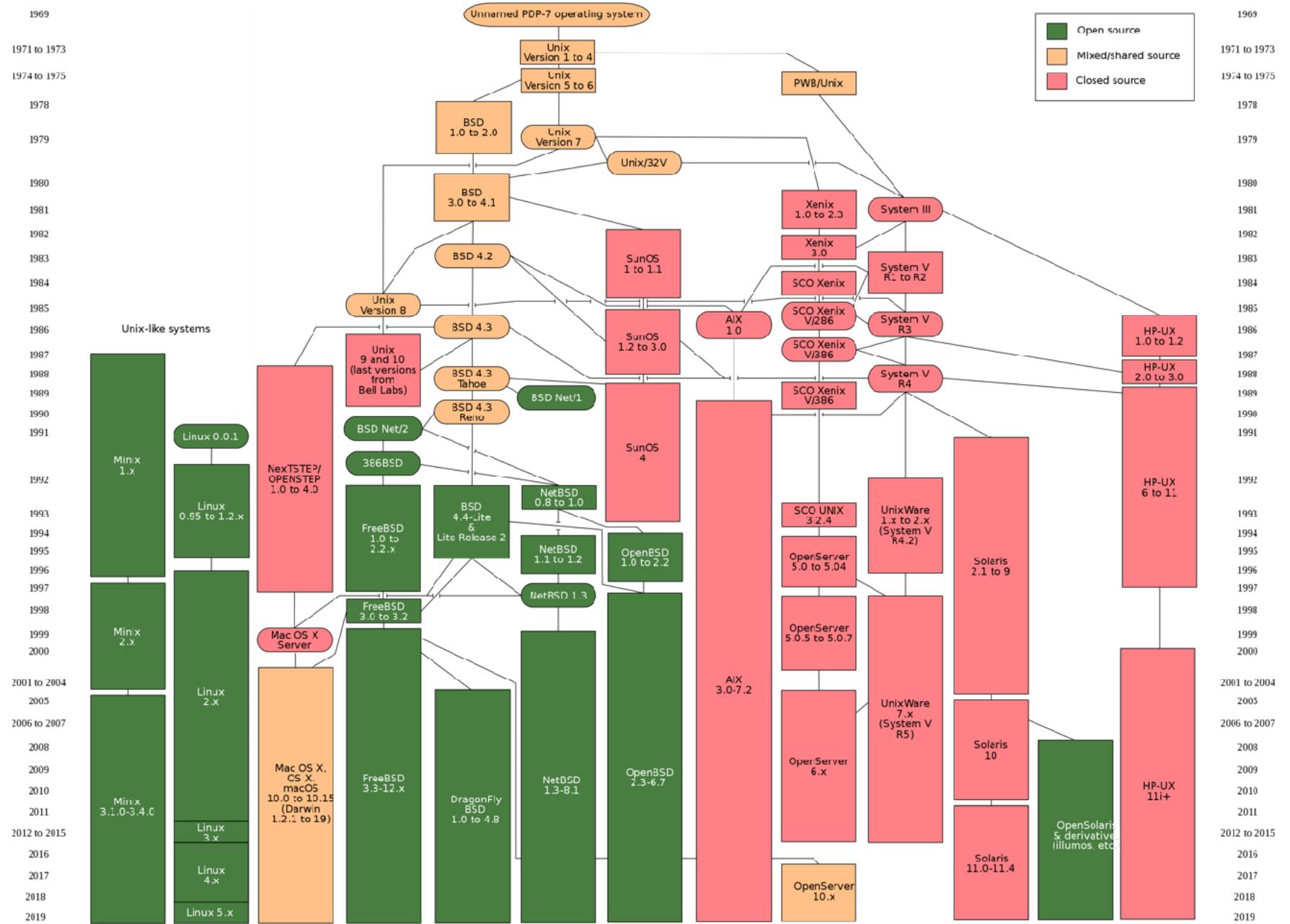
- Un système d'exploitation peut être décrit comme un système de traitement d'interruptions
  - Interruptions matérielles
    - ➡ Gestion des entrées/sorties
  - Interruptions du processeur (instructions illégales en mode utilisateur)
    - ➡ Isolation des erreurs
  - Interruptions logicielles
    - ➡ Service d'appels systèmes



