



TP Système d'Exploitation (Linux)

Brahim ELBHIRI
GSCM_LRIT, FSR

Bienvenu à Tous!!!



Faculté des Sciences de Rabat

Plan

- Objectifs
- Notions
- Installation
- Exemple : Ubuntu
- Commandes de base

Objectifs

Se familiariser avec l'environnement Linux, les logiciels libres disponibles, commandes, gestion des processus, etc....

Notions

- Dans l'idée de créer un système multi-tâches, multi-utilisateurs:
 - Unix, Ken Thompson et Dennis Ritchie du laboratoire Bell en 1969
 - Linux, Linus Torvalds en 1991
- Pour le partage de connaissances et des fichiers sources:
 - GPL ("Gnu Public License") et de la FSF ("Free Software Foundation") toutes deux fondées par Richard Stallman

Notions de noyau et de distribution

- Linux est architecturé autour d'un **noyau** (en anglais **kernel**) qui prend en charge le matériel existant sur la machine;
- On appelle **distribution** l'assemblage d'un ensemble de logiciels autour du noyau Linux afin de fournir un système fonctionnel;
- Chaque distribution a une interface graphique propre ainsi qu'un système de paquetages;
- Les distributions les plus connues sont :
 - **RedHat** ;
 - **Debian** ;
 - **SuSe** ;
 - **Mandriva**;
 - **Ubuntu**.

Étapes d'installation

1. Amorçage du système

Capacité d'utiliser plusieurs systèmes à la fois

- *boot : noyau linux*
- *root : programme d'installation*

2. Création de partitions

- *partition principale (racine) créée en utilisant le système de fichiers Linux Ext2*
- *partition secondaire (swap) utilisée comme mémoire virtuelle en cas d'insuffisance de la mémoire vive*

Étapes (suite)

1. Crédit du système de fichiers

Opération automatique pour la majorité des distributions

- activation du swap : `mkswap -c partition taille`
- création du SF : `mke2fs -c partition taille`

2. Installation propre

- type de clavier, choix de langues, type d'installation
- points de montage : après la sélection des partitions, il s'agit de les nommer (racine → /; documents → /home; etc)
- formatage des partitions
- installation de des paquetages (packages)
- fin d'installation ☺

