

Introduction à l'Informatique (INF 311)



F. Morain



Amphi 10 : système; sécurité

3 juillet

- I. Introduction à UNIX: histoire, processus, fichiers.
- II. Sécurité: firewall, virus.
- III. Conclusions sur le cours.

I. Introduction à UNIX

Z) Qu'est-ce qu'un système d'exploitation?

Interface de haut niveau avec la machine, comme un chauffeur de maître pour une rolls.

S'occupe de la gestion des fichiers, des processus (programmes), des périphériques (imprimante, carte réseau, etc.) de leurs interactions (entre eux ou entre utilisateurs).

Comment communique-t-on? interprète de commandes (SHELL, MSDOS) de plus en plus souvent remplacés par des interfaces graphiques (systèmes de fenêtrage).

0. Où en sommes-nous?

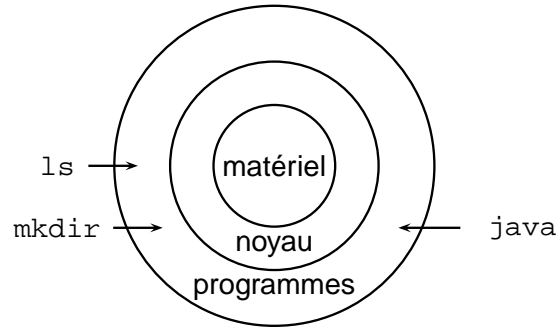
- Amphi 1 : introduction.
- Amphi 2 : programmer en Java.
- Amphi 3: fonctions/fonctions récursives.
- Amphi 4: tableaux/String.
- Amphi 5: classes.
- Amphi 6: tables.
- Amphi 7: algorithmes et complexité.
- Amphi 8: Internet.
- Amphi 9: listes.
- Amphi 10: système/sécurité.

A) Un bref historique

- 1965 : Bell Labs, General Electric et le MIT développent le système Multics. Échec partiel.
- De 1970 à 1980 : Ken Thompson et Dennis Ritchie développent Unix, qui est tout, sauf un logiciel propriétaire!
- Expansion rapide dans le milieu de la recherche ; plusieurs variantes (SYSTEM V d'AT&T, BSD à Berkeley, . . . , GNU/Linux).

Principales caractéristiques

- Multi-utilisateurs, multi-tâches.
- Isole la partie logicielle de la partie matérielle.



- Clarté, simplicité; écrit dans un langage de haut niveau (langage C) ⇒ très facile à porter sur les nouvelles architectures.

Le matériel passe, le logiciel reste !

B) Les processus

processus = programme en cours d'exécution.

Le noyau permet la création, la suspension, la fin d'un processus. Il gère de manière équitable l'exécution de tous les processus présents. Il alloue et gère la place mémoire nécessaire à leur exécution.

Les différents processus sont stockés dans une table et repérés par leur numéro d'ordre.

```
unix% ps
PID TTY          TIME CMD
 646 pts/1        00:00:00 bash
 740 pts/1        00:00:00 ps
```

Fabrication et gestion d'un processus

Un processus consiste en une suite d'octets que le processeur interprète comme des instructions machine, ainsi que des données et une pile.

Chaque processus s'exécute dans une zone de la mémoire qui lui est réservée et protégée des autres processus. Il peut lire ou écrire les données dans sa zone mémoire propre, sans pouvoir interférer avec celles des autres processus.

⇒ plusieurs utilisateurs peuvent exécuter le même programme simultanément : un processus nouveau est créé à chaque fois et une nouvelle zone de mémoire physique lui est allouée.

```
PID TTY          STAT       TIME COMMAND
  1 ?            S          0:03  init [5]
  2 ?            SW         0:00  [keventd]
  3 ?            SW         0:00  [kapm-idled]
  4 ?            SW         0:00  [kswapd]
  5 ?            SW         0:00  [kreclaimd]
  6 ?            SW         0:00  [bdflush]
  7 ?            SW         0:00  [kupdated]
  8 ?            SW<        0:00  [mdrecoveryd]
 50 ?            SW         0:00  [khubd]
544 ?            S          0:00  /usr/sbin/ssfd
620 ?            S          0:00  sendmail: accepting connections
895 ?            S          0:00  fvwm2
950 ?            S          0:00  xterm -geometry 80x45+3+0 -j -s -l
951 ?            S          0:00  xterm -geometry 80x45+270+0 -j -s
1066 pts/0        R          0:00  ps ax
```

