# Organisation et utilisation des systèmes d'exploitation 2

1. Introduction

Guillaume Pierre

Université de Rennes 1

Hiver 2014




## Table of Contents





## Table of Contents





## Objectifs

- **Organisation des systèmes d'exploitation**
  - Mieux comprendre le **fonctionnement d'une machine**
  - Pour pouvoir ensuite mieux l'utiliser
- Langage C
- **Utilisation des systèmes d'exploitation**
  - Fonctionalités avancées, programmation avancée

## Organisation

- 8 cours magistraux
- 4 séances de TP
  - Pour vous familiariser avec les concepts essentiels
- 12 séances de projet
  - Développement à cheval sur plusieurs séances
- 1 TD
  - Pour vous préparer à l'examen
  - Attention: **il n'y aura pas de séance de TD pour répéter ce qu'on a dit en cours!**
  - Mais vous pouvez poser autant de questions que vous voulez en TP et en séances de projet...



## Table of Contents





## Traduction de langages

Que se passe-t-il quand vous exécutez **javac Foo.java** ?

- Le programme Foo en langage "Java" est **traduit** en un programme équivalent en langage "bytecode Java".
- Formellement:
  - Remplacer chaque instruction en langage L1 par une séquence d'instructions équivalentes en langage L0.
  - L0 est un langage de plus bas niveau que L1 (plus proche de ce que des transistors peuvent faire)
  - On peut ensuite exécuter le programme en langage L0 autant de fois que l'on veut
- Autre possibilité: **interpréter** le langage L1
  - Lire une instruction en langage L1, exécuter une séquence équivalente en langage L0
  - Lire l'instruction L1 suivante, etc.
  - C'est essentiellement ce que fait votre *machine virtuelle Java*

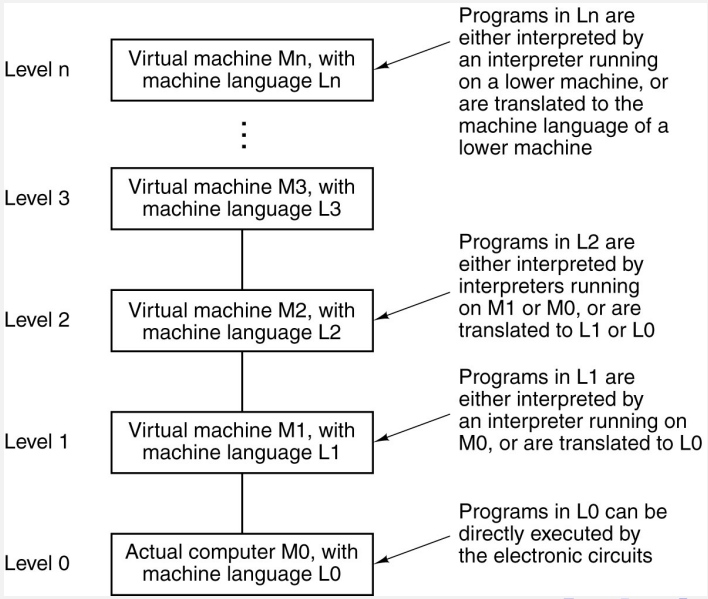
**Java → traduit en bytecode Java → interprété en assembleur**



## Machines virtuelles

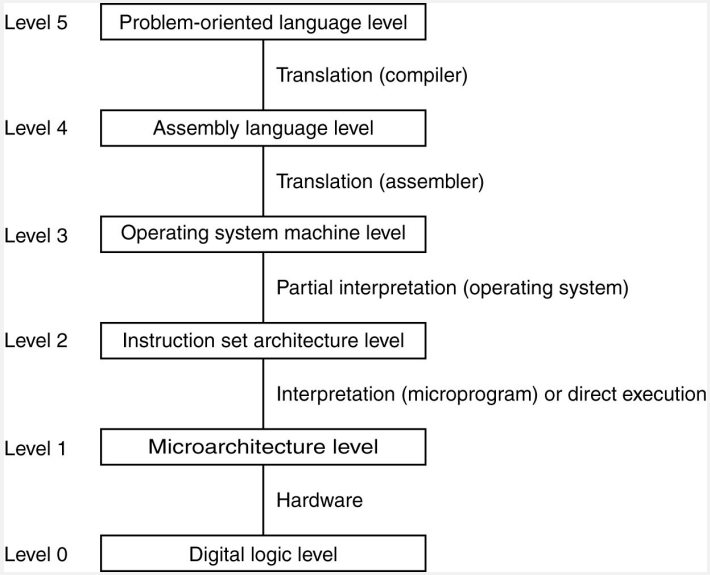
- Imaginez une machine conçue pour exécuter uniquement le langage L1
- On peut construire cette **machine physique** entièrement en matériel (transistors etc.)
  - Si L1 est complexe ça va être difficile et coûteux
  - Si vous changez le langage L1 il faut jeter votre machine et en reconstruire une nouvelle
  - Tous vos programmes doivent être écrits en langage L1
- On peut programmer cette **machine virtuelle** en logiciel
  - La machine virtuelle L1 s'exécute sur la machine physique L0
  - Si vous changez L1 il suffit de mettre la machine virtuelle L1 à jour
  - Si vous voulez utiliser un autre langage de programmation il suffit de construire une machine virtuelle L2 en langage L0 ou L1
- Les machines modernes sont composées de **plusieurs couches de machines virtuelles** les unes sur les autres

