Architecture d'un système d'exploitation

Jean-Claude Bajard

IUT de Montpellier, Université Montpellier 2

Première année de DUT

Historique
Les premières années

- 1945-1955
  - Chargement manuel du programme
  - Machine à tubes (sans mémoire)
  - Exécution séquentielle d'un programme (instructions en binaire)
- 1955-1965
  - Apparition des transistors
  - Traitement par lots (cartes perforées): langage de programmation, interprèteur , compilateur,
  - Premiers programmes résidants: FMS(Fortran Moniteur System) IBSYS (IBM)

Historique
Les premiers OS (Operating Sytems)

- 1965-1980
  - Circuits intégrés, disques, bandes...
  - OS 360 d'IBM
  - temps partagé (cpu, lecture de données...)
  - multi-taches multi-utilisateurs (Multics, Unix)
  - Mini ordinateurs : DEC PDP 1, 7, 11
- 1980-1990
  - VLSI (Very Large Scale Integration), micro-ordinateurs
  - Systèmes interactifs, msdos, os mac, unix
  - Réseaux, systèmes distribués...

Définition

Operating System - Système d'Exploitation

- Un système d'exploitation est un ensemble de programmes qui réalisent l'interface entre les matériels (unité centrale, périphériques) et les utilisateurs.
- Notion de machine virtuelle sur la machine physique.
- Gestion du partage des ressources.

Couches fonctionnelles

Programmes utilisateurs
Programmes d'applications (éditeurs, tableurs,...)
Programmes systèmes (assembleur, compilateurs, chargeurs, ...)
Système d'exploitation (noyau, gestion des périphériques,...)
Instructuctions machine
Microprogrammation
Matériel

Les processus

- Processus: programme en cours d'exécution
  - Programme chargé en mémoire : segments code, données, pile, tas.
  - Compteur ordinal, exécution, ...
  - Environnement : registres, descripteurs de fichiers,...
- L'OS gère les différents processus en cours (table des processus, `/proc`):
  - Partage du CPU
  - Accès aux périphériques
  - Interruptions

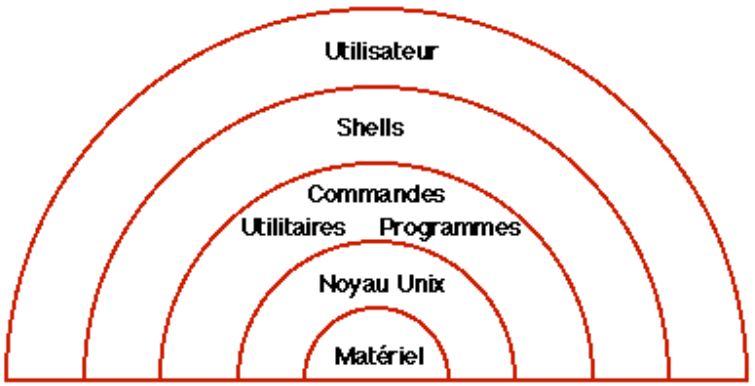
Le système de fichiers

- Organisation arborescente
  - Noeuds : les répertoires (fichiers particuliers) contenant des noms et les liens associés (adresse disque, inode, ...)
  - Feuilles : les fichiers sous différents formats (ascii, binaires,...),
- Droits d'accès `rwx` (lecture, écriture, exécution) exemple: droit d'écriture dans un répertoire = création-suppression de fichers
- Gestion des accès multiples à un fichier
- Fichier spéciaux : pipe, shared memory (shm), boîte à lettre, sémaphores...

La gestion de la mémoire

- Segmentation : segments de tailles variables identifiés code,données,pile-tas
- Pagination : mémoire adressée de façon linéaire, chargée par page de même taille au fur et à mesure des besoins
- Mémoire cache : mémoire interne au processeur, accès rapide
- Gestion de l'adressage de la mise en cache : adresse physique - adresse logique

Interpréteurs de commandes - shell



([1])

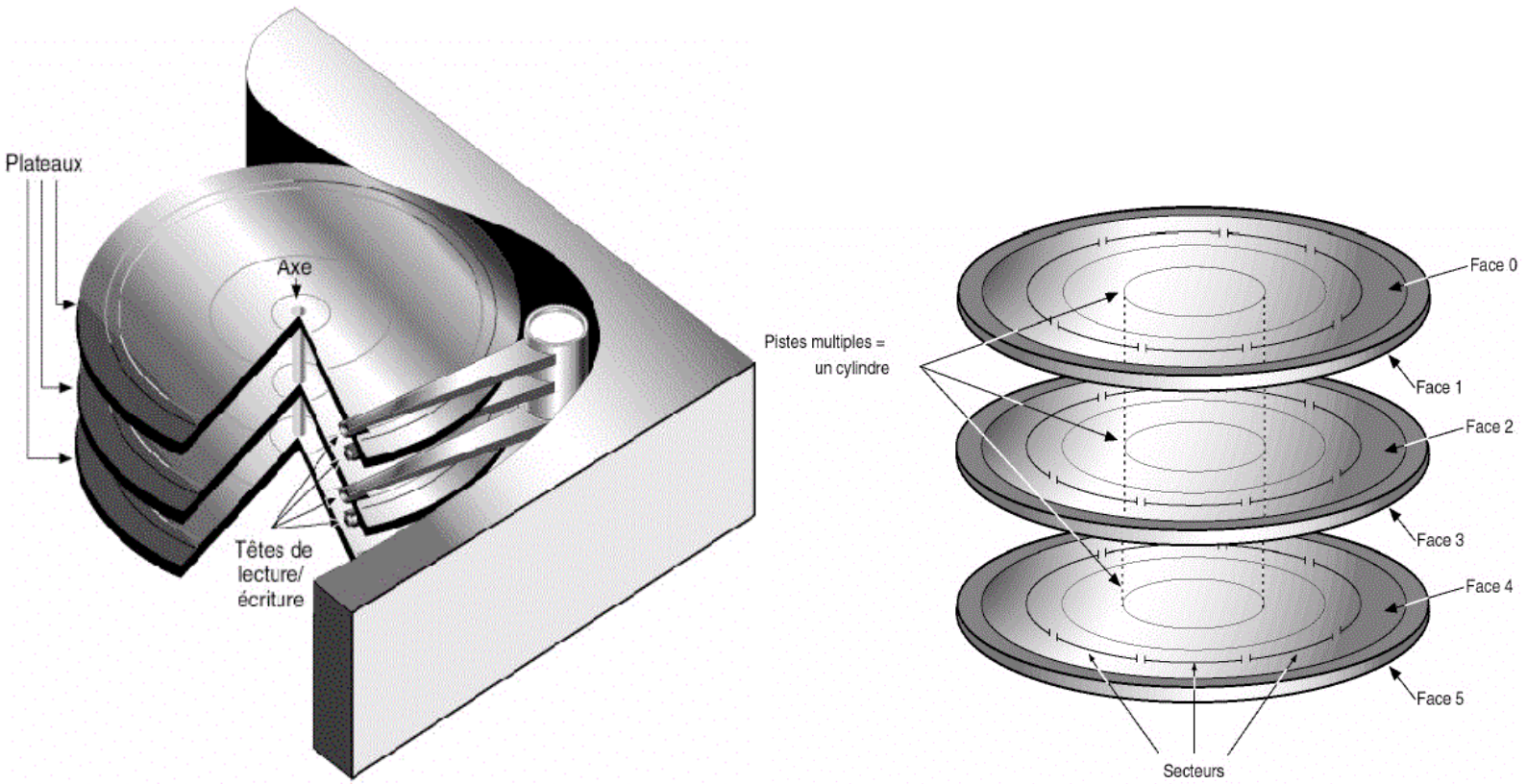
- Interface basique entre l'utilisateur et le système d'exploitation via un terminal (ou une fenêtre de ce type)
- Linux : différents shell possibles, le plus courant est le bash
- Notion de scripts: petits programmes interprétés, par exemple les scripts rc du démarrage

[1] Image http://enacit1.epfl.ch/guide_unix/introd_unixR.html

Partie 1 : Le Système de Fichiers (linux)

Formatage physique

Organisation du disque dur en cylindres - pistes - secteurs qui sont les points de repère d'adressage physique.



