55

Système d'Exploitation

86

Sys1 : everyone

Sys2 : electif ⇒ SRS + GISTRE

Assembleur : everyone

OS:

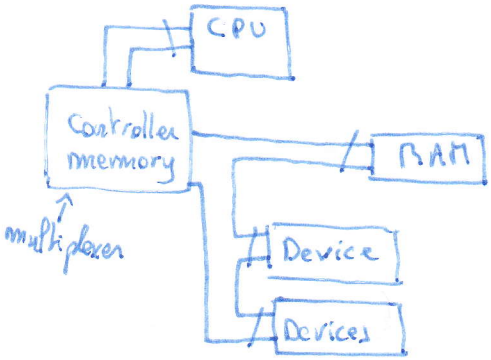
Permet à un développeur de ne pas se préoccuper des autres programmes.

→ Fait l'arbitrage entre les programmes pour l'assignation de ressources.

→ Fournir des interfaces et des abstractions pour les programmes utilisateurs

Architecture d'un ordinateur :

CPU : Lit et écrit en mémoire
+
Execute des instructions



Registre : zone mémoire petite dans le CPU (8B)

- Généraux : Permet d'executer des instructions normales
- Contrôle : Permet de configurer le CPU (et la machine)
  - → Set l'espace d'adressage
  - → le cache
  - → Interruption (error)

Programmes : qui execute du code sur la machine.

CPU Modes : (x86)

- User mode : (~~ring~~ ring 3)
- Supervisor mode : Peut modifier les registres de contrôle. (ring 0)

Noyau de Système d'exploitation (kernel) : Abus de langage : kernel ≠ OS

Programme en mode superviseur. = privilegié

↳ Fourni interfaces aux programmes user. via les syscalls

En cas de syscall ⇒ context switch (changement de mode)

C = portable ⇒ peut pas faire de syscall.

↳ Utilisation d'un wrapper. (libc) ⟹ Specifique à chaque architecture.

4 cores ⇒ 25 programmes simultanée :

Ordonnanceur : donne un temps d'éxecution pour chaque processeur.

Mecanisme d'interruption : Permet au kernel de reprendre la main (ressource CPU)
(Timer : horloge quartz)

CPU :

Bus de données + d'address. Bus de contrôle (R or W)

Fichiers

↗ syscall ⇒ ↗ erreur ⇒ ↗ agressivité attaque.

↘ syscall ⇒ ↘ audit ⇒ ↗ safe

i-node / inode: contient toutes les infos d'un fichier
numero d'inode: l'adresse qui permet de retrouver le fichier sur le disque

inode:
* Emplacement
* Type de fichier
* Permissions, uid/gid
* Dates: creation / modif / etc
* size ...

path ← stat / fstat → fd

stat(path);
open(path); ← Pas de garanti que le fichier n'a pas été modifié.

stat: return struct stat {} voir fstat(2)

Types de fichier
→ Regulier → liens symbolique → socket
→ directory → Special files: char device, block device → named pipe (FIFO)

Getdents (2): read directory

Directory walk:
opendir / readdir / closedir (3)

Comment on execute un nouveau programmme ?
syscall fork()
↳ duplique le process courant

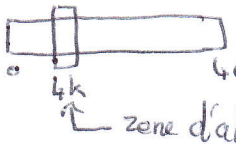
↳ fichier qui contient du code
process:
↳ programme qui s'execute (la plupart du temps)

syscall wait*(): wait(), wait_pid(), ...
↳ Attendre que le programme se finissent / interrompu.
syscall execve: creer un nouvel espace memoire dans le process courant
$ find . -type "*.pyc" -exec rm {} \;

```
syscall execve
pid_t pid = fork()
if (pid < 0)
    err()
if (pid)
    waitpid(pid /*...*/)
execvp("rm", ...)
```

```
* char *argv[] = {
        "rm",
        "toto.pyc",
        NULL,
};
execvp(argv[0], argv);
err(1, "error");
```

Espace d'adressage :

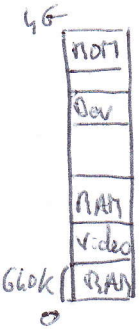


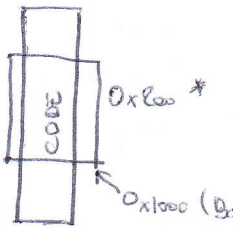
↳ zone d'allocation unitaire minimum

Tous les devices / CPU sont decrits par une adresse.

