Programmation Système

Cours 1: Introduction

Khaddouja ZELLAMA
Khaddouja.zellama@dauphine.psl.eu

Licence L2 Mathématiques et Informatique
Département MIDO

Planning cours

- Introduction
- Système de fichier UNIX
- Processus
- Programmation shell
- Appels système
- Threads

Systèmes d'exploitation (OS)

- Les programmes qui s'exécutent sur un ordinateur
  - Légitimes
  - Peu fiables
    Utilisation de ressources trop importantes / infini
  - Mal-intentionnés
    Piratage de données, ouvertures d'accès à des utilisateurs non autorisés
  - Avoir besoin de communiquer/collaborer
  - En compétition les uns avec les autres pour l'accès aux ressources

- L'OS s'assure du bon déroulement de l'exécution des programmes en contrôlant l'accès aux principales ressources :
  - CPU (i.e. temps de calcul), mémoire
  - Périphériques, canaux de communication
  - Données utilisateurs

Objectifs pédagogiques de ce cours

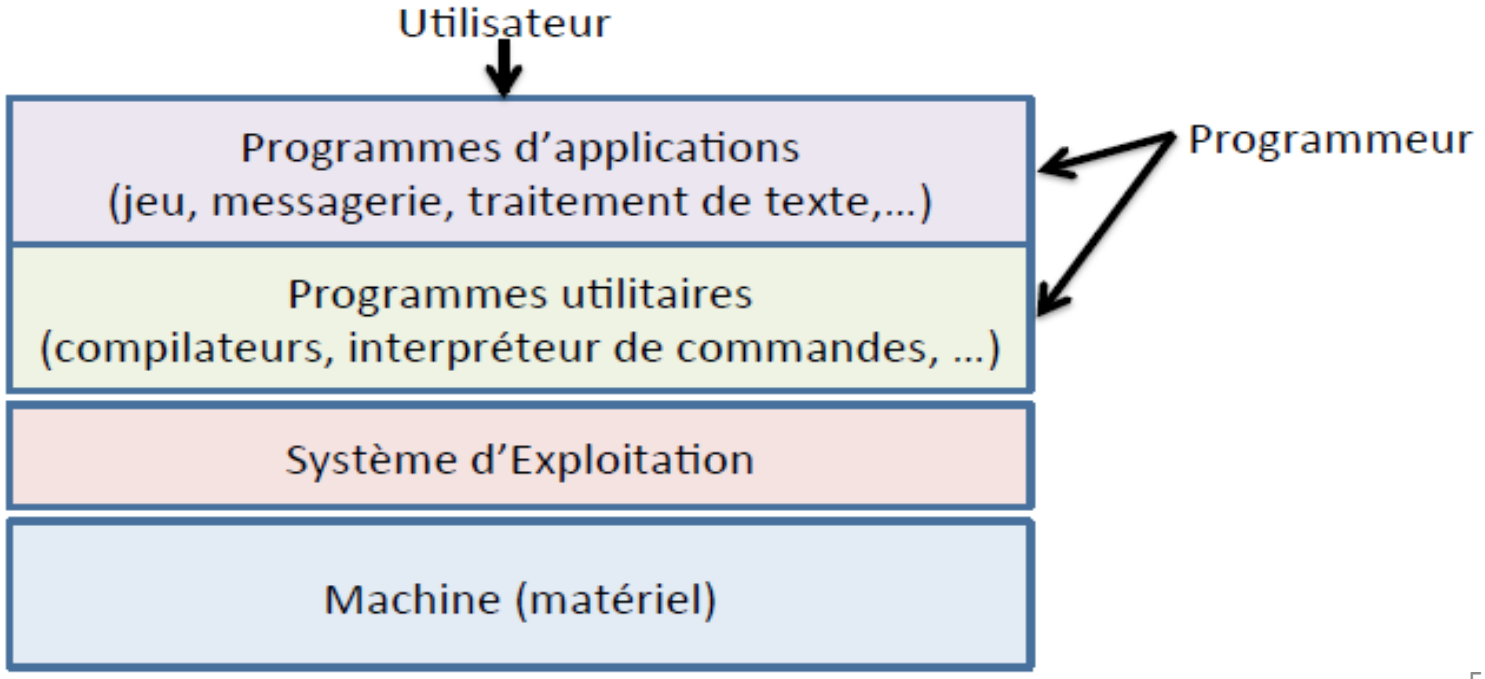
- Familiariser avec l'utilisation de l'environnement Unix

- Donner une perspective de développeur plutôt que d'utilisateur/consommateur

Planning:
10 cours (1h30), 12 séances TP/TDs (1h30).