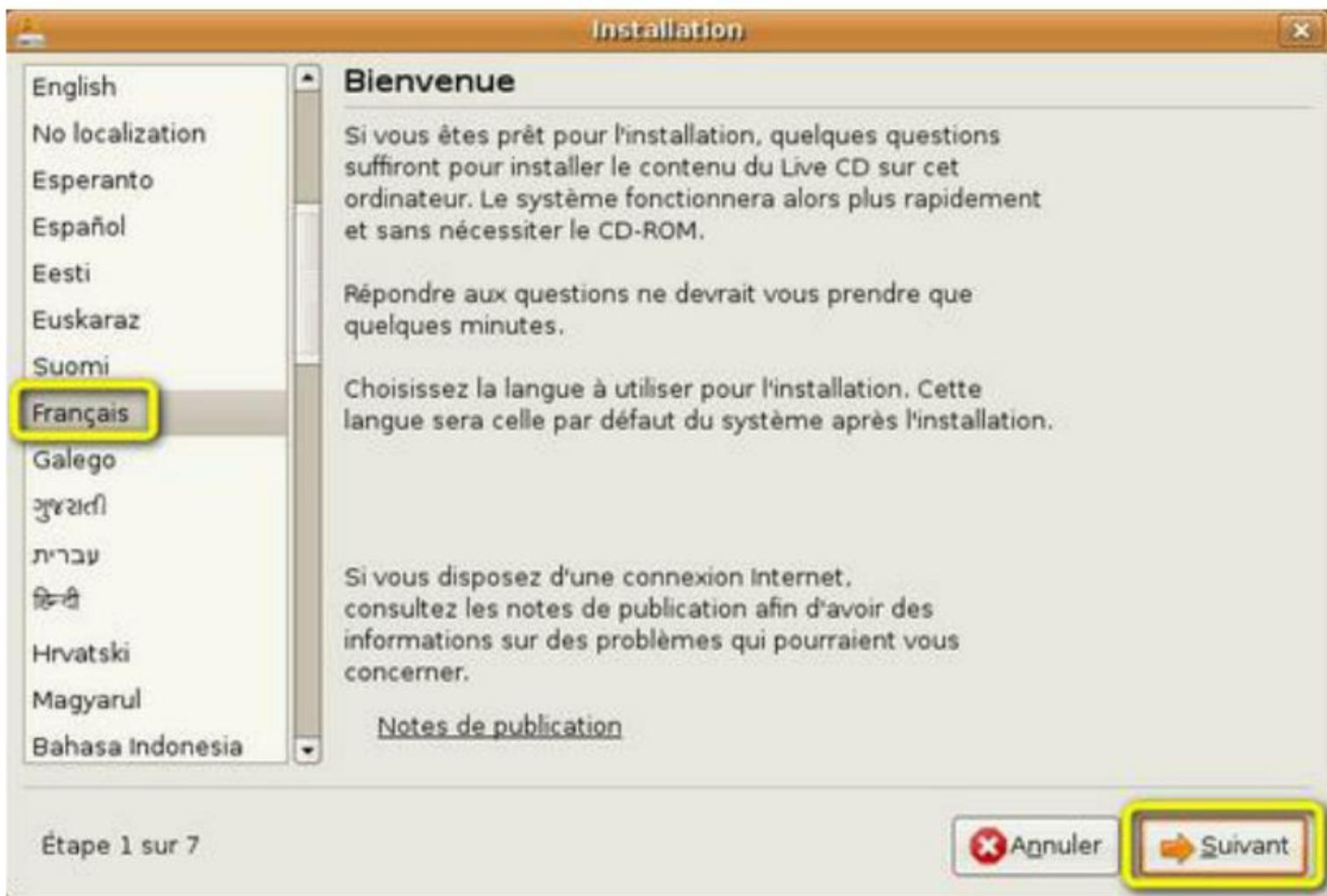
Ubuntu

I. Lancement du Desktop CD

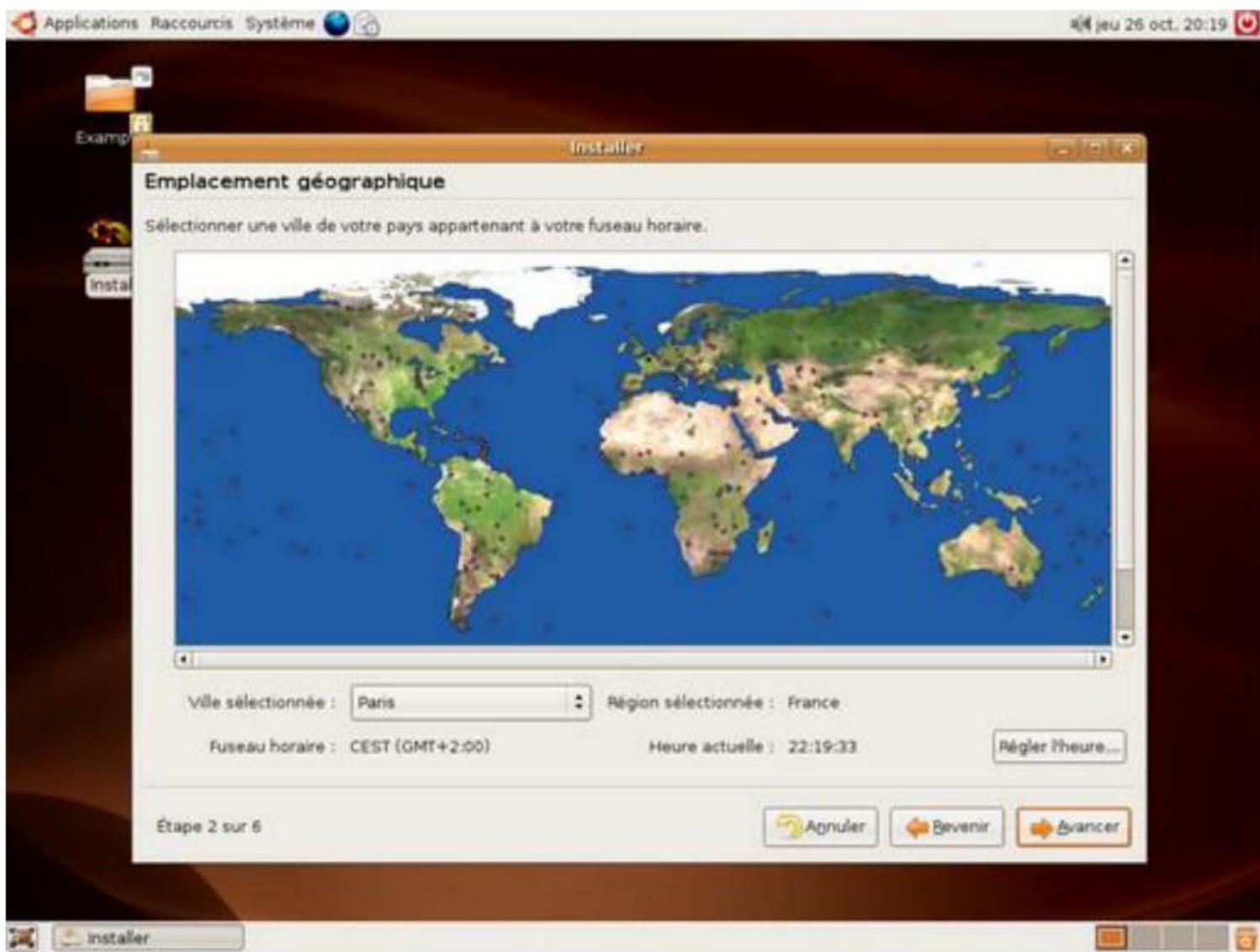


- I. Commencez dès-lors par explorer le menu présent :
 - « Language » avec la touche « F2 » ;
 - « Clavier » avec la touche « F3 » ;
 - Résolution/nombre de couleurs (« VGA ») avec la touche « F4 »;
 - Autres options avec la touche « F6 »;
- Choisir « Démarrer ou installer Ubuntu » (1^{ère} option).