État

Le **contexte** d'un processus est la donnée de son code, les valeurs des variables, les structures de données, etc. qui lui sont associées, ainsi que de son état. À un instant donné, un processus peut être dans quatre **états** possibles :

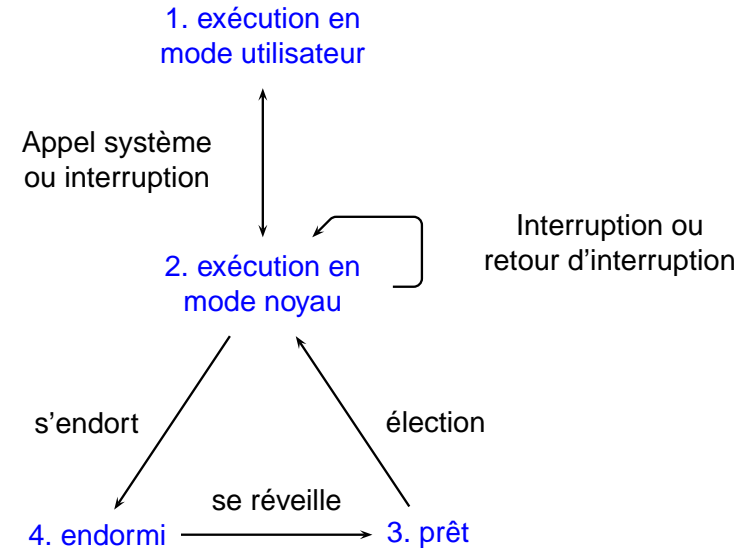
- en cours d'exécution en **mode utilisateur** ;
- en cours d'exécution en **mode noyau** ;
- pas en exécution, mais **prêt** à l'être ;
- **endormi**, quand il ne peut plus continuer à s'exécuter (parce qu'il est en attente d'une entrée sortie par exemple).

L'ordonnancement des tâches

UNIX est multi-tâches : le processeur accorde à chaque processus un peu de temps de traitement à tour de rôle.

Principe : allouer à chaque processus un **quantum** de temps pendant lequel il peut s'exécuter dans le processeur. À la fin de ce quantum, le système le préempte et réévalue la priorité de tous les processus au moyen d'une file de priorité. Le processus avec la plus haute priorité dans l'état "prêt à s'exécuter, chargé en mémoire" est alors introduit dans le processeur. La priorité d'un processus est d'autant plus basse qu'il a récemment utilisé le processeur.

Le temps est mesuré par une horloge matérielle, qui interrompt périodiquement le processeur (générant une interruption). À chaque top d'horloge, le noyau peut ainsi réordonner les priorités des différents processus, ce qui permet de ne pas laisser le monopole du processeur à un seul processus.



Le mystère du démarrage

La première tâche à accomplir est de faire charger en mémoire le programme de mise en route (*boot*) du système.

Comment le système peut-il se charger lui-même ?

Dans les temps anciens, il s'agissait d'une procédure quasi manuelle. De nos jours, le processeur à peine réveillé va lire une mémoire ROM contenant les instructions de boot. Après quelques tests, le système récupère sur disque le noyau (fichier `/vmlinix`). Le programme de boot transfère alors la main au noyau qui commence à s'exécuter (processus 0), qui crée le processus 1 (`init`) et se transforme en `kswapd`.

C) La gestion mémoire

Chaque processus fait comme s'il avait toute la mémoire à lui tout seul (mémoire virtuelle). Le système s'occupe de faire coïncider cette mémoire virtuelle avec la mémoire physique.

Tant que la mémoire centrale peut contenir toutes les demandes des utilisateurs, tout va bien.

Le système se contente de charger en mémoire la partie de celle-ci dont le processus a vraiment besoin à un instant donné (pagination), le reste est swappé.