Fonctions d'un OS

- Deux catégories de programmes :
  - Les programmes utilitaires ou systèmes (éditeurs, compilateurs,...) destinés aux programmeurs
  - Les programmes d'applications (tableurs, traitement de texte,...) destinés aux utilisateurs finaux



Exécution d'un programme simple

Instructions:

- ADD: Additionner le contenu de deux registres
- LOAD: Charger une valeur de la RAM vers un registre
- STORE: Stocker une valeur dans un registre vers la RAM

Programme: séquence d'instructions + des données

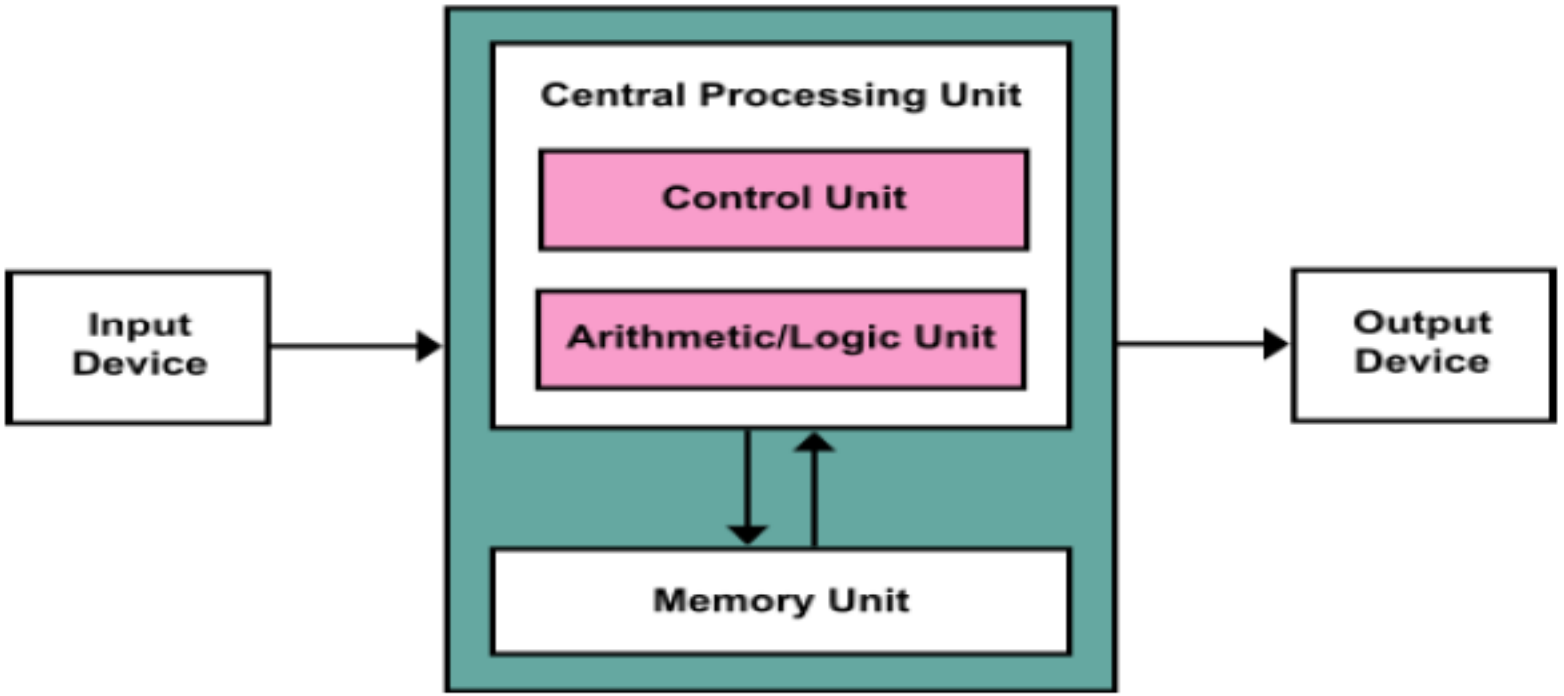
```
1 // Programme en C qui additionne deux nombres et qui affiche de résultat
2 int a = 15;
3 int b = 17;
4 int c = a + b;
5 print("%d", c);
```

```
1 // Sequence d instructions CPU correspondante (après compilation)
2   1: LOAD @6,R1
3   2: LOAD @7,R2
4   3: ADD R1,R2
5   4: STORE R1 @7
6   5: PRINT @7
7   6: 32
8
```