([2])

[2] http://www.infoprat.net/dossiers/materiel/article01/article01_centrale.htm

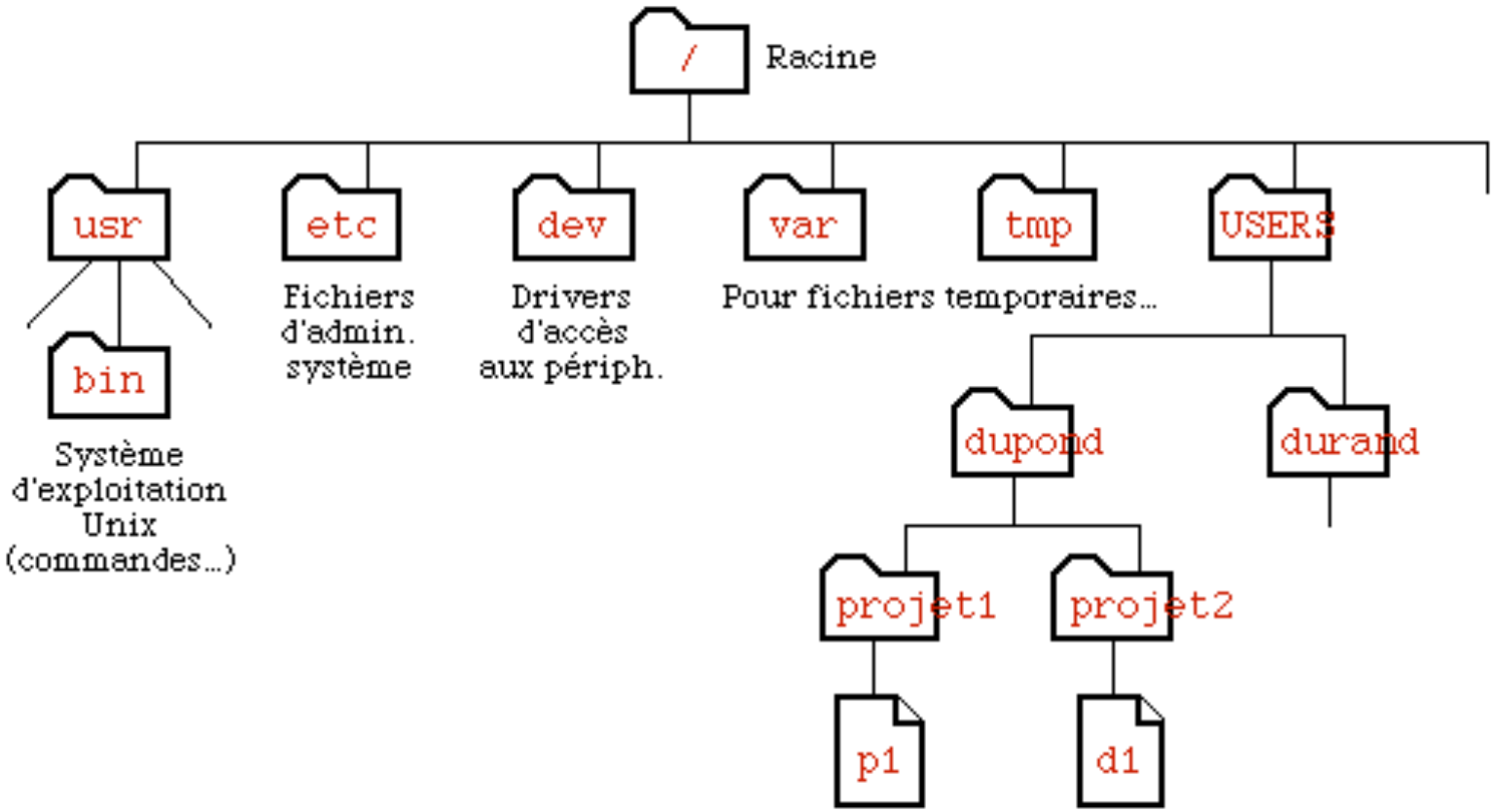
Partitions

- Décomposition d'un disque physique en plusieurs disques logiques (partitions principales, partitions étendues, groupes de cylindres)
- Master Boot Record : amorce de boot, organisation du disque (premier segment)
- Disques logiques :
  - `swap` mémoire virtuelle
  - Système de fichiers : organisation logique de l'arborescence, gestion des droits, gestion de l'espace libre (éviter la fragmentation,...)...
  - un OS peut gérer plusieurs FS exemples `fat32` `ext2` `ext3`....

Particularités UNIX

- Une arborescence unique regroupant tous les périphériques (`/dev`)
- Principe de montage local ou distant (NFS).
- Notion de chemin : soit à partir de la racine "/" "absolu", soit à partir du répertoire courant "relatif"
- Répertoires : noeuds de l'arborescence
- Fichiers : feuille
- Fichiers spéciaux pour les périphériques.

Principaux répertoires de l'arborescence UNIX



([3])

[3] Image http://enacit1.epfl.ch/guide_unix/introd_unixR.html

Principaux répertoires de l'arborescence UNIX

Souvent sur partitions propres

- `/` contient les outils et fichiers de configuration vitaux pour le système
- `/usr` toutes les applications dont celles utilisateurs
- `/var` en particulier les fichiers de tailles variables : `spool` (mail, imprimantes,...) `log`,....
- `/home` les répertoires des comptes utilisateurs

La Racine /

Principaux répertoires de l'arborescence UNIX

- `/bin` commandes élémentaires (`ls, rm, bash`,...)
- `/sbin` commandes administration (`showmount`,...)
- `/boot` noyau
- `/dev` fichiers spéciaux des périphériques (son, disques, cd,...)
- `/etc` fichiers de configuration (`rc,passwd,fstab`,...)
- `/lib` librairies partagées du noyau
- `/proc` les processus
- `/tmp` les fichiers temporaires

Les applications `/usr`

Principaux répertoires de l'arborescence UNIX

- En général sur partition propre (commande `df` et `mount` pour voir les montages)
- Nous retrouvons des répertoires semblables à certains de la racine `/usr/bin`, `/usr/lib`, `/usr/sbin` pour des commandes et utilitaires non vitaux
- Autres répertoires : `/usr/include` pour les entêtes de fonctions partagées, `/usr/X11R6` pour l'environnement X11, `/usr/man` pour le manuel

Les variables /var

Principaux répertoires de l'arborescence UNIX

- /var/spool/mail messagerie électronique, un fichier par utilisateur
- /var/spool/cups files d'attente des imprimantes
- /var/nis ou /var/yp gestion des pages jaunes NIS
- /var/cron scripts des crontab pour des actions régulières (ex. sauvegardes,...)
- /var/log fichiers traces

Gestion des droits UNIX

Principaux propriétaires

- `u` "user" utilisateur, soit lié au compte d'une personne physique, soit lié au compte d'une application (voir `/etc/passwd`)
- `g` groupe, un groupe de personnes (année1,...), un groupe pour une application ou un périphérique (carte son, cd,...)
- `o` les autres (ni u, ni g)
- `a` tout le monde

Types de droits
Gestion des droits UNIX

- `r` lecture : fichier `cat`, `more`, `cp`, `read()`, editeurs (en lecture), répertoire `ls`
- `w` écriture : fichier éditeurs, `write()`, répertoire rm, mv, cp, mkdir, rmdir,
- `x` exécution : fichier instructions machine, scripts shell, répertoire cd