Choix de langue



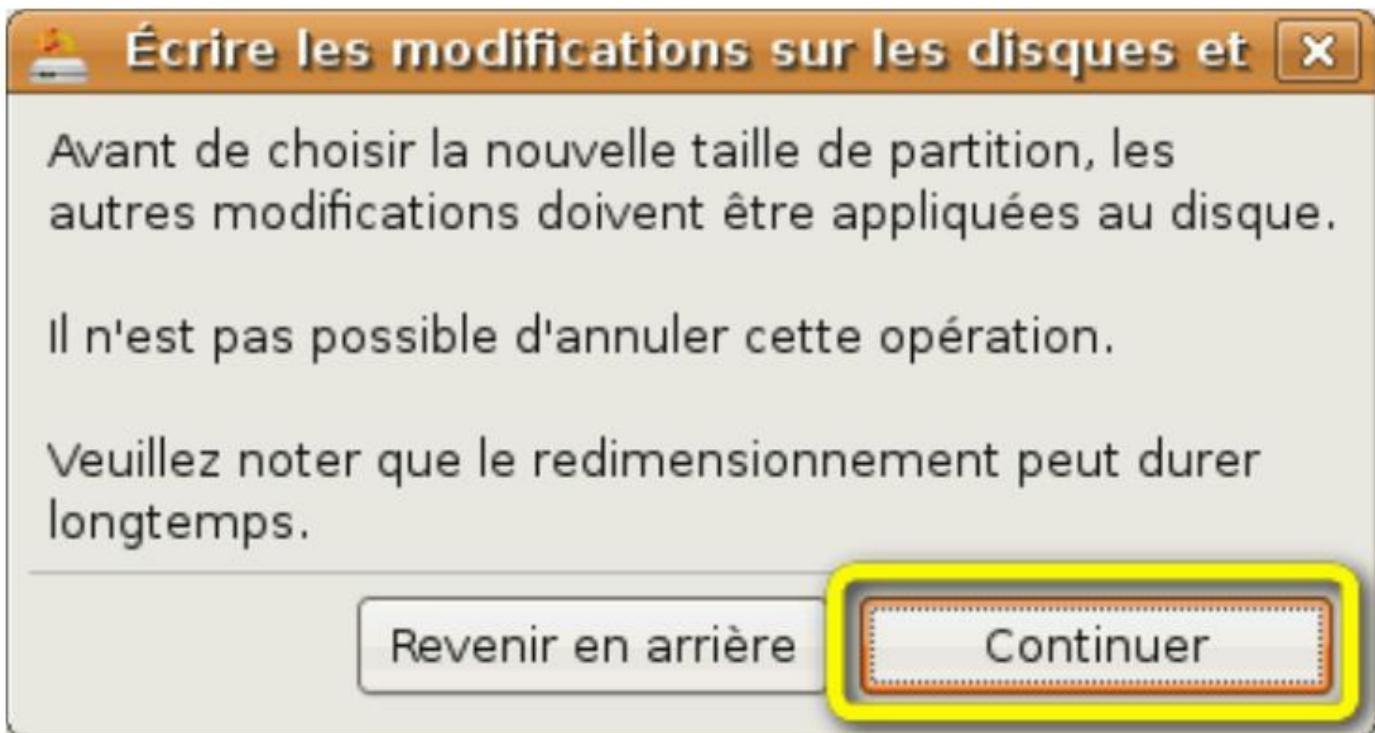
Votre géographie



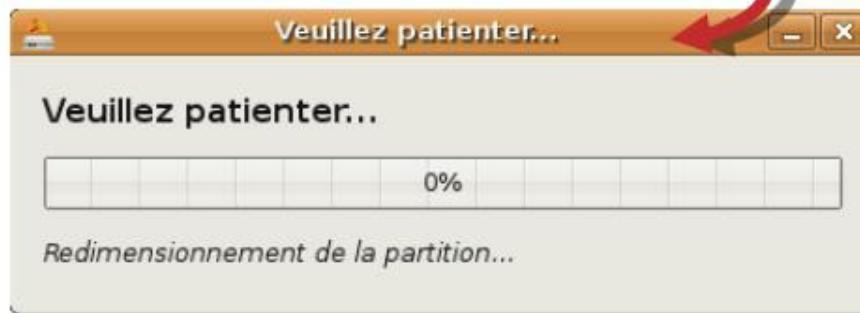