credit: Stephen Charles Thompson, CC BY-SA 3.0

# UNIX

- Famille de systèmes d'exploitation
  - De nombreuses déclinaisons !
  - Principes communs : KISS, séparation mécanisme/politique, abstractions proposées
- Dans ce cours, nous nous focaliserons sur le système GNU/Linux
  - Linux : le noyau
  - GNU : la collection d'utilitaires et de bibliothèques associés
- Les autres systèmes d'exploitation suivent largement les principes introduits par UNIX



credit:Wikimedia, CC BY-SA 3.0

# Partie 2 : Utilisation de la ligne de commande

# Syllabus

- Cette partie correspond à la partie “Utilisation d'un système Unix” de la Théorie
- <https://sites.uclouvain.be/SystInfo/notes/Theorie/shell/shell.html>
- Pour aller plus loin : Section “Shell” de la section “Outils” (<https://sites.uclouvain.be/SystInfo/notes/Outils/>)

# Utilitaires UNIX

- Origine d'UNIX : Bell Labs, 1970
  - Mini-ordinateurs : plusieurs utilisateurs via terminaux
  - Manipulation et traitement de documents texte
- “Philosophie” UNIX : KISS
  - “Keep It Simple, Stupid”
  - Programme simples, petits, parfaitement adaptés à une tâche ou fonction unique
  - La force de ce principe est dans la facilité de *composition* de commandes pour réaliser des opérations plus complexes

# Quelques utilitaires standard

Utilitaire	Fonction
<b>cat</b>	lire/afficher le contenu d'un fichier ex : cat fichier.txt
<b>echo</b>	afficher une chaîne de caractères passée en argument, ex. : echo "Bonjour Monde"
<b>head / tail</b>	affiche le début resp. la fin d'un fichier ex. : tail errors.log
<b>wc</b>	compte le nombre de caractères / de lignes d'un fichier. ex. : wc -l students.dat
<b>sort</b>	trie un fichier. ex. : sort -n -r scores.txt
<b>uniq</b>	extrait les lignes uniques ou dupliquées d'un fichier <b>trié</b> fourni en argument. ex. : uniq -d students.dat

# La documentation

- Chaque utilitaire dispose d'une *page de manuel* spécifiant
  - sa fonction
  - les paramètres attendus en entrée
  - les *options* permettant de modifier son comportement
- Accessible en utilisant l'utilitaire `man`
  - ... documentation accessible avec `man man`

# Exemple de documentation (man uniq)

```
uniq -d students.dat
commande option(s) paramètre(s)
```

## UNIQ

Section: User Commands (1)

Updated: April 2010

### NAME

**uniq** - report or omit repeated lines

### SYNOPSIS

**uniq** [*OPTION*]... [*INPUT* [*OUTPUT*]]

### DESCRIPTION

Filter adjacent matching lines from *INPUT* (or standard input), writing to *OUTPUT* (or standard output).

With no options, matching lines are merged to the first occurrence.

Mandatory arguments to long options are mandatory for short options too.

#### **-c, --count**

prefix lines by the number of occurrences

#### **-d, --repeated**

only print duplicate lines

#### **-D, --all-repeated[=*delimit-method*]**

print all duplicate lines *delimit-method*=*{none(default),prepend,separate}*

Delimiting is done with blank lines

#### **-f, --skip-fields=N**

avoid comparing the first N fields

#### **-i, --ignore-case**

ignore differences in case when comparing

### syntaxe

### option

### Systèmes Informatiques — E. Rivière

#### **-s, --skip-chars=N**

avoid comparing the first N characters

#### **-u, --unique**

only print unique lines

#### **-z, --zero-terminated**

end lines with 0 byte, not newline

#### **-w, --check-chars=N**

compare no more than N characters in lines

#### **--help**

display this help and exit

#### **--version**

output version information and exit

A field is a run of blanks (usually spaces and/or TABs), then non-blank characters. Fields are skipped before chars.

Note: 'uniq' does not detect repeated lines unless they are adjacent. You may want to sort the input first, or use `sort -u` without `uniq`. Also, comparisons honor the rules specified by `LC\_COLLATE`.