Pour visualiser les droits de fichiers d'un répertoire : `ls -l`

```
drwxr-xr-x 63 fernand ann1 2142 Apr 22 10:39 TeX
-rw-r--r-- 1 fernand ann1 6430 Sep 11 16:43 toto.c
```

Modification des droits : commande chmod

Gestion des droits UNIX

- Utilisation :
  - `chmod u=rwx,g=rx,o= toto.sh`
  - en octal avec r=4,w=2,x=1 : `chmod 750 toto.sh`
  - `chmod u-w toto.sh`
- Droits particuliers :
  - setuid : (fichier) permet l'exécution avec les droit du propriétaire du l'exécutable et non pas ceux de l'exécutant. `chmod u+s toto.sh`, ex. la commande `passwd`
  - setgid : (fichier) idem mais au niveau groupe `chmod u+s toto.sh`
  - sticky bit : (répertoire) donne le droit de création à tous (si rwx pour tous) mais de suppressions et le modification unique au propriétaire du fichier (non à celui du répertoire qui est le cas usuel) `chmod o+t Test` ex. le répertoire `/tmp`

Commandes utiles
Gestion des droits UNIX

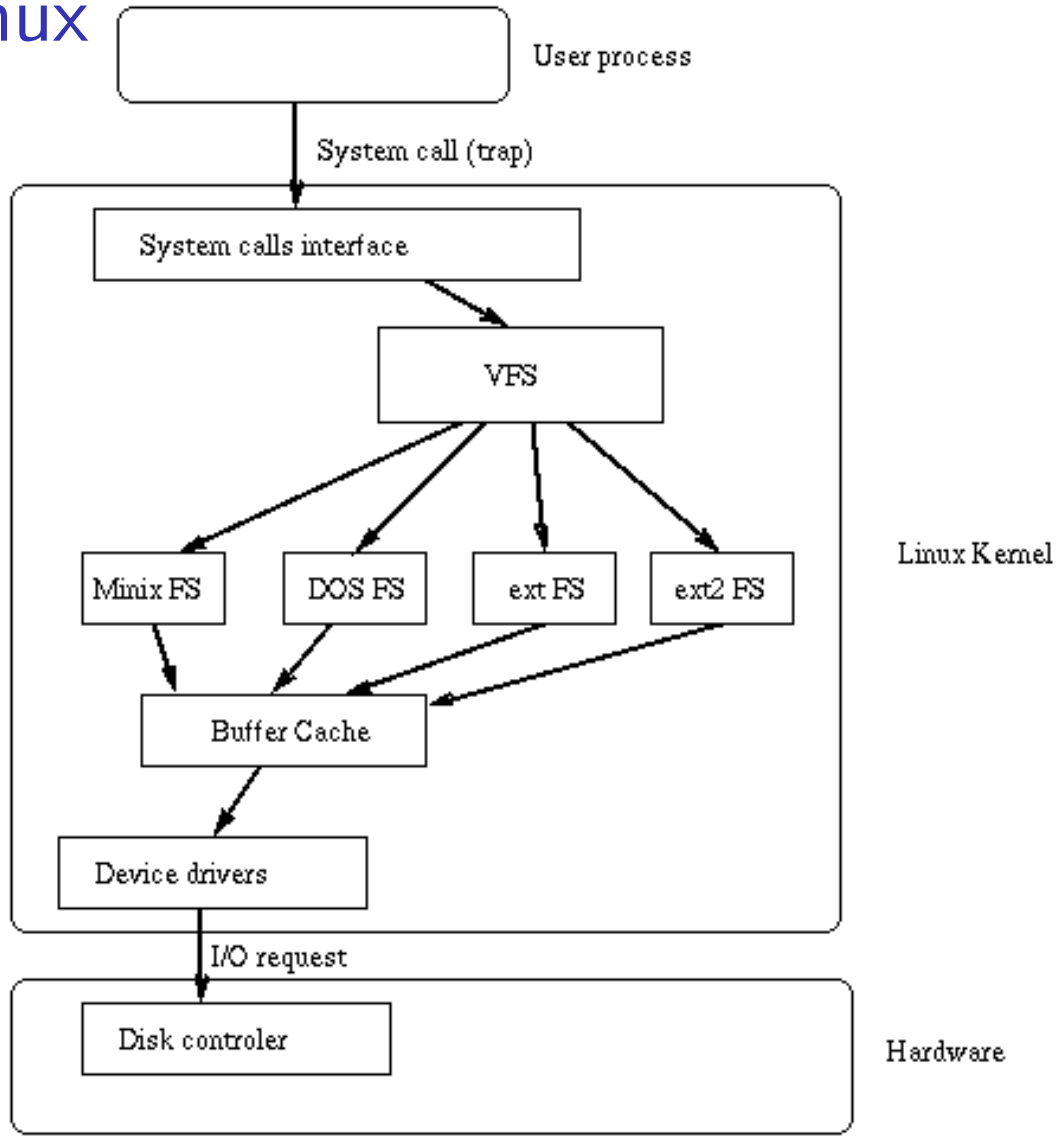
- `chown` changer le proriétaire et le groupe d'un fichier ou répertoire (l'option -R permet de traiter la sous arborescence)
- `umask` affecter les droits par défaut. umask 022 masque les droits w d'écriture du groupe et des autres
- `chgrp` changer le groupe d'une fichier ou répertoire
- `newgrp` changer le groupe d'un utilisateur

Gestion des FS sous Linux

- Virtual File System : Linux via un système virtuel, gèrent plusieurs type de FS; fat32,ntfs,ext2,ext3,nsf,...
- ext2 et ext3 systèmes étendus de linux
- ext3 version avec un système de journalisation simplifiant la vérification de l'intégrité du système (`fsck`)

Gestion des FS sous Linux

Virtual File System



[4] http://web.mit.edu/tytso/www/linux/ext2intro.html

Structure de Ext2

Organisation de la partition

- Division en groupes de blocs

boot.sector 1Ko	groupe 0	groupe 1		groupe n

- Chaque groupe comporte

Super Block	Group Descriptor	Block bitmap	Inode bitmap	Inode Table	Data Blocks

Structure de Ext2
Organisation de la partition

- Super-Block: Informations du FS (nb total d'inode, de blocs, nb par groupe,...)
- Group-Descriptor: Informations du groupe (adresses des blocs suivants, BB IB IT et DT, espace libre,..)
- Block-Bitmap: statut de chaque bloc, 0-libre 1-utilisé
- Inode-Bitmap: idem pour les inodes,
- Inode-Table: propre à chaque groupe, un inode définit un fichier physique
- Data-Block: les blocs de données