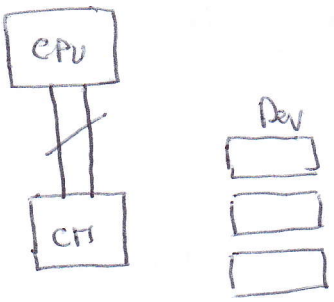
En cas de reset, le pointeur d'instruction monte dans la plus haute adresse.

Segment : (Base, Limite)
Adresse ↳ Offset de la dernière adresse

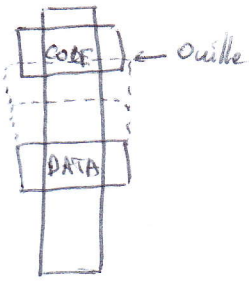




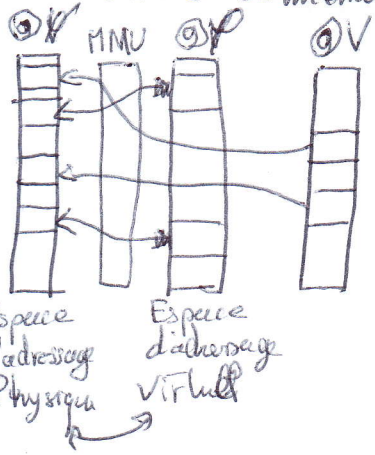
* 0x200 par rapport à la base



Contre exemple :



Virtualisation de mémoire (Pagination)



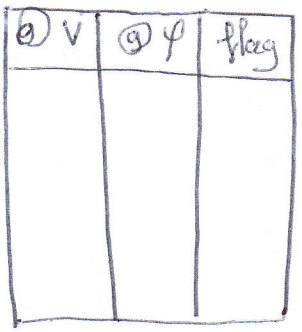
↓

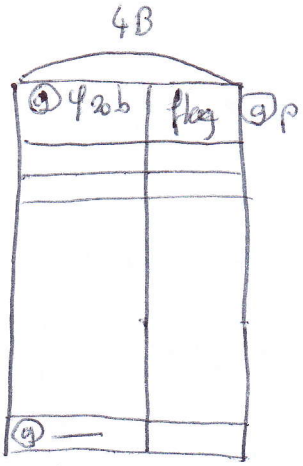
La mémoire est découpée en page de 4kb

Ⓖ page : 20 bits

Un espace d'adressage de 48 bits peut décrire 4 Gb.

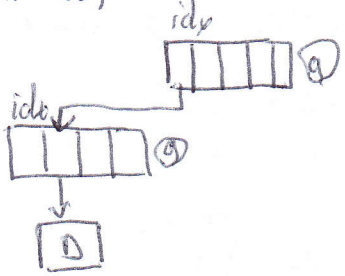
MMU : Memory Management Unit.
↳ S'occupe de la traduction entre l'espace Virtuel et Physique.