### description

# Sections de manuel

- Section 1: Utilitaires disponibles pour tous les utilisateurs
  - Section 2: Appels systèmes en C
  - Section 3: Fonctions de la librairie
  - Section 4: Fichiers spéciaux
  - Section 5: Formats de fichiers et conventions pour certains types de fichiers
  - Section 6: Jeux
  - Section 7: Utilitaires de manipulation de fichiers textes
  - Section 8: Commandes et procédure de gestion du système
- 
- **8 sections (raisons historiques/ arbitraires)**
    - ambiguïtés possibles !
      - `printf` l'utilitaire ou  
`printf` la fonction de la librairie standard C ?
    - Spécifier la section explicitement :  
man 1 `printf` **vs** man 3 `printf`

# Shell / interpréteur de commande

- Interaction avec le système d'exploitation
  - Différents shells possibles : bcsh, bash, zsh, etc.
    - bash est le plus répandu
  - Exécution d'utilitaires/programmes et de commandes internes
    - e.g. cd permet de changer le répertoire courant
  - Complémentaire avec une interface graphique !
- Permet de combiner les opérations
  - Principe KISS !



# Flux standards et redirections



- 3 flux standards (1 entrée, 2 sorties)
- Redirections permettent de combiner des commandes en utilisant des fichiers intermédiaires

< file	redirige le contenu de file vers STDIN
> file	redirige STDOUT vers file (si le fichier n'existe pas, il est créé, si il existe déjà son contenu est écrasé)
>> file	redirige STDOUT vers file (si le fichier n'existe pas, il est créé, si il existe déjà le contenu est ajouté à la suite du fichier)
2>&1	redirige STDERR vers STDOUT

# Pipes

- Redirection directe de STDOUT d'une commande vers STDIN d'une autre commande
  - Sans utiliser de fichier intermédiaire !
- Avec le symbole | (barre horizontale)
  - cmd1 | cmd2

# Exemples de redirections

```
$ echo "Un petit texte" | wc -c
      15
$ echo "bbbb ccc" >> file.txt
$ echo "aaaaaa bbbbb" >> file.txt
$ echo "bbbb ccc" >> file.txt
$ cat file.txt
bbbb ccc
aaaaaa bbbbb
bbbb ccc
$ cat file.txt | sort | uniq
aaaaaa bbbbb
bbbb ccc
```

# Scripts

- Un système UNIX peut exécuter
  - Programmes compilés en langage machine
  - Langages interprétés (bash, perl, python, etc.)

```
#!/bin/bash
echo "Hello, world"
```

Par convention, les deux premiers octets du fichier `#!` indiquent qu'il s'agit d'un programme interprété; l'interpréteur à utiliser suit

```
#!/usr/bin/python
```

```
#!/usr/bin/env python
```

# Variables

```
#!/bin/bash  
PROG="LINFO"  
COURS=1252  
# concaténation  
echo $PROG$COURS
```

# : commentaire

accès à la valeur de la variable avec \$



une variable non déclarée vaut “” (chaîne vide)

⚠ ambiguïté possible ?

milieu = "mi"; echo do\$milieuno

utiliser des { } pour délimiter le nom de la variable :

echo do\${milieu}no

# Arguments de la ligne de commande

```
#!/bin/bash
# $# nombre d'arguments
# $1 $2 $3 ... arguments
echo "Vous avez passé" $# "arguments"
echo "Le premier argument est :" $1
echo "Liste des arguments :" $@
```

# Conditionnelles (if)

⚠️ espace important (source de bug commune)

```
#!/bin/bash
# Vérifie si les deux nombres passés en arguments sont égaux
if [ $# -ne 2 ]; then
    echo "Erreur, deux arguments sont nécessaires" > /dev/stderr
    exit 2
fi
if [ $1 -eq $2 ]; then
    echo "Nombres égaux"
else
    echo "Nombres différents"
fi
exit 0
```

⚠️ ; ou passage à la ligne (pas les 2)

`$i -eq $j` est vraie lorsque les deux variables `$i` et `$j` contiennent le même nombre.