Structure de Ext2

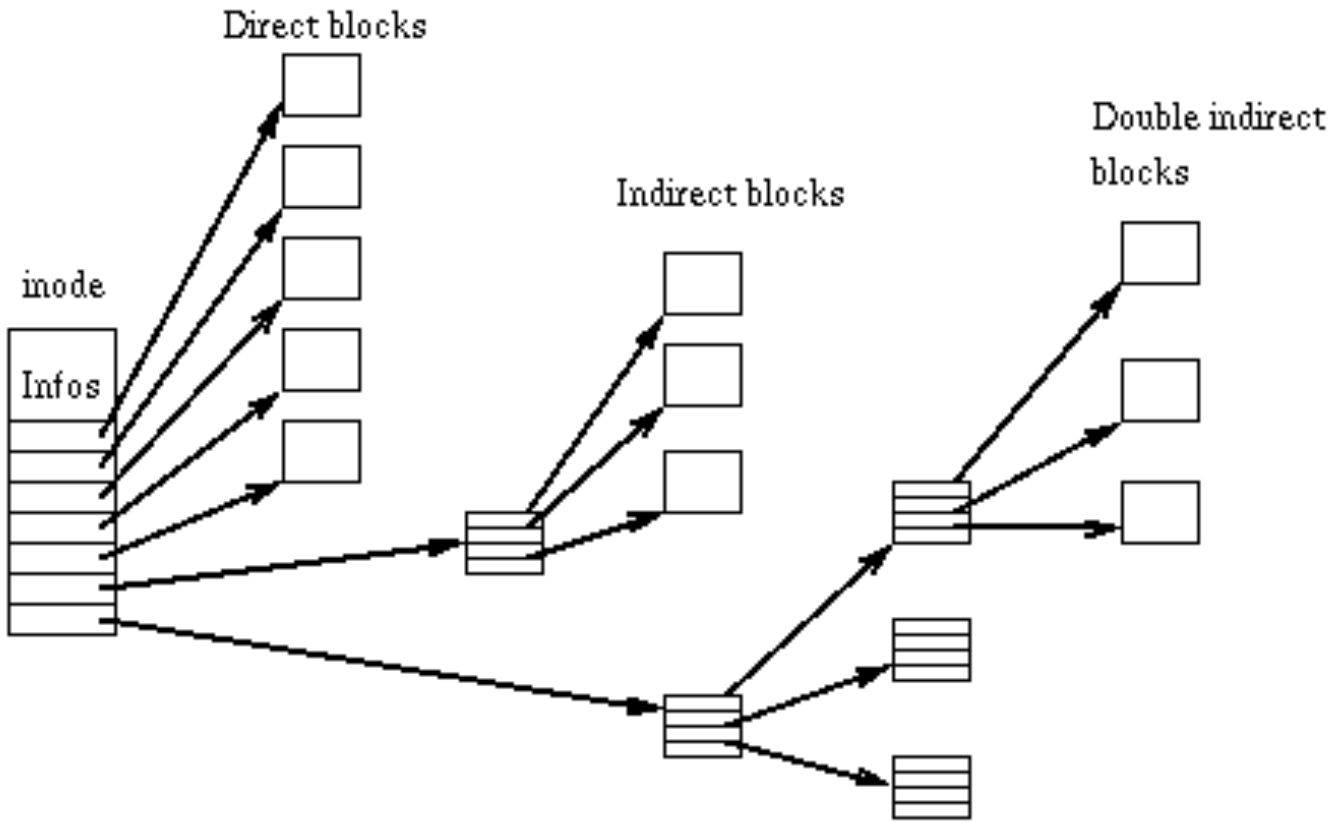
Organisation d'un inode

	nb octets	commentaires
i_mode	2	type(4b) : 0 unknown,1 regular-file, 2 dir,3 c-dev,4 b-dev,5 fifo,6 sock,7 lien droits (12b): rwx+st pour u,g,o
i_uid	2	ID du propriétaire
i_size	4	taille en octets
i_?time	4	a accès, c création, m modification
i_gid	2	ID du groupe
i_links_count	2	nombre de liens
i_blocks	4	nombre de blocs
i_flags	4	ouverture, lecture, écriture
i_block[15]	4	15 @, 12 dir., 1 indir., 1 double, 1 triple

Structure de Ext2

Organisation d'un inode

Si bloc = 1024 octets = 256 @ alors taille max = 2^{34} octets



5

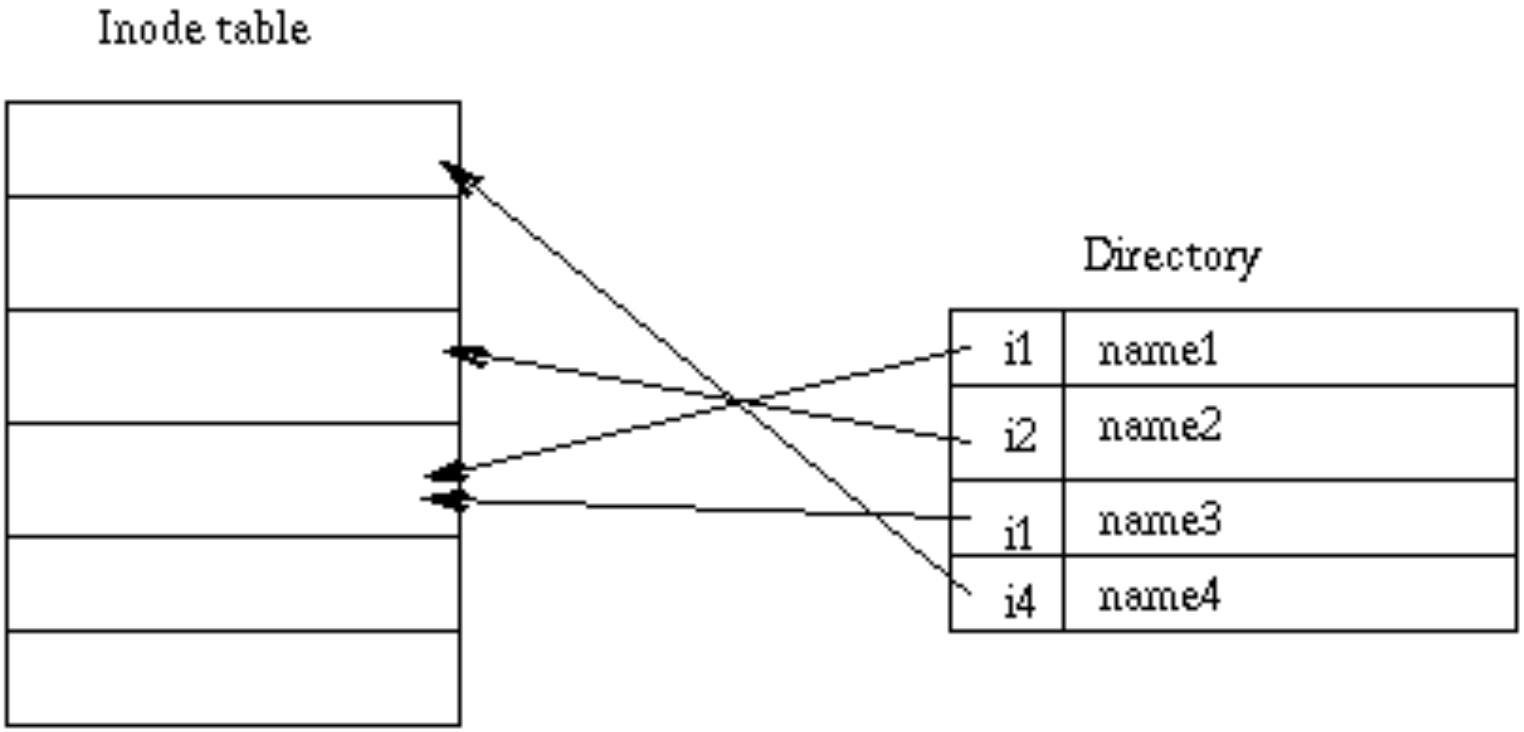
[5] http://web.mit.edu/tytso/www/linux/ext2intro.html

Structure de Ext2
Organisation de la partition

- `mke2fs` : créer un FS de type ext2
- `dumpe2fs` : lire les informations sur un FS ext2, exemple:
  - Inode count: 50400
  - Block count: 200812
  - Block size: 1024
  - Blocks per group: 8192
  - Inodes per group: 2016
  - Inode blocks per group: 252
  - Inode size: 128
- `/proc/partitions`