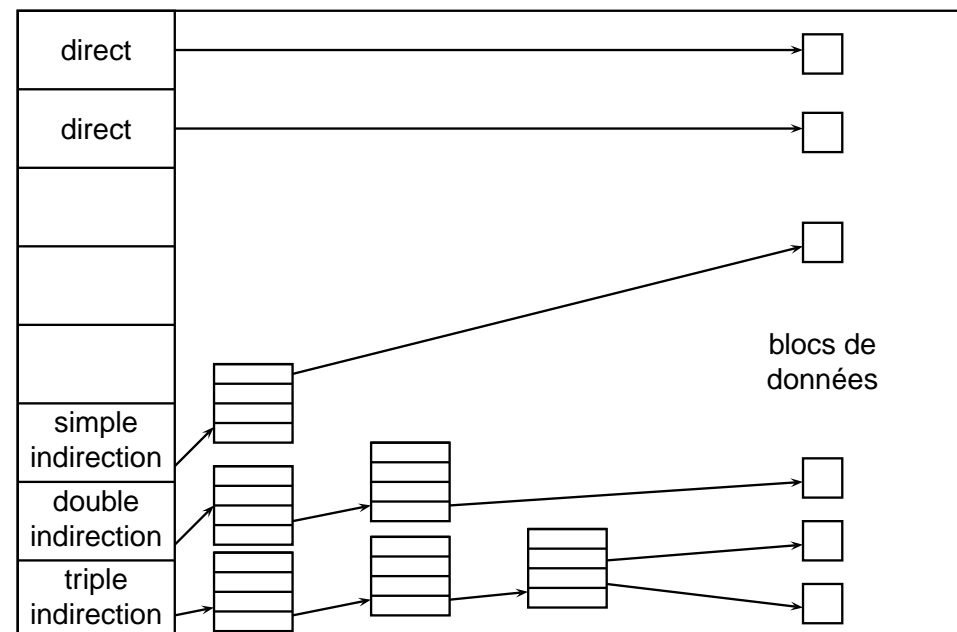
Quand la mémoire approche de la saturation, le système définit des priorités d'accès à la mémoire.

Mise en pratique

- En interne, un fichier est décrit par un **i-nœud** (*inode* pour *index node*). Un fichier est vu comme une suite d'octets, rassemblés en blocs. Ces blocs n'ont pas vocation à être contigus (cf. amphi 9).
- **B-arbre** : table contenant des numéros de blocs (blocs directs, simple indirection, double indirection, triple indirection).

D) Les fichiers sous UNIX

- **Système arborescent** : l'ancêtre de tous les chemins possibles étant la racine (notée /) ; on dispose de répertoires, qui contiennent eux-mêmes des répertoires et des fichiers.
- **Vision d'un fichier par l'utilisateur** : espace mémoire potentiellement infini dans lequel il peut ranger ses données à sa convenance.
- Les fichiers sont **non typés** : tous les fichiers (y compris les répertoires) contiennent des suites d'octets, charge aux programmes de les interpréter correctement.
- Les périphériques sont traités comme des fichiers (/dev/audio).



Généralement: les fichiers sont stockés sur un serveur de fichiers centralisé (α ou ω).

Quand vous voulez éditer un fichier: le démon **nfsd** attend et traite les requêtes:

- le serveur renvoie le fichier (par le réseau);
- vous travaillez sur le fichier en local (dans l'espace mémoire du processus);
- quand vous le sauvegardez, il est renvoyé au serveur de fichiers.

Il peut y avoir des problèmes de **concurrency** si le même fichier est édité en même temps sur deux machines.

A) Un peu d'intimité

Comme Unix est un système multi-utilisateurs, il est bon de préserver l'intimité de ceux-ci.

Chaque utilisateur appartient à un *groupe* défini dans le fichier `/etc/group`.

Trois niveaux d'acteurs: l'utilisateur, son groupe, et les autres. Quand l'utilisateur crée un fichier, celui-ci se voit attribuer des droits par défaut, que l'on peut observer à l'aide de la commande `ls -lg`, comme dans l'exemple:

```
-rw-r--r-- 1 morain users 3697 Apr 7 18:08 poly.tex
```

Qu'est-ce que la sécurité informatique?