## Organisation en couche d'un ordinateur






## Organisation en couche d'un ordinateur






## Tour de magie!

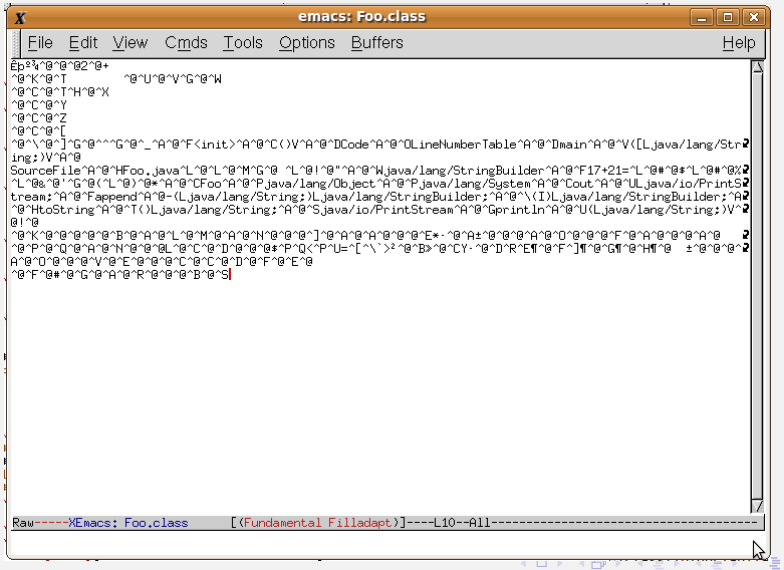
```
$ cat Foo.java
public class Foo {
    public static void main(String[] args) {
        int i = 17;
        int j = 21;
        int k = i + j;
        System.out.println("17+21=" + k);
    }
}
$ javac Foo
$ java Foo
17+21=38
$
```

**Que s'est-il passé?**



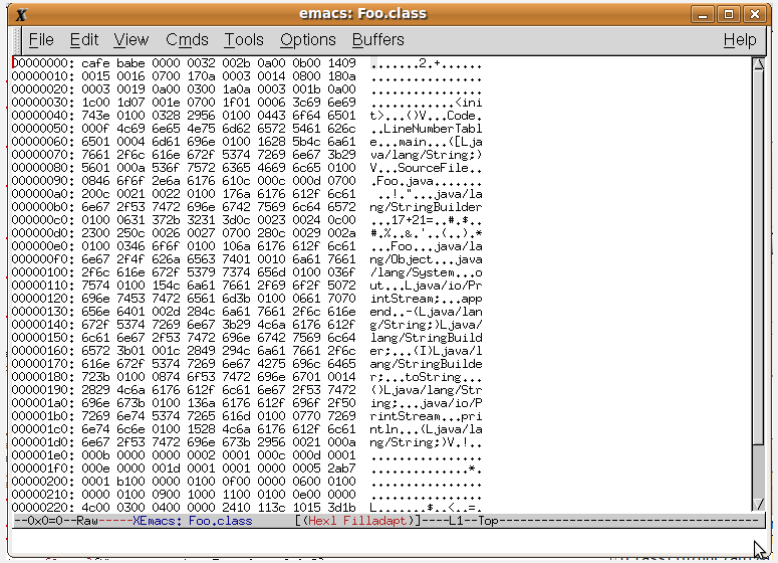
## Niveau langage utilisateur (L5)