Structure de Ext2

Répertoires et liens



6

[6] http://web.mit.edu/tytso/www/linux/ext2intro.html

Structure de Ext2

Répertoires et liens

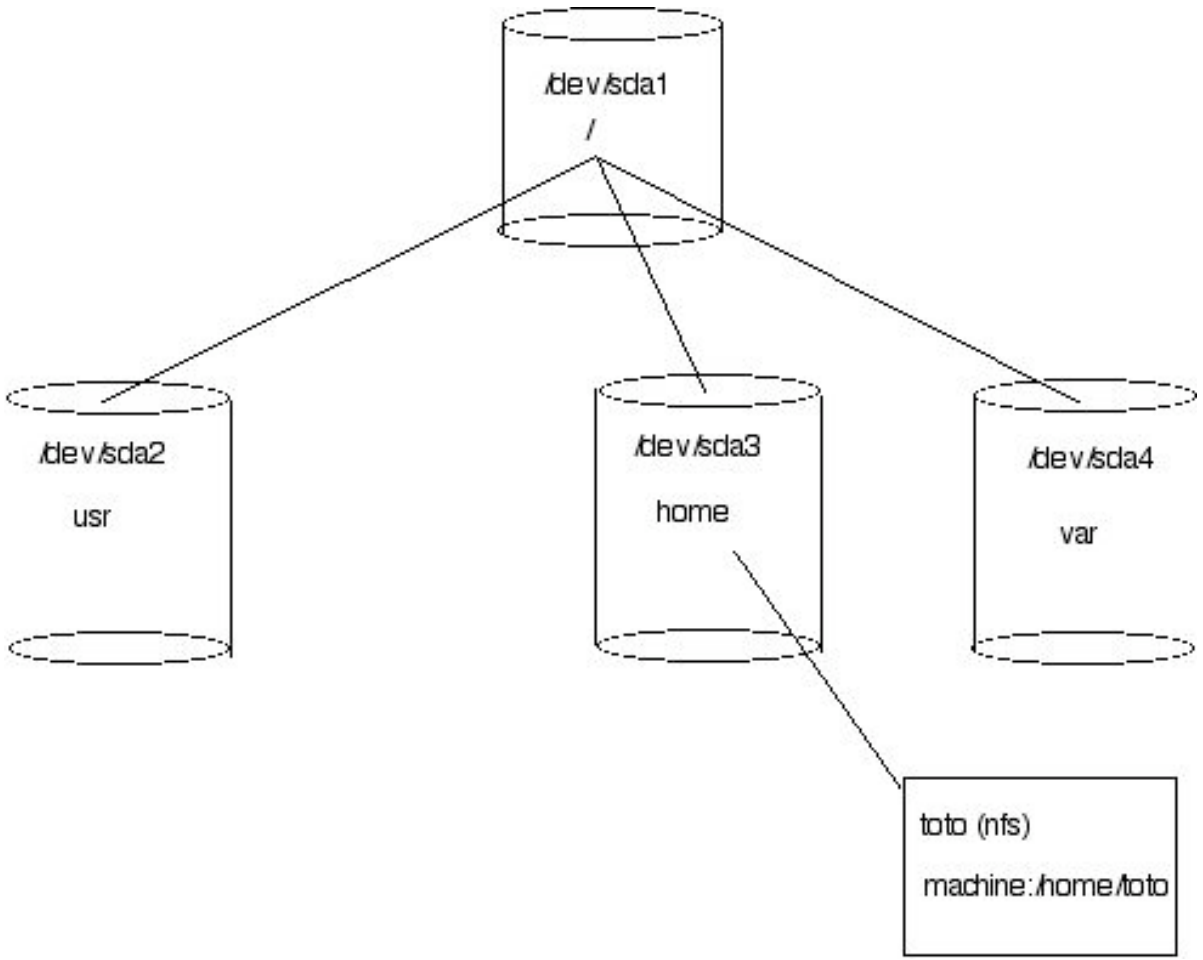
- Répertoire : fichier suite d'enregistrements de 4 champs
  - `inode` : numéro de l'inode
  - `rec_len` : longueur de l'enregistrement
  - `name_len` : longueur du nom
  - `name[256]` : le nom
- `ls -i` pour voir les inodes, `ls -l` affiche des info de l'inode
- Lien physique : faire pointer u nom sur un inode (sur un même FS)

  ```
  ln source cible
  ```
- Lien symbolique : créer un fichier qui contient le chemin ver un autre, (les FS peuvent être différents)

  ```
  ln -s source cible
  ```

Structure de Ext2

Répertoires et montages



Structure de Ext2

Répertoires et montages

- Commandes utiles :
  - `mount` permet d'effectuer et de visualiser les montages
    - locaux : type de FS, périphérique, point de montage
    - distant : type NFS (Network File System), serveur, le répertoire distant, point de montage

    (la commande `df` permet de voir les montages et l'occupation)
  - `exportfs` : gestion des répertoires accessibles de l'extérieur
  - `showmount` : donne les informations d'un serveur NFS
- Fichiers utiles:
  - `/etc/fstab` : montages par défauts
  - `/etc/mtab` : montages visibles en cours
  - `/proc/mounts`: montages en cours effectifs

Gestion des FS par l'OS
Virtual File System

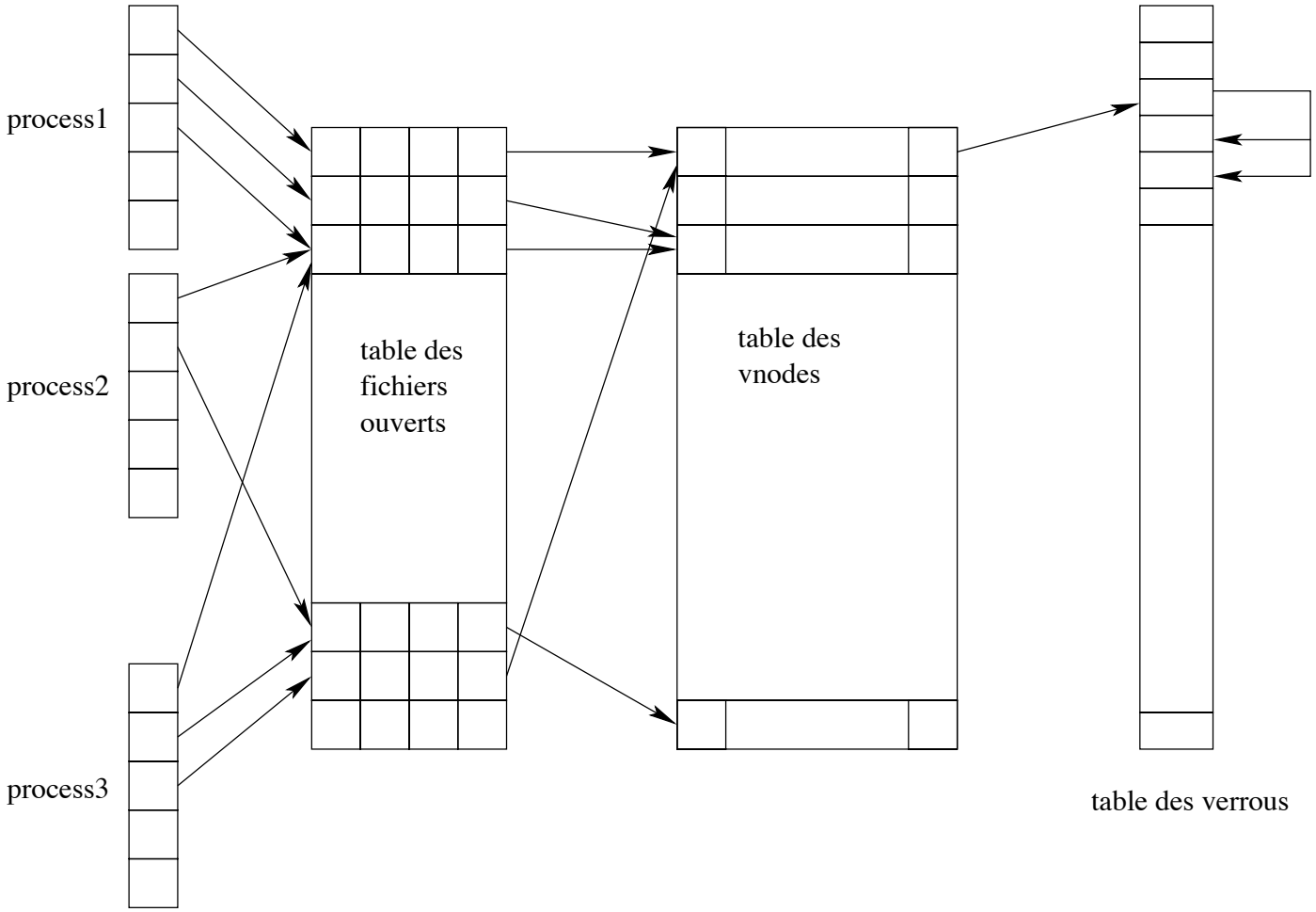
- Table de descripteurs de fichiers ouverts par un processus: une table par processus
  - les fonctions de bases sont : `open()`, `fopen()`, `diropen()` elles renvoient entre autre un numéro d'entrée dans la table (0=stdin, 1=stdout, 2=stderr)
  - un descripteur contient un pointeur sur un élément de la table des fichiers ouverts par le système
- Le système gère la table des fichiers ouverts.
  - Plusieurs processus peuvent avoir ouvert un même fichier. (ex. `stdout`)
  - Une entrée de cette table comprend : nb de descripteurs, mode d'ouverture, position courante, pointeur sur l'inode mémoire ou vnode.