- **Protection de l'ordinateur contre l'utilisateur:** mémoire protégée, etc.
- **Protection des données:** droits d'accès; duplication, fragmentation, etc.
- Contrôle des **accès à distance**.
- Empêcher les **virus** de se propager.
- Être sûr que les programmes n'ont pas de **bugs vitaux** (systèmes embarqués dans les avions, les voitures, etc.).
- **La sécurité commence par un bon système d'exploitation... ! Et un bon protocole réseau (TCP > UDP).**

On peut changer ces permissions par la commande `chmod`:

```
unix% chmod o-r poly.tex
```

empêche les autres de lire le fichier:

```
unix% ls -lg poly.tex
-rw-r----- 1 morain users 3697 Apr 7 18:08 poly.tex
```

Notons pour finir que les droits de création des fichiers sont régis par défaut par la variable d'environnement `UMASK` qui est positionnée au lancement de l'environnement de l'utilisateur. À l'X, elle a une valeur parano (personne ne peut lire votre compte).

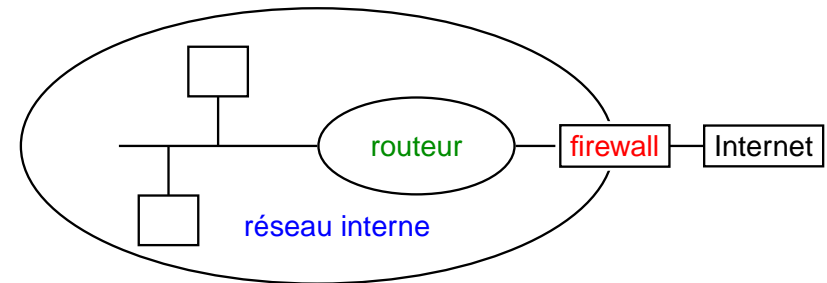
Quelques applications sécurisées

SSH: (secure shell) login à distance (**slogin**).

SSL/TLS: tunnel chiffrant pour échanger des données. Par ex., HTTPS envoie des pages web dans ce mode-là.

Ces outils utilisent la **cryptographie asymétrique** (certificats, etc.).

B) Firewall et filtrage de paquets



Un firewall permet de filtrer toutes les données en entrée (accès, mail, etc.), réduisant la maintenance des machines en interne. Fait souvent du **filtrage de paquets**.

Filtrage bas niveau

```
From: alice@poly
To: bob@isp
Subject: Re: déjeuner?
```

SMTP

```
CONNECT <129.104.1.1.25,133.12.104.12.25>
```

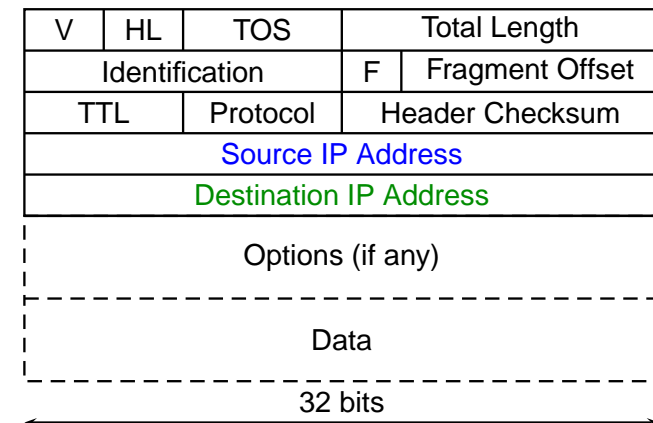
TCP

```
133.12.104.12, port 25
```

TCP demande une connection entre le port 25 de la machine client et le port 25 de la machine serveur, puis envoie les paquets.

TCP

Transmission Control Protocol: l'information est découpée en **paquets** (paquets IP) qui sont envoyés sur le réseau.



Protocole	nom complet	Port(s)
FTP	File Transfer Protocol	21/20
SMTP	Simple Mail Transfer Protocol	25
DNS	Domain Name System	53
HTTP	Hypertext Transfer Protocol	80
NNTP	Network News Transfer Protocol	119

En général, ces **services** utilisent toujours ces ports.

Scanner tous les ports, c'est pas bien!

WARNING WARNING

Les attaques décrites
ci-après constituent un
casus belli de la plus belle
eau.

WARNING WARNING

Pour se protéger d'accès indésirables (trous de sécurité liés à certains programmes qui **écoutent** certains ports), il est prudent de filtrer les paquets, en entrée de site, comme chez soi (ADSL).