**Ouvrons le bytecode Java bytecode dans un éditeur de texte:**



## Niveau langage utilisateur (L5)

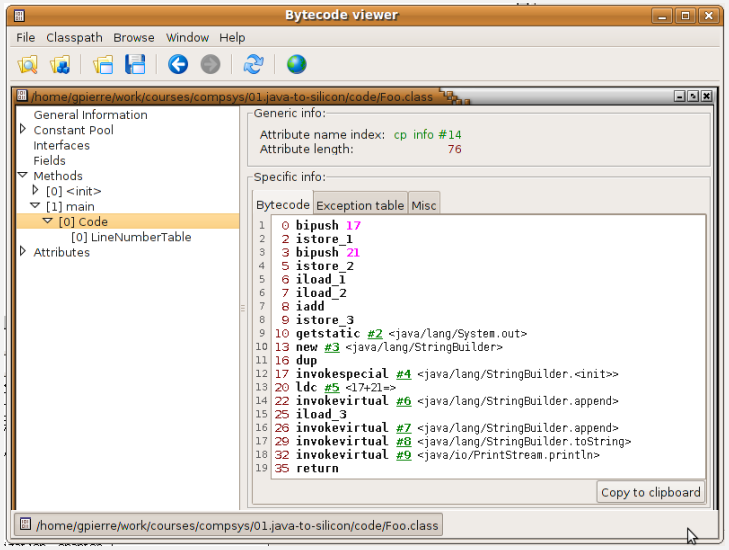
**En mode binaire:**






## Niveau langage utilisateur (L5)

**Avec un éditeur de bytecode Java:**




http://www.ej-technologies.com/products/jclasslib/overview.html



## Niveau assembleur (L4)

- Le programme "java" interprète le fichier 'Foo.class'
  - Le programme "java" est écrit en langage **Assembleur**
  - Plus bas niveau que le bytecode Java
  - **Beaucoup** plus bas niveau que Java
- Est-ce que l'assembleur s'exécute directement sur le hardware?
  - Le **langage assembleur** est un langage *relativement* lisible
  - Il faut le traduire en **langage machine** avant de l'exécuter sur le hardware

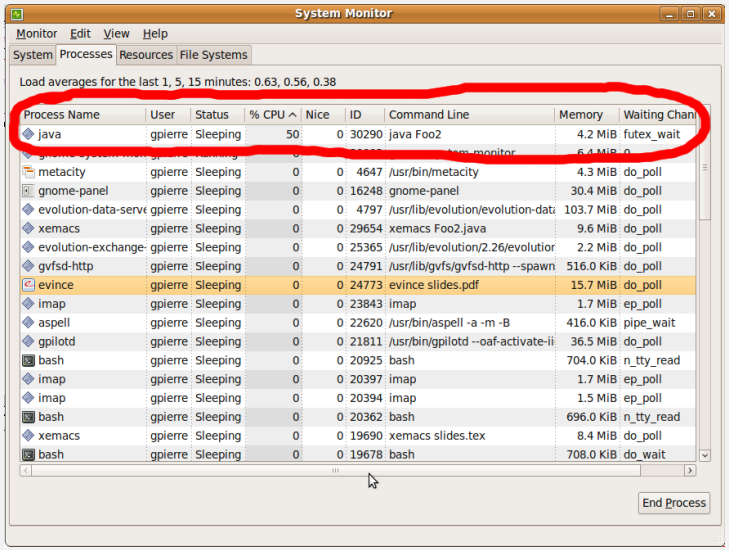


## 17+21 en Assembleur

```
$ cat foo.c
#include <stdio.h>
int main() {
  int i = 17;
  int j = 21;
  int k = i+j;
  printf("i+j=%d\n", k);
  return 0;
}
$ gcc -S foo.c
$ cat foo.s
        .file   "foo.c"
        .section .rodata
.LC0:
        .string "i+j=%d\n"
        .text
.globl main
        .type   main, function
main:
        leal    4(%esp), %ecx
        andl    $-16, %esp
        pushl   -4(%ecx)
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %ecx
        subl    $36, %esp
        movl    $17, -8(%ebp)
        movl    $21, -12(%ebp)
        movl    -12(%ebp), %edx
        movl    -8(%ebp), %eax
        addl    %edx, %eax
        movl    %eax, -16(%ebp)
        movl    -16(%ebp), %eax
        movl    %eax, 4(%esp)
        movl    $.LC0, (%esp)
        call    printf
        movl    $0, %eax
        addl    $36, %esp
        popl    %ecx
        popl    %ebp
        leal    -4(%ecx), %esp
        ret
        .size   main, .-main
        .ident  "GCC: (Ubuntu 4.3.3-5ubuntu4) 4.3.3"
        .section .note.GNU-stack,"",progbits
$
%$
```