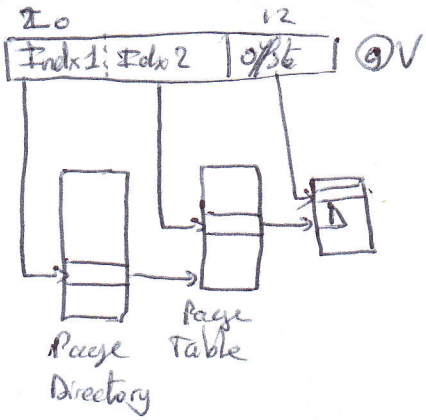




Stockée en mémoire contigue ↑
Les adresses sont aussi des Ⓖ φ

BTree : the solution :
(Alors Base c'est le stockage d'index dans les bases de données)





Process :

- Pid
- Uid / Gid : A qui appartient le process
- Adress Space : Memoire allouée
- fd table
- Signals | handlers
  | signaux à servir
  └ Erreur processeur SIGSEV

En cas de fail d'un appel syscall ⇒ Errno est set.

```
re = write(...);
perror();
```

EPERM : père mort

```
int handler(int R) {
    open(' ');
}
```

- Sauver la valeur d'errno
- Executer syscall
- Restaurer la valeur d'errno

Fd tables :

- fd : numero

Struct file * fd table [ ] : fd

```
cat > file
↓
fork
  close(1)
  open(file)
execvp
```

P1 (shell) :
0 : /dev/pts/0
1 : /dev/pts/0

```
fork()
P2 (shell)      ← new process
0:
1:
```

```
newfd = dup(oldfd)
cat > file
fork
  close(1)
  dup(open(file))
execvp(cat)
```

O_CLOEXEC : Persiste en cas de fork, mais est close en cas d'appel à un autre programme
fcntl ↕
↳ permet de set des propriétés des fichiers ouverts

```
dup2(oldfd, newfd);
≈ close(newfd)
  newfd = dup(oldfd)
```

Redirection Shell

```
cat > toto
≈ fd = open(toto)
  dup2(1, fd)
```

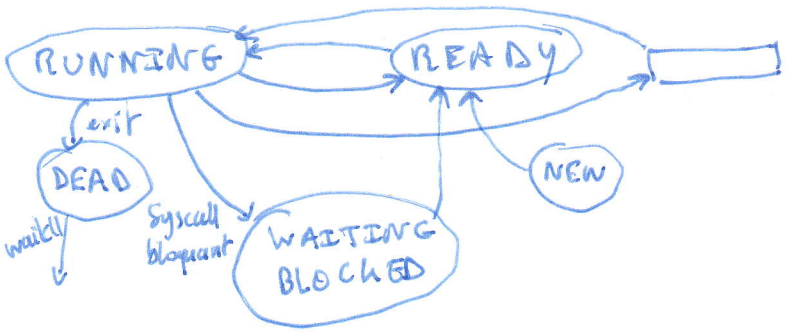
> vs >>
→ Option d'ouverture differentes.

```
cat | cat2
out →  ← in
pipe(int fd[2])
fd[0] r
fd[1] w
```

buf

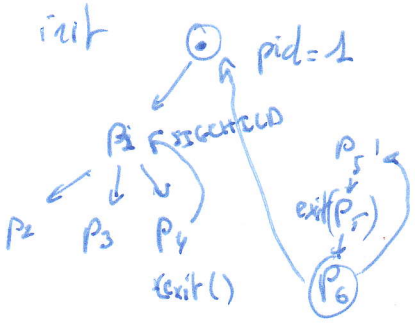
```
fd[0]   pipe[1]
fork()
dup2(fd[1], 1)
fork()
execvp(cat)
```

```
close(fd[0])
dup2(fd[1], 0)
execvp(cat2)
```



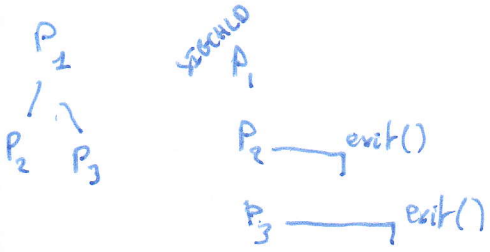
System ③

Syscall bloquant : Un syscall qui empêche de continuer à être running :
↳ wait ↳
↳ Sleep



SIGCHLD : Signal envoyé au père en cas d'exit d'un fils

```
signal (SIGCHLD, handler)
  handler () {
    wait(_,_);
  }
```




```
handler () {
  while () {
    wait (_, NOHANG)
    catch (ECHILD)
}}
```

Process :

uid/gid : id de l'user qui a lancé le process

bit setuid : 0777
u g o

euid/egid :

& bit setuid = 0 → uid/gid = euid/egid

1777 root :⇒ En cas d'exécution le programme n'aura pas mes droits mais celui du proprio (ici root)

Par ex : sudo est setuid 1 pour root.

ping :

créer une socket ICMP setuid root pour la création socket.