Règles de filtrage typiques:

Adr. source	Adr. dest.	Port source	Port dest.	action
*	123.4.5.6	> 1023	23	permit
*	123.4.5.7	> 1023	25	permit
129.6.48.254	123.4.5.6	> 1023	119	permit
*	*	*	*	deny

Attaque par submersion (flooding)

Le protocole de connexion:

$$\begin{aligned} A &\rightarrow B : \text{SYN}(X) \\ A &\leftarrow B : \text{SYN}(Y), \text{ACK}(X) \\ A &\rightarrow B : \text{ACK}(Y) \end{aligned}$$

Rem. X et Y sont des entiers souvent prévisibles (compteurs).

Attaque: C envoie des milliards de demandes de connexion au serveur, mais ne renvoie aucun ACK(Y), ce qui bloque le serveur, qui ne peut plus accepter de connexions.

Défense: utiliser un firewall qui s'interpose et coupe les connexions trop longues.

Spoofing

Contexte: B fait confiance à A; C lance une attaque sur B en se faisant passer pour A.

$C/A \rightarrow B : \text{SYN}(X)$
 $A \leftarrow B : \text{SYN}(Y), \text{ACK}(X)$
 $C \rightarrow B : \text{ACK}(Y)$

C fabrique un message TCP qui ressemble à un message émanant de A (avec le numéro IP de A) et envoie $\text{SYN}(X)$ à B.

Si C devine Y, il renvoie $\text{ACK}(Y)$ à B et donc établit la connexion.

C doit empêcher le message $\text{SYN-ACK}(Y)$ d'arriver à A, par exemple en lançant un flooding sur A.

Défense: nombres aléatoires; filtrage des paquets (ne pas accepter un message extérieur arrivant avec une adresse intérieure);
authentifier la source par des techniques cryptographiques.

D) Virus

Déf. (É. Filiol) Un virus est une séquence de symboles qui, interprétée dans un environnement adéquat, modifie d'autres séquences de symboles dans cet environnement, de manière à y inclure une copie de lui-même, copie ayant éventuellement évolué.

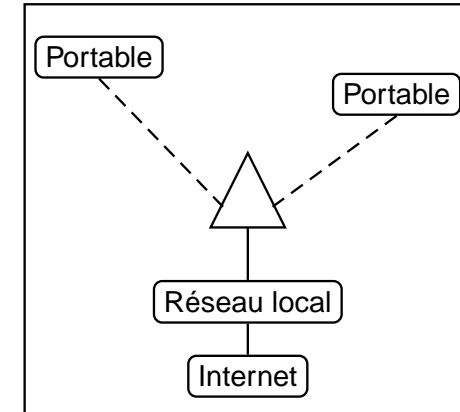
Mode de fonctionnement typique: un déclencheur (programme trop intelligent; utilisateur moldu) permet le démarrage de l'infection; le virus infecte alors les fichiers et peut se répandre.

Il y a encore quelques années: infection par disquettes (vive les formats simplistes – comme le FAT !!!)

Aujourd'hui: propagation par le réseau, ce qui demande plus de connaissances au pirate, mais aussi aux fabricants d'anti-virus.

C) Un exemple à ne pas suivre : IEEE 802.11

WiFi (Wireless Fidelity), **Internet ambient**: INTERNET par radio.



Brève histoire (d'après É. Filiol; O. Zilbertin)

- 1981: Xerox (accident); virus pour l'Apple II.
- 1983: Elk Cloner pour AppleDOS 3.3.
- 1986: thèse de Fred Cohen (élève de Len Adleman), invente le terme **virus**.
- 1988: **ver internet**.
- 1991: **Frodo/4096** arrive par l'intermédiaire d'une disquette vendue dans la revue *Soft et Micro*.
- 1998: on estime à **17745 virus différents recensés en 1998, contre 18 en 1989**.
- 2000: virus "**I love you**" se propage très vite.

Le ver internet de 1988 et ses conséquences

- Auteur connu: **Robert T. Morris, Jr**; finalement condamné en 1991 à trois années de probation, 400 heures de travaux d'intérêt général et 10,000 \$ d'amende.
- Utilise des **bugs** dans **sendmail**, **finger** (buffer overflow).
- Cherche des comptes avec des **mots de passe faibles** (craquage) pour se propager au moyen **rsh**, **rexec** ⇒ remplacement de **/etc/passwd** par **/etc/shadow**.
- Utilise des techniques de **furtivité** (effacement des arguments, dissimulation du nom du programme, fork, etc.).
- **Mauvaise gestion de crise**: démarrage le 2/11/88, fin de crise le 8/11/88!!! Couper le mail n'était pas une bonne idée.