## Niveau système d'exploitation (L3)

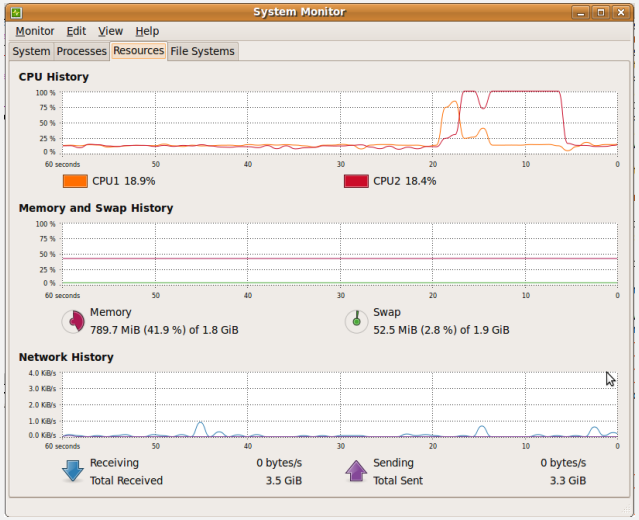
Exécutons un programme Java un peu plus long: fib(45)...






## Niveau système d'exploitation (L3)

Exécutons un programme Java un peu plus long: fib(45)...






## Le système d'exploitation: une couche hybride

- Vouz avez sans doute entendu dire que "*les programmes s'exécutent sur le système d'exploitation*"
  - **C'est faux!!** (99% du temps)
- Le système d'exploitation est une **couche hybride**
  - La plupart du temps les programmes en langage machine (par exemple: "java") d'exécutent **directement sur le hardware**
  - Mais pour certaines opérations sensibles le programme en langage machine **demande de l'aide** au système d'exploitation
    - ★ **"Appel système"**
- De temps à autres le système d'exploitation **reprend le contrôle** tout seul
  - Afin de permettre à d'autres programmes de s'exécuter ("processus")
  - Et pour d'autres tâches de fond...



## Niveau jeu d'instructions (L2)

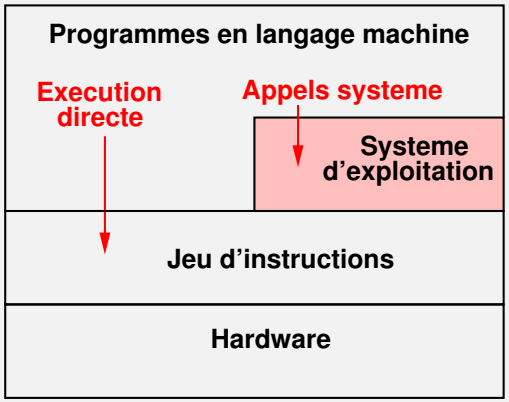
- Différents processeurs utilisent des langages machine différents
  - Par exemple: le langage machine x86 est **très différent** du langage machine SPARC our ARM...
  - Même si l'essentiel des concepts restent similaires
- Chaque **famille de processeurs** a son propre langage machine
  - Le langage machine de votre processeur Intel core i7 est **fondamentalement le même** que celui d'un 8086 des années 1980
  - On a rajouté de nouvelles fonctionalités (de plus en plus complexes)
  - Mais votre machine moderne reste compatible avec les anciens programmes
- Bien entendu l'architecture matérielle d'un Intel core i7 n'a strictement rien à voir avec celle d'un 8086...

## Table of Contents





## Le système d'exploitation: une couche hybride






## Appels système

```
$ strace java5 Foo
[...]
stat64("/usr/lib/jvm/java-1.5.0-sun-1.5.0.19/jre/lib/i386/server/libjvm.so", {st_mode=S_IFREG|064
getgid32()                              = 1000
getegid32()                             = 1000
getuid32()                              = 1000
geteuid32()                             = 1000
execve("/usr/lib/jvm/java-1.5.0-sun-1.5.0.19/jre/bin/java", ["java5", "Foo"], [/* 53 vars */]) =
[...]
stat64("/home/gpierre/compsys/Foo.class", {st_mode=S_IFREG|0644, st_size=631, ...}) = 0
open("/home/gpierre/compsys/Foo.class", O_RDONLY|O_LARGEFILE) = 3
stat64("/home/gpierre/compsys/Foo.class", {st_mode=S_IFREG|0644, st_size=631, ...}) = 0
read(3, "\312\376\272\276\0\0\0001\0+\n\0\v\0\24\t\0\25\0\26\7\0\27
     \n\0\3\0\24\10\0\30\n\0"..., 631) = 631
close(3)                                = 0
[...]
write(1, "17+21=38"..., 8)              = 8
write(1, "\n"..., 1)                    = 1
[...]
```