Gestion des FS par l'OS
Virtual File System

- Table des vnodes liens vers les fichiers physiques: un vnode comporte
  - nb d'ouverture
  - ID du disque
  - vecteur de fonctions
  - des informations de l'inode
- Transferts :
  - mode bloc : (cas des disques), utilisation d'une mémoire tampon (buffer cache), accès en deux temps (vérifiaction si dans le cache sinon chargement dans le cache, puis action) Attention : actions sur le cache, modification du fichier physique à la fermeture (`close()` ou `fclose()`),ou si appel à synchronisation (`sync()` ou `fsync()`)
  - mode caractère : ecriture lecture directes, ex. carte son,...

Gestion des FS par l'OS

Virtual File System



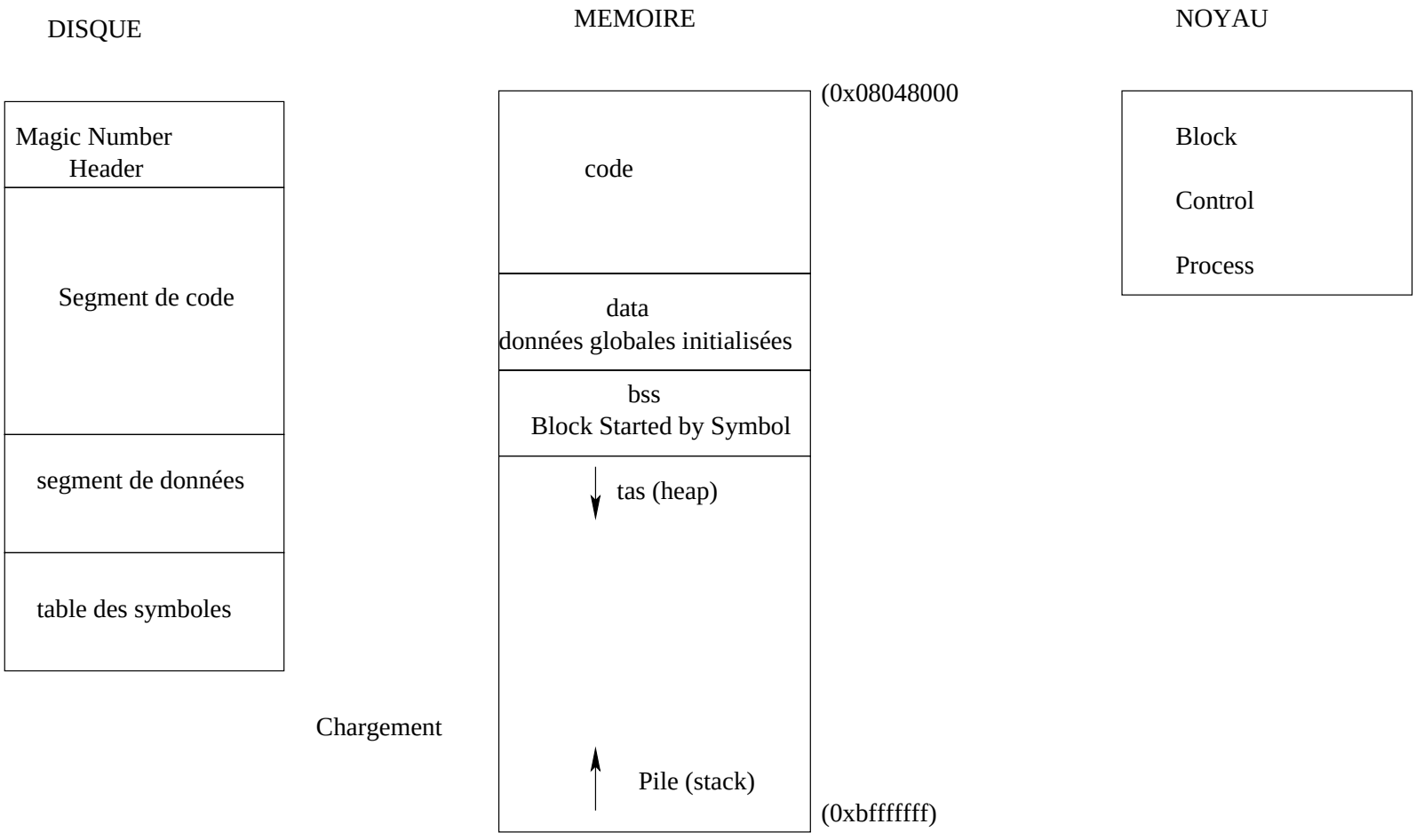
Tables des descripteurs

Partie 2 : Gestion des Processus (linux)

Les grands principes
Définitions

- Processus : programme en cours d'exécution :
  - compteur ordinal
  - registre
  - environnement mémoire (code, data, BSS, Heap, Stack)
  - table de descripteurs de fichiers