`$i -lt $j` est vraie lorsque la valeur de la variable `$i` est numériquement strictement inférieure à celle de la variable `$j`

`$i -ge $j` est vraie lorsque la valeur de la variable `$i` est numériquement supérieure ou égale à celle de la variable `$j`

`$s = $t` est vraie lorsque la chaîne de caractères contenue dans la variable `$s` est égale à celle qui est contenue dans la variable `$t`

`-z $s` est vraie lorsque la chaîne de caractères contenue dans la variable `$s` est vide

# Valeur de retour, fichier spéciaux

```

#!/bin/bash
# wordin.sh
# Vérifie si le mot passé en premier argument est présent
# dans le fichier passé comme second argument
if [ $# -ne 2 ]; then
    echo "Erreur, deux arguments sont nécessaires" > /dev/stderr
    exit 2
fi
grep $1 $2 >/dev/null
# $? contient la valeur de retour de grep
if [ $? -eq 0 ]; then
    echo "Présent"
    exit 0
else
    echo "Absent"
    exit 1
fi

```

- `$?` contient la valeur de retour du dernier programme ( $=0 \rightarrow \text{OK}$ )
- `/dev/null` : “trou noir” (à quoi sert `/dev/random`?)

# Conditions sur les fichiers

```

#!/bin/bash
# vérifie si le fichier fourni en entrée contient des données
if [ $# -ne 1 ]; then
    echo "Erreur, un seul argument est nécessaire" > /dev/stderr
    exit 2
fi
if [ -d $1 ]; then
    echo "Erreur, $1 est un répertoire, pas un fichier" > /dev/stderr
    exit 2
fi
if [ ! -f $1 ]; then
    echo "Erreur, $1 n'existe pas" > /dev/stderr
    exit 2
fi
if [ -s $1 ]; then
    echo "Le fichier $1 contient des données" > /dev/stdout
else
    echo "Le fichier $1 ne contient pas de données" > /dev/stdout
fi

```

## Combinaisons logiques

```

if [ -d $1 -o ! -f $1 ]; then
    echo "Erreur, $1 n'existe pas ou est un répertoire" > /dev/stderr
    exit 2
fi

```

... aussi -a (AND)

# Boucles : for

```
#!/bin/bash
# exemple_for.sh
students="Julie Maxime Hakim"
for s in $students; do
    l=`wc -l TP1-$s.txt | cut -d' ' -f1`
    echo "Bonjour $s, ton compte rendu de TP comporte $l lignes."
done
```

- **s** prend successivement les valeurs présentes dans la liste d'entrée **\$students**
- `command` permet d'assigner la sortie d'une commande à une variable
  - que fait la commande composée ci-dessus ?
- boucles `while` et `until` : principe similaire, avec une condition d'arrêt booléenne

# Exercices

- Cette semaine, exercices sur inginious
  - Utilisation de commandes standards
  - Utilisation de la documentation
  - Deux “capture the flag” interactifs
  - Écriture d'un script bash selon un cahier des charges
- À valider pour la fin de S3  
(dimanche 8/10/23 23:59)

# Conclusion

# Conclusion

- Système informatique
  - Processeur et mémoire
  - Périphériques d'entrée/sortie (y. c. stockage)
  - Système d'exploitation (y.c. utilitaires)
  - Applications
- Principe des interruptions matérielles et logicielles permettant la mise en œuvre du système d'exploitation et l'isolation d'avec et entre les applications
- Interface en ligne de commande et principes de composition de programmes / scripts

# À la suite ...

- La semaine prochaine : “Révisions” de programmation en langage C
  - Et concepts avancés peu ou non vus en LEPLI503
  - Pour les étudiants n’ayant pas suivi LEPLI503 ou se sentant peu à l’aise en C
    - Syllabus commun détaillé à (re)lire
    - Exercices optionnels sur Inginious (enregistrez vous à LEPLI503 : <https://inginious.info.ucl.ac.be/course/LEPLI503>)
- Séance tutorée de démarrage (vendredi ou mercredi)
  - Prise en main et mise en place
  - Exercices Inginious
  - Aide si nécessaire à l’installation d’une machine virtuelle ou d’un système Linux en *dual boot* — faites votre sauvegarde avant !