## System calls

- Les appels système permettent aux programmes de **demander quelque chose au système**
- **Gestion de processus:**

| | |
|---|---|
| pid = fork() | Create a child process identical to the parent |
| pid = waitpid(pid, &statloc, opts) | Wait for a child to terminate |
| s = wait(&status) | Old version of waitpid |
| s = execve(name, argv, envp) | Replace a process core image |
| exit(status) | Terminate process execution and return status |
| size = brk(addr) | Set the size of the data segment |
| pid = getpid() | Return the caller's process id |
| pid = getpgrp() | Return the id of the caller's process group |
| pid = setsid() | Create a new session and return its process group id |
| l = ptrace(req, pid, addr, data) | Used for debugging |

## System calls

- **Gestion de signal:**

| | |
|---|---|
| s = sigaction(sig, &act, &oldact) | Define action to take on signals |
| s = sigreturn(&context) | Return from a signal |
| s = sigprocmask(how, &set, &old) | Examine or change the signal mask |
| s = sigpending(set) | Get the set of blocked signals |
| s = sigsuspend(sigmask) | Replace the signal mask and suspend the process |
| s = kill(pid, sig) | Send a signal to a process |
| residual = alarm(seconds) | Set the alarm clock |
| s = pause() | Suspend the caller until the next signal |



## System calls

- **Gestion de fichiers:**

| | |
|---|---|
| fd = creat(name, mode) | Obsolete way to create a new file |
| fd = mknod(name, mode, addr) | Create a regular, special, or directory i-node |
| fd = open(file, how, ...) | Open a file for reading, writing or both |
| s = close(fd) | Close an open file |
| n = read(fd, buffer, nbytes) | Read data from a file into a buffer |
| n = write(fd, buffer, nbytes) | Write data from a buffer into a file |
| pos = lseek(fd, offset, whence) | Move the file pointer |
| s = stat(name, &buf) | Get a file's status information |
| s = fstat(fd, &buf) | Get a file's status information |
| fd = dup(fd) | Allocate a new file descriptor for an open file |
| s = pipe(&fd[0]) | Create a pipe |
| s = ioctl(fd, request, argp) | Perform special operations on a file |
| s = access(name, amode) | Check a file's accessibility |
| s = rename(old, new) | Give a file a new name |
| s = fcntl(fd, cmd, ...) | File locking and other operations |



## System calls

- **Gestion de répertoires et systèmes de fichiers:**

| | |
|---|---|
| s = mkdir(name, mode) | Create a new directory |
| s = rmdir(name) | Remove an empty directory |
| s = link(name1, name2) | Create a new entry, name2, pointing to name1 |
| s = unlink(name) | Remove a directory entry |
| s = mount(special, name, flag) | Mount a file system |
| s = umount(special) | Unmount a file system |
| s = sync() | Flush all cached blocks to the disk |
| s = chdir(dirname) | Change the working directory |
| s = chroot(dirname) | Change the root directory |



## System calls

- **Protection:**

| | |
|---|---|
| s = chmod(name, mode) | Change a file's protection bits |
| uid = getuid() | Get the caller's uid |
| gid = getgid() | Get the caller's gid |
| s = setuid(uid) | Set the caller's uid |
| s = setgid(gid) | Set the caller's gid |
| s = chown(name, owner, group) | Change a file's owner and group |
| oldmask = umask(complmode) | Change the mode mask |

## System calls

- **Gestion du temps:**

| | |
|---|---|
| seconds = time(&seconds) | Get the elapsed time since Jan. 1, 1970 |
| s = stime(tp) | Set the elapsed time since Jan. 1, 1970 |
| s = utime(file, timep) | Set a file's "last access" time |
| s = times(buffer) | Get the user and system times used so far |