Les grands principes
Définitions



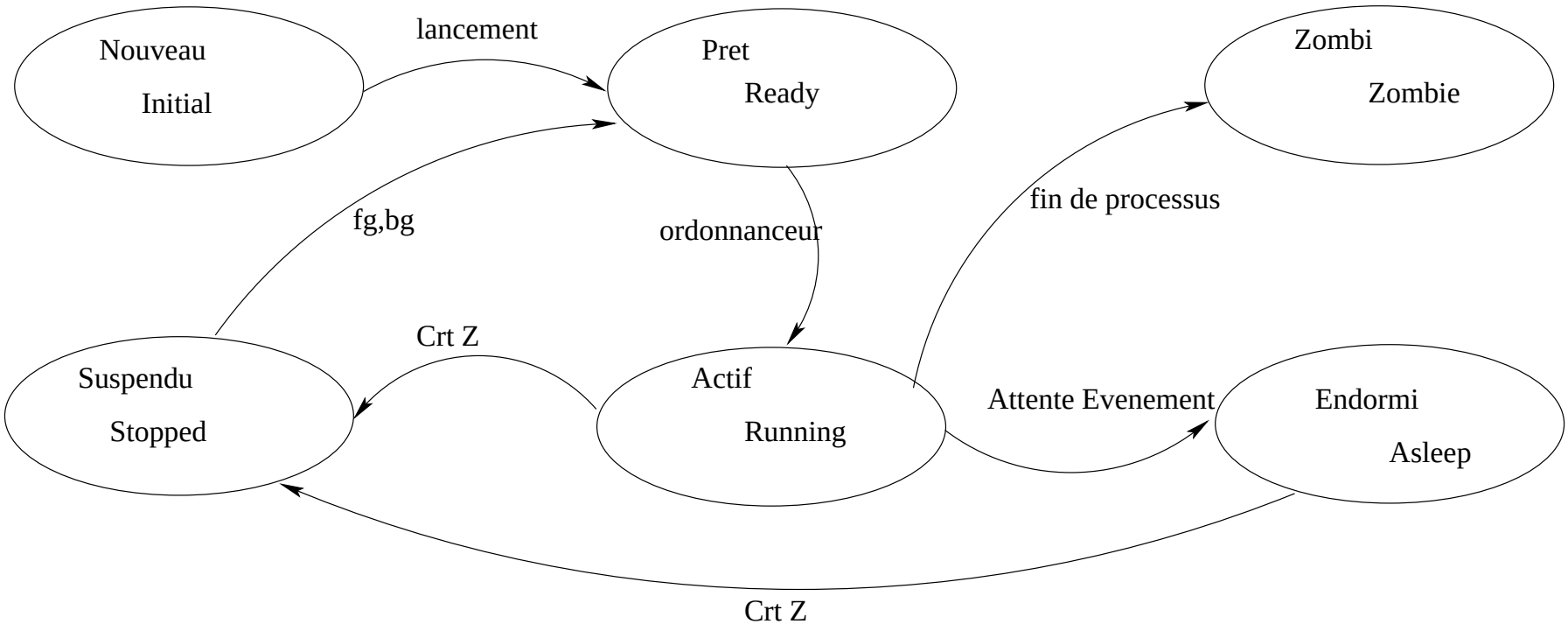
Les grands principes
Utilitaires

- La commande `size` donne des informations sur la structure de l'exécutable
- `file` donne le type d'un fichier
- `readelf` nombreuses informations pour les fichiers exécutables de type ELF "Executable and Linkable Format"

Les grands principes
Attributs des processus

- `pid` identifiant du processus
- `ppid` identifiant du processus père
- `uid` propriétaire réel qui lance le processus
- `euid` propriétaire effectif de l'exécutable (`setuid`)
- `gid,egid` identifiants du groupe
- session, groupe de processus, terminal
- temps cpu (`<sys/times.h>`)
- état du processus
- priorités : nice (-20 le plus à 20 le moins, par défaut 0)

Les grands principes
les changements états



Les grands principes
les commandes de gestion des processus

- `top` permet de visualiser l'utilisation des ressources par les processus (attention cette commande est elle même gourmande)
- `ps` affichage des processus, nombreuses options (exemples : `ps -aux` ou `ps -alx` "vision large"
- `kill` permet d'envoyer un signal à un processus, en général de terminaison (exemples : `kill -TERM pid` ou `kill -15 pid` pour une fin avec fermeture des fichiers, `kill -KILL pid` ou `kill -9 pid` pour une fin sans précautions)

Les grands principes

Pour les processus d'un terminal

- Le lancement dans un `shell` entraîne la suspension du `shell`
- Notion d'avant plan *foreground (fg)* et d'arrière plan *background (bg)*
  - lancement en arrière plan : `toto &`
  - sinon contrôle-z suspend le processus et permet de reprendre le `shell` dans ce cas possibilité de reprise du processus avec *fg* et *bg*
- Notion de job : ensemble de processus d'une même tâche.

Les grands principes

Redirection des entrées et sorties standards

- | redirection de la sortie standard vers l'entrée standard.
- Exemple : `a.out | grep toto` un seul job avec la commande `jobs` mais deux processus pour la commande `ps`.
- < redirection de l'entrée standard, lecture d'un fichier.
- > redirection dans un fichier de la sortie standard (écrasement) , >> redirection à la fin du fichier.
- 2 > redirection des erreurs
- Exemple : `find /etc -name "group*" -print 2> erreur.trace > resultat.trace`

Création de processus
Principe du fork

- L'appel de la fonction `fork()` crée un processus fils copie du père (le processus appelant) au moment de l'appel.
- Cette fonction retourne :
  - le `pid` du fils pour le processus père
  - 0 au processus fils créé.
  - -1 en cas d'echec.

Création de processus

Exemple de fork : code

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
int main(int argc, char **argv)
{
int i,j,n;
printf("[processus %d] je suis avant le fork\n", getpid());
i = fork();
if (i != 0) /* i != 0 seulement pour le pere */
    printf("pere pid =%d retour fork = %d \n",getpid(),i);
    else
    printf("fils pid =%d retour fork = %d \n",getpid(),i);
printf("[processus %d] je suis apres le fork \n", getpid());
}
```

Création de processus

Exemple de fork : exécution

pere pid =473 retour fork = 474
fils pid =474 retour fork = 0
%ps -l
UID PID PPID STAT TT TIME COMMAND
503 256 255 S p1 0:00.35 -bash
503 473 256 S p1 0:00.01 ./a.out
503 474 473 S p1 0:00.00 ./a.out
[processus 474] je suis apres le fork
[processus 473] je suis apres le fork

Création de processus

Quelques remarques

- Attention le père peut terminer avant ses fils et mourir.
- Par contre un fils qui a terminé devient zombi.
- Pour temporiser : `sleep(int s)`, `usleep(int micros)`, `nanosleep(int nanos)`.

```
$ ./fork1
[processus 428] je suis avant le fork
pere pid =428 retour fork = 429
fils pid =429 retour fork = 0
[processus 428] je suis apres le fork
[processus 429] je suis apres le fork
```

Gestion des processus

Signaux de fin d'exécution : `wait()` et `exit()`

- pid_t wait(int *status); (#include<sys/types.h> et #include <sys/wait.h>)
  - Attente d'un signal, en général de fin de processus fils
  - Renvoie le pid du processus fils ou -1 si erreur.
  - status récupère la valeur donnée par le fils avec l'appel à `exit()`.
- void exit(int status); (#include <stdlib.h>) sortie de processus avec envoi d'un signal
- status :

exit / wait	octet3	octet2	octet1	octet0
	?	?	octet0	signal

signal = 0 si exit() sans erreur, sinon signal de kill.

Gestion des processus
Fin d'exécution

- pid_t waitpid(pid_t pid,int *status,int option);
  (#include<sys/types.h> et #include <sys/wait.h>)
  - Attente d'un signal venant d'un processus précis
  - Option : WNOHANG (non bloquant), WUNTRACED (bloquant)
- fonctionnement :
  - si pas de signal, continue, waitpid renvoie 0 si le processus pid existe
  - si signal, octet de status et pid pour confirmation
  - si pas de processus pid alors renvoie -1

Gestion des processus

Exemple de waitpid