File systems:

→ READ (bloc)
→ WRITE (bloc)
→ SYNC : Commit le cache stp

Un accès au file system est considéré comme lourd.

Packet : bloc (= page pour la RAM)

→ size : { 512 B
2 KiB (cd-rom)
4 KiB

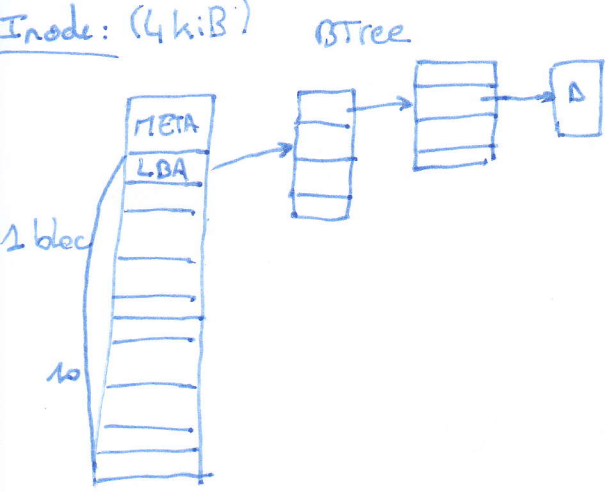
→ (q) → CHS : Cylinder Heads Sector
→ LBA : Logical Block Address

Améliorations : Ajout d'un cache

En cas de cache plein et d'un accès en lecture, il faudrait vider une partie du cache pour y mettre la ressource demandée.
Exemple d'Algo : LRU : Least Real Used
Le cache est adressé par le disque (pour éviter de les adresser en CHS)

Carte mémoire : A la construction, il est rare que tous les blocks fonctionnent. Pour éviter de jeter les block avec cellule défaillante, elles sont revendue avec une capacité annoncée plus faible

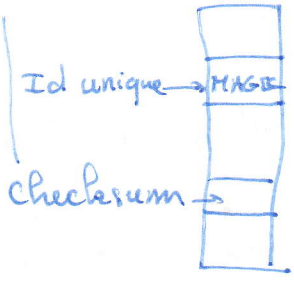
Inode : (4 KiB)



Organisation sur File System sont diverses et variées.
Seul différence : OS

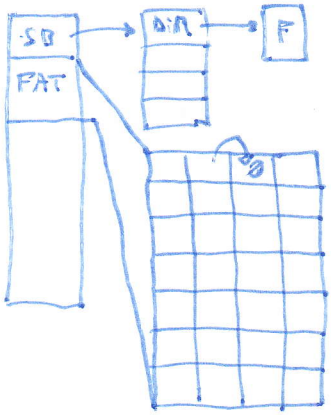
→ Table de correspondance n° Inode → (q)
→ BTree
→ Root Inode
→ Struct Superblock (point d'entrée du File System)

ACPI : Economie d'énergie et découverte du matériel



FAT 32 :
Taille fichier int => Max 4 GiB
Designé pour DOS
→ Mono utilisateur
→ 8B ~~nom~~ file name
→ 3B extension file

FAT peu de code (800 lignes)
EXT 2 (2000 lignes)
EXT 4 (100 000 lignes, gourmand, multi taches)



FAT : tableau avec plusieurs listes chaînées imbriquées.
Utile pour les fichiers qui font > 1 bloc

En cas de corruption, on corrompt 70% du FS