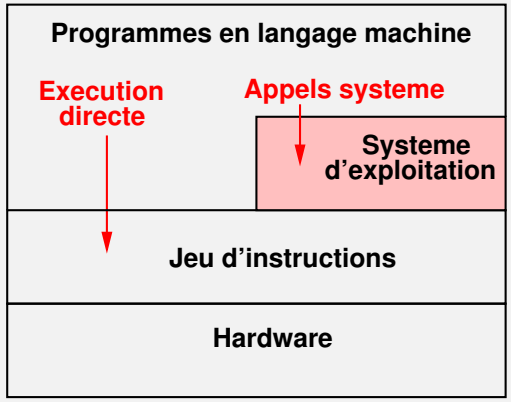
## Table of Contents





## Le système d'exploitation

Le système d'exploitation se trouve entre le hardware,
son jeu d'instructions
et les programmes utilisateurs






## Qu'est-ce qu'un système d'exploitation?

- Un système d'exploitation est une **machine étendue**
  - Le système offre des **opérations de haut niveau** qui n'existent pas dans le langage machine
  - Les programmes utilisent ces nouvelles opérations grâce aux **appels système**
  - Les programmes utilisent aussi toutes les opérations de base du langage machine (sauf quelques instructions réservées au système)
- Un système d'exploitation est un **gestionnaire de ressources**
  - Permettre à plusieurs programmes de s'exécuter simultanément sans se gêner
  - Controller l'accès aux ressources physiques (donner/reprendre un accès à une resource, gérer les conflits, etc.)

## Processus

- Un process est un **programme en train de s'exécuter**
  - Espace mémoire privé (isolé des autres processus)
  - Pile mémoire, registres PC et SP, etc.
  - Un processus est toujours créé par un autre processus
- Le système se charge de **l'ordonnancement des processus**
  - A un instant T, un seul processus peut s'exécuter sur un processeur donné
  - Le système doit s'assurer que chaque processus obtient une fraction équitable du temps CPU
- Les processus sont **isolés** les uns des autres
  - Chaque processus a l'illusion de disposer de la machine entière
  - Les processus peuvent communiquer entre eux grâce à des mécanismes spéciaux de **communication inter-processus**



## Entrées/sorties

- La plupart des processus ont besoin d'**échanger des données** avec des resources externes au CPU (clavier, écran, disque, réseau, iomprimante etc.)
- Toutes les ressources sont différentes!
  - Différents types (disques, claviers, réseaux, etc.)
  - Différentes vitesses
  - Différents constructeurs
  - Etc.
- Le système doit offrir:
  - Une **interface homogène** pour accéder à des ressources différentes
  - Il doit également **arbitrer les conflits entre processus** qui veulent utiliser la même ressource en même temps



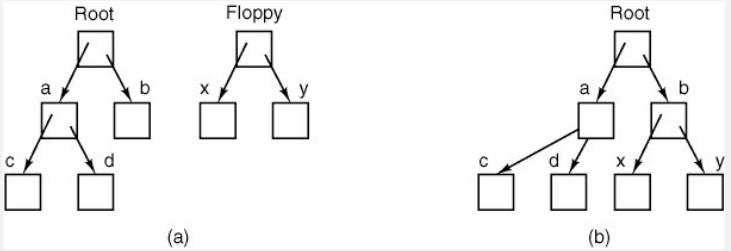
## Gestion de mémoire

- Les processus ont besoin de mémoire
  - Il faut isoler l'espace mémoire entre processus
  - Mieux: donner l'illusion à chaque processus qu'il a accès à **tout l'espace mémoire de la machine** sans se préoccupper des autres processus
  - Encore mieux: donner l'illusion à chaque processus qu'**il peut utiliser l'ensemble de l'espace d'adressage** (adresses mémoire 0 – $2^{32}$) sans se préoccupper de la quantité de mémoire de la machine physique
- Le système supporte:
  - L'isolation mémoire
  - Des mécanismes de **Paging** (c'est-à-dire la possibilité de libérer de la mémoire en copiant son contenu sur disque, et vice-versa) space for others)
  - Des mécanismes de **Memory mapping** (i.e., offrir l'ensemble de l'espace d'adressage à chaque processus)



## Systèmes de fichiers

- Qui parmi vous ne sait pas ce qu'est un fichier? :-)
  - Manipulation de **fichiers, répertoires, protection, etc.**
  - Transformation ces belles abstractions en blocs sur un disque
- Plusieurs systèmes de fichiers peuvent être **montés** en une seule hiérarchie de répertoires




Avant le montage Après le montage

- **Fichiers spéciaux**
  - Beaucoup de concepts système sont représentés sous forme de fichiers
  - Ressources matérielles, communication inter-processus, etc.