Le cas de W32/Korgo.F (W32/Padobot) – juin 2004

"This worm attempts to take advantage of a buffer overflow vulnerability in the Windows Local Security Authority Service Server (LSASS). The vulnerability allows a remote attacker to execute arbitrary code with SYSTEM privileges. [...]"

The worm propagates by scanning random IP addresses on port 445/tcp for vulnerable systems. Upon finding a vulnerable system [...] this worm will open a connection on port 113/tcp or port 3067/tcp and may attempt to connect to a list of pre-determined IRC servers."

Pour connaître les derniers virus.

Le cas de Nimda

Découvert le 18/09/01, attaque les systèmes Windows*.

Cycle de vie:

1. **Entre** en utilisant deux trous de sécurité : *Unicode exploit* pour infecter les serveurs IIS et *MIME exploit*.
2. **Infection des fichiers**: localise les fichiers EXE et les assimile, propagation par échange de fichiers (classique).
3. **Envoi massif de courriels**: récupère des adresses mail dans le client de courrier et dans les pages html; envoie alors un courrier à toutes les adresses avec un fichier README.EXE en attachement (grand classique).
4. **Attaque des serveurs web**: recherche les serveurs web proches; quand il en trouve un, essaie d'y entrer avec l'un des trous de sécurité connus. S'il y arrive, modifie des pages web au hasard pour infecter les navigateurs des surfers.
5. **Propagation par les réseaux locaux**: en cas de fichiers partagés, crée un fichier RICHED20.DLL qui sera activé par Word, etc., ce qui provoquera une infection de la machine distante au moment de l'activation.

Thm. (F. Cohen, 1986) **Détecter un virus est un problème indécidable.**

Démonstration: supposons qu'il existe une fonction **Detecte(fonction_java)**, on construit:

```
static void f(){
    if(! Detecte(f))
        infecter();
}
```

Rem. C'est le retour de Papyrus!

Conséq. On ne peut pas détecter un virus de façon syntaxique pure. Sauf si on a détecté une **signature** particulière. On soigne, mais on ne prévient pas l'infection.

Conclusion sur les virus

Une menace de plus en plus présente dans l'avenir (téléphones mobiles).

Attaques de plus en plus professionnelles, avec des buts de plus en plus divers (spam).

Virus **de plus en plus difficiles à éradiquer**: virus de BIOS, virus binaires, etc.

Vaccins? on peut utiliser le même mode de propagation pour soigner son parc de machines. . .

Quelques défis à relever

- **Intégrer l'informatique dans votre vision du monde!**
- Déploiement sans précédent de l'informatique embarquée (voitures).
- Sécurité au sens large.
- Loi de Moore.

L'informatique doit servir. . .

. . . pas asservir.

III. Conclusions sur le cours

Ce qu'on a vu: programmation en **Java**, demande de la rigueur, base essentielle pour voir les autres domaines de l'informatique:

- **architecture des processeurs et ordinateurs (machines parallèles, vectorielles, multiprocesseurs); systèmes d'exploitation; réseaux; programmation distribuée et temps-réel;**
- **langages (conception, compilation, etc.); logique mathématique et preuves mécaniques; certification des programmes;**
- **algorithmique (conception, analyse); calcul formel; protection des données (crypto); bases de données; images (analyse, synthèse, vision; géométrie algorithmique, etc.); robotique; etc.**

Prochain rendez-vous : pale papier, mardi 11 juillet (9h–11h).

Consignes: arrivez à l'heure en sachant où vous allez (T5 ou T6 – il faut courrir vite en cas d'erreur); **tous documents permis**; les EV2 auront 30 minutes de plus, et des énoncés en anglais disponibles.

Si vous ratez, faites-vous rattraper! Important pour aller aux US après. . .

Jedis' tricks:

- lisez l'énoncé jusqu'au bout pour noter par avance ce que vous pensez savoir faire facilement, d'autant plus que les exercices sont indépendants;
- si vous ne savez pas faire une question, abandonnez la pour en essayer une autre, quitte à y revenir plus tard.

Générique de fin (?)

Merci

- aux deux Scola pour leur aide irremplaçable;
- aux 1703 élèves qui ont subi mon cours;
- à toutes les équipes qui ont eu à me supporter depuis 6 ans.