Un CPU (1 coeur) n'exécute qu'un seul programme à la fois

Architecture de Von Neumann (1945)

► Inventée par John von Neumann (1903-1957)

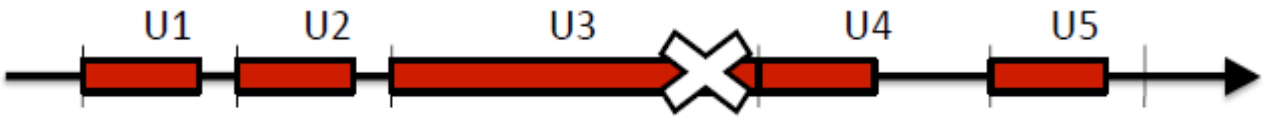


Composants principaux:

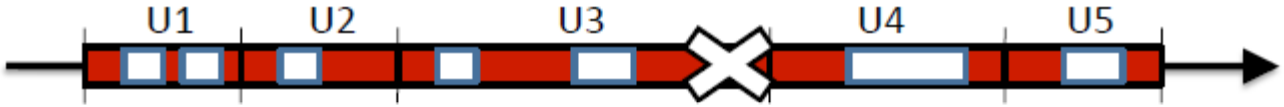
- CPU, capable d'effecteur des opérations (instructions assembleur)
- Mémoire RAM, capable de stocker des variables
- Entrées / Sorties, capables d'interagir avec l'utilisateur

Historique

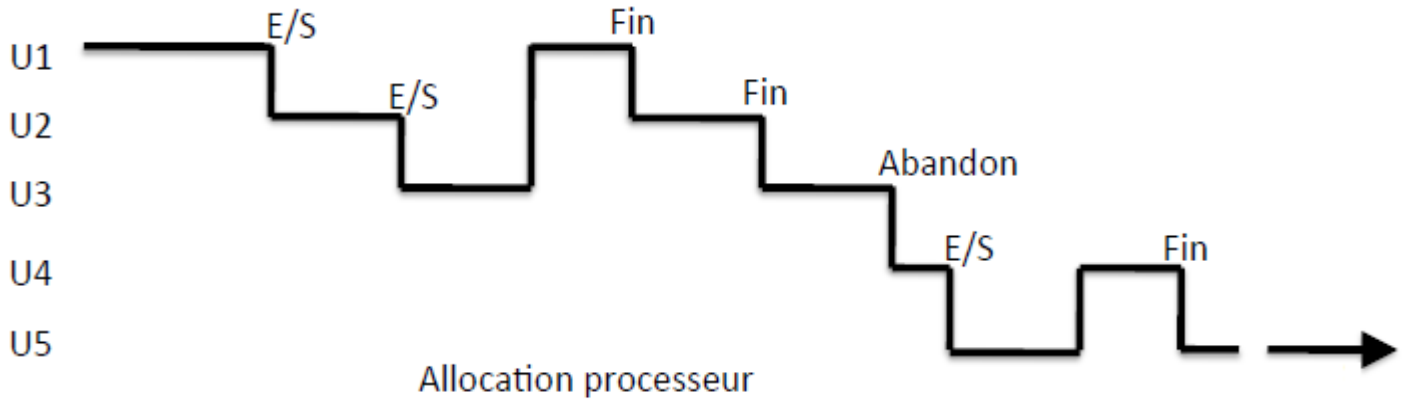
• 1940 - 1950: Un seul utilisateur par ordinateur



• 1960: traitement par lots: cartes Fortran en entrée et cartes en langage assembleur en sortie.