```
void f(void)
    { printf("fonction f pid %d \n",getpid());while(1); }
int main(int argc, char **argv)
    { int i,j,n=3,m,b=1; int tabpid[20];
     for(j=0;j<n;j++)
    { tabpid[j] = fork();
     if (tabpid[j] != 0 )
     printf("pere pid =%d retour fork = %d \n",getpid(),tabpid[j]);
     else {f(); exit(j); /* le exit sert a sortir de la boucle for */ } }
     while(b == 1) {b=0;      for(j=0;j<n;j++)
     { i= waitpid(tabpid[j],&m,WNOHANG);
     if( i >= 0) b = 1;
     if( i > 0 ) printf("pid %d etat %x \n",i,m);sleep(1); } } }
```

Gestion des processus
Envoi de signaux : commande kill

- La commande kill permet de terminer un processus avec l'envoi d'un signal
- status récupère le numéro du signal dans son premier octet.
- Un processus fils terminé devient zombi jusquà la lecture par le père du signal.

Gestion des processus

Envoi de signaux : commande kill

JCB@atgc/Prog 57 % a.out
processus 14301 je suis avant le fork
pere pid =14301 retour fork = 14302
pere pid =14301 retour fork = 14303
pere pid =14301 retour fork = 14304

pid 14303 etat b

pid 14304 etat f
Ctr-C

JCB@atgc/Prog 44 % ps xl
14301 12632 S pts/0 0:00 a.out
14302 14301 R pts/0 0:13 a.out
14303 14301 R pts/0 0:10 a.out
14304 14301 R pts/0 0:16 a.out
JCB@atgc/Prog 45 % kill -11 14303

JCB@atgc/Prog 49 % kill -15 14304

Héritage et fork
Principes

Lors d'un fork les processus fils héritent

- d'une copie des variables du père
- d'une copie de la table des descripteurs de fichiers
  tout fichier ouvert par le père sont accessibles
- d'une copie du code
  les fils ont le même code que le père.

Remarque: sur les fichiers, il peut y avoir des effets de bords (voir TD)

Héritage et fork
Exemple

```
#include <unistd.h>
#include <stdio.h>
int main(int argc, char **argv)
{ int i,j; int n;
printf("[%d] avant le fork\n", getpid());
i = fork();
n=5; if (i == 0) n=n+2;
for(j=1;j<n;j++){
printf("pid =%d j = %d \n",getpid(),j);
sleep(1);}
}
```

```
mourvedre$ ./a.out
[619] avant le fork
pid =619 j = 1
pid =619 j = 2
pid =619 j = 3
pid =620 j = 1
pid =619 j = 4
pid =620 j = 2
mourvedre$ pid =620 j = 3
pid =620 j = 4
pid =620 j = 5
pid =620 j = 6
```

Recouvrement
Commandes exec...

- Remplacement du code d'un processus en cours par celui d'une exécutable qui prend son pid
- Famille de commande commençant par `exec...`
- execve() commande de base

NAME
execve – execute a file
SYNOPSIS
#include <unistd.h>
int execve(const char *path, char *const argv[], char *const envp[]);
DESCRIPTION
Execve() transforms the calling process into a new process.

Recouvrement

Exemples: exec...

commande execv

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
char *arg[3];
arg[0] = "ps";
arg[1] ="-aux";
arg[2] =NULL;
execv("/bin/ps",arg);
fprintf(stderr,"erreur dans execv");
exit(EXIT_FAILURE);}
```

commande execlp

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
int main(int argc, char *argv[]){
char *arg[3];
execlp("ps","ps","-aux",NULL);
fprintf(stderr,"erreur dans execlp");
exit(EXIT_FAILURE);}
```

Partie 3 :Tubes de communication (linux)

Les grands principes
Fichier virtuel fifo

- Communication entre deux processus via un fichier virtuel de type fifo
- Communication unidirectionnelle
  in (1) $\rightarrow$ ()$\overline{\qquad\qquad\qquad\qquad\qquad\qquad}$) $\rightarrow$ out (0)
- Créer par et pour un processus et sa descendance
- Ecriture côté (1), Lecture côté (2), principe fifo (first in first out), lecteur-écrivain
- La lecture est destructrice
- Problème de taille maxi (tube plein)
- Exemple de redicrection par tube: du -s * | sort -n

Les tubes anonymes

La commande pipe()

- #include <unistd.h>
  int pipe(int *tuyau);
- tuyau est un tableau de deux entiers : int tuyau[2];
- int tuyau[1]; est l'entrée du tube, écriture (stdout=1)
  int tuyau[0]; est la sortie du tube, lecture (stdin=0)
- la commande pipe() renvoie : 0 si succès, -1 si échec.

Les tubes anonymes
Mise en place

- Pour une prise en compte par les fils, pipe() doit être lancé avant fork()
  - Création du tube
  - Appel fork()
- Pour un échange Père− >fils
  - le père ferme la lecture: close(tuyau[0])
  - le fils ferme l'écriture: close(tuyau[1])
- Pour un dialogue création de deux tubes

Les tubes anonymes

Lecture : read()

```
#include <sys/types.h>
#include <sys/uio.h>
#include <unistd.h>
ssize_t read(int d, void *buf, size_t nbytes);
```

- Si le tube n'est pas vide : place dans buf , nbytes du tuyau d
- Si le tube est vide:
  - si pas d'écrivain (aucun processus avec tuyau[1] ouvert) renvoie 0
  - Sinon
    - si lecture bloquante alors suspension des processus
    - sinon, retourne -1

Les tubes anonymes
Lecture (suite)

- Remarque pour non blocage:

```
int attributs,tuyau[2];
attributs=fcntl(tuyau[0],F_GETFL);
fcntl(tuyau[0],F_SETFL, attributs | O_NONBLOCK);
```

Les tubes anonymes

Ecriture

ssize_t write(int d, const void *buf, size_t nbytes);

- Si pas de lecteur : envoi de SIGPIPE, fin de processus, "broken pipe"
- Sinon
  - Si écriture bloquante : retour après écriture des nbytes de buf dans tuyau d, si plus d'ocets que PIPE_BUF attente de lecture pour fin écriture.
  - Si écriture non bloquante :
    - si pas assez de place erreur retour -1
    - sinon écriture

Les tubes anonymes

Inter-blocages : exemple

```
i = fork();
if (i==0) {
    read(tube1[0],mot1,80);        attente de caractères dans tube1
    write(tube2[1],mot1,80); }
else {
    read(tube2[0],mot1,80);        attente de caractères dans tube2
    write(tube1[1],mot1,80); }
```

Les tubes anonymes

Pointeur de type *FILE

- Le type *FILE permet l'appel de fonctions formatées comme fprintf, fscanf,...
- La fonction fdopen permet d'associer ce type de pointeur à un tube
- `FILE * fdopen(int fildes, const char *mode);`
  - int tube[2]; FILE *f,*g;
    pipe(tube);
    f = fdopen(tube[1],"w"); écriture
    g = fdopen(tube[0],"r"); lecture

Les tubes nommées

Le principe

- Même principe que le tube anonyme mais pour des processus sans lien de parenté
- Nécessité de référencer le fifo dans le système de fichier
- Tout processus peut utiliser ce fichier suivant les droits "utilisateur"
- Avec la commande ls -l les tubes nommés sont signalés par p
  prw-r–r– 1 bajard bajard 0 Apr 17 18:33 toto

Les tubes nommées

Création : mkfifo()

- Dans un shell : avec la commande unix mkfifo

```
MacBajard$ mkfifo toto
MacBajard$ ls -l toto
prw-r--r-- 1 bajard bajard 0 Apr 17 18:33 toto
```

- Dans un programme :

```
#include <sys/types.h>
#include <sys/stat.h>
int mkfifo(const char *path, mode_t mode);
```

path le chemin, mode les droits par exemple en octal 0644 pour rw - r- - r- -

Les tubes nommées

Exemple dans un shell

% mkfifo toto		% toto	
% cat > toto		% cat toto	
hello	écriture	hello	affichage
bye	écriture	bye	affichage

Les tubes nommées

Exemple de programmes

```
int g;
mkfifo("riton",0644);
g = open("riton",O_WRONLY);
write(g,"hello riton \n",20);
close(g);
```

```
int g,i;
char mot[20];
g = open("riton",O_RDONLY);
i = read(g,mot,20);
write(1,mot,i)
close(g);
```

- Les deux open forment un rendez-vous.
- Attention aux risques d'interblocage