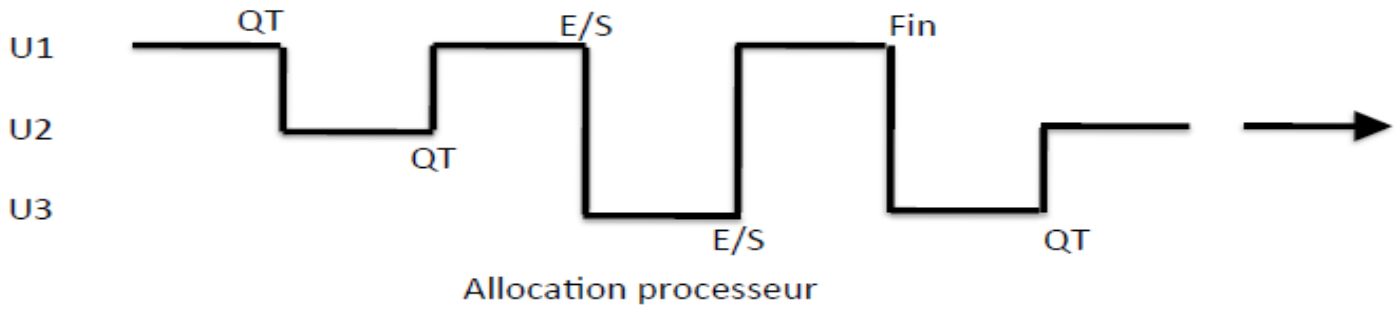


• 1965 - 1980: multiprogrammation – Traitements par lots parallèle: exécutions parallèles d'un ensemble de programmes.



Historique

• 1980 - : temps partagé – Les programmes n'ont plus de temps d'exécution Maximum (quanta)



UNIX

▶ Nécessité d'un **système** capable de gérer plusieurs programmes et plusieurs utilisateurs

1965 – **MULTICS:** Un premier système d'exploitation avec

- Multi-utilisateurs
- Partage du temps CPU (*time sharing*) et des autres ressources

1969 – **UNICS** puis **UNIX:**

- Par *Ken Thompson* et *Dennis Ritchie*
- En assembleur pour PDP-7, puis PDP-11/45

1973 – Réécriture de UNIX dans un langage de haut niveau

- Invention du langage C par *Dennis Ritchie*

Historique UNIX

Unix est une famille de système d'exploitations.

- Dérivés de Unics (premier système d'exploitation)
- Très nombreux forks

Quelques Unix (ou Unix-like)

- BSD (Berkeley Software Distribution) descendant de UNIX
- Free BSD version libre et gratuite de BSD
- GNU/Linux version libre et gratuite dérivée de BSD
- ...

Dérivé d'UNIX

- Android dérivé de Linux
- OSX (OS des Mac) dérivé de FreeBSD
- IOS dérivé de OSX
- Orbis OS (OS de la PS4), dérivé de Free BSD
- Freebox OS, dérivé de Linux

UNIX vs. Windows

- Windows: l'utilisateur n'est pas un programmeur
  ▶ Des programmeurs professionnels développent des applications pour les utilisateurs.

- Unix: utiliser un ordinateur c'est le programmer pour pouvoir automatiser des tâches
  ▶ Tout utilisateur doit pouvoir facilement programmer lorsqu'il en a besoin.

▶ UNIX offre des avantages aux utilisateurs compétents.

Comment faciliter la programmation/l'automatisation de tâches répétitives?

Philosophie UNIX

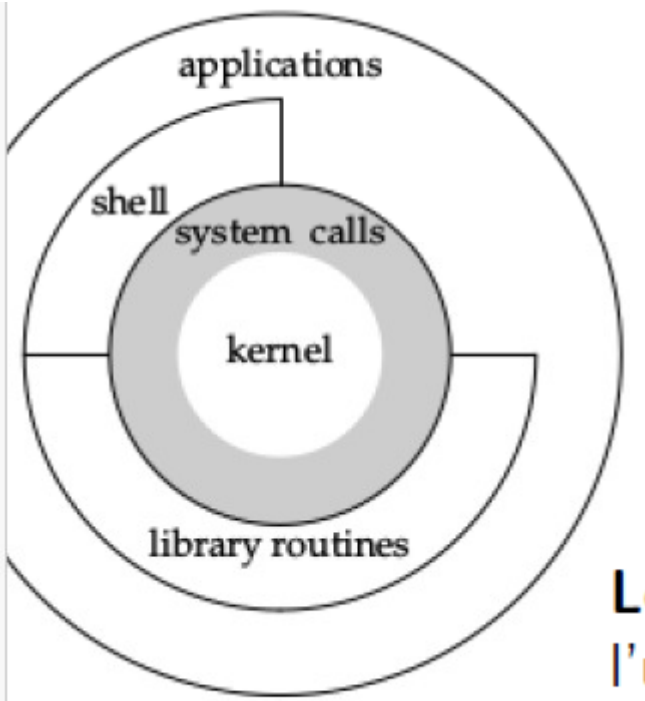
1. Écrire des programmes qui font qu'une chose mais qui le font bien
2. Écrire des programmes qui collaborent

Exemple avec le langage shell:

- La commande `ls` liste le contenu d'un répertoire (un fichier par ligne)
- La commande `wc -l` compte le nombre de ligne dans un fichier
- La commande `ls | wc -l` compte le nombre de fichiers dans un repertoire

Architecture UNIX

Le noyau: contient les composants logiciels qui permettent de gérer l'accès au matériel



l'API système: permet aux applications de communiquer le matériel via une API Elle permet d(e)

- unifier l'accès aux différent matériels
- contrôler l'exécution d'opérations critiques pour éviter les applications buggées/malveillantes de tout faire planter

Le shell, et les applications: fournissent à l'utilisateur des fonctionnalités de haut niveau

age: https://github.com/congdv

Utilisateurs UNIX

- Les utilisateurs sous Unix détiennent des ressources (fichiers, processus, etc.)
- Les utilisateurs sont identifié par un nom d'utilisateur (login)
- Différents utilisateurs peuvent avoir différents privilèges
- L'utilisateur `root` à tous les privilèges

Lorsqu'un programme s'exécute, **il hérite de tous les privilèges de l'utilisateur qui l'a lancé**, et peut accéder à toutes les ressources détenues par l'utilisateur.

⚠ Ne pas exécuter des programmes d'origine incertaine
⚠ Ne pas exécuter de programme en tant que `root` si ça n'est pas strictement nécessaire

Connexion à un système UNIX

Le système a recours à un mécanisme d'authentification (e.g. mot de passe)

Basé sur une fonction h non-inversible

Authentification avec un mot de passe p:

- Le mdp crypté $e = h(p)$ est stocké sur le serveur
- L'utilisateur envoie un mdp p' au serveur
- Le serveur calcule $h(p')$
- Le serveur accepte la connexion si $h(p') = e$

Note: Le serveur ne connait pas le mdp non crypté p!

Invite de commandes (shell)

Prompt: (ne pas recopier)

```
$
```

Format des commandes:

```
$ nom_commande [options] [arguments]
```

Commande simple

```
$ ls
steibergs.png test.c test.dat test.dat
```

Commande avec option (-l : long listing format)

```
$ ls -l
-rw-r--r-- 1 bnegreve bnegreve 14674 Jul 2 2019 steibergs.png
-rw-r--r-- 1 bnegreve bnegreve 78 Nov 16 23:48 test.c
-rw-r--r-- 1 bnegreve bnegreve 16 Dec 5 19:35 test.dat
-rw-r--r-- 1 bnegreve bnegreve 16 Dec 5 19:35 test.dat
```

Commande avec argument et option

```
$ wc -l test.txt
438
```

Autres exemples

```
$ who # qui est connecté sur la machine
or       pts/11       2020-01-08 14:25 (10.1.3.176)
tamby    pts/12       2020-01-09 12:52 (10.1.3.208)
tamby    pts/13       2020-01-09 13:26 (10.1.3.208)
tamby    pts/20       2020-01-09 14:07 (10.1.3.208)
bnegrevergne pts/21     2020-01-27 15:03 (10.1.3.208)
$ cat test.txt # affiche le contenu du fichier test.txt dans le terminal
salut
$ pwd # affiche le nom de repertoire courant
/home/bnegrevergne
$ mkdir unix # crée un nouveau repertoire dans le repertoire courant
$ cd unix # se déplace dans le répertoire unix
$ mv a.txt b.txt # renomme ou déplace un fichier
$ gzip b.txt # compresse un fichier
$ date # affiche la date
$ cal # affiche un calendrier
$ sort # trie les lignes dans l'ordre alphabétique (ou autre)
```

Obtenir de l'aide

```
$ man ls
```

```
LS(1)

      User Commands

      LS(1)

NAME
       ls - list directory contents

SYNOPSIS
       ls [OPTION]... [FILE]...

DESCRIPTION
       List information about the FILEs (the current directory by default). Sort
            entries alphabetically if none of -cftuvSUX nor --sort is specified.

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

       -a, --all
              do not ignore entries starting with .

       -A, --almost-all
              do not list implied . and ..
```

man man

```
1       1   Executable programs or shell commands
2       2   System calls (functions provided by the kernel)
3       3   Library calls (functions within program libraries)
4       4   Special files (usually found in /dev)
5       5   File formats and conventions, e.g. /etc/passwd
6       6   Games
7       7   Miscellaneous (including macro packages and conventions), e.g. man(7),
              groff(7)
8       8   System administration commands (usually only for root)
9       9   Kernel routines [Non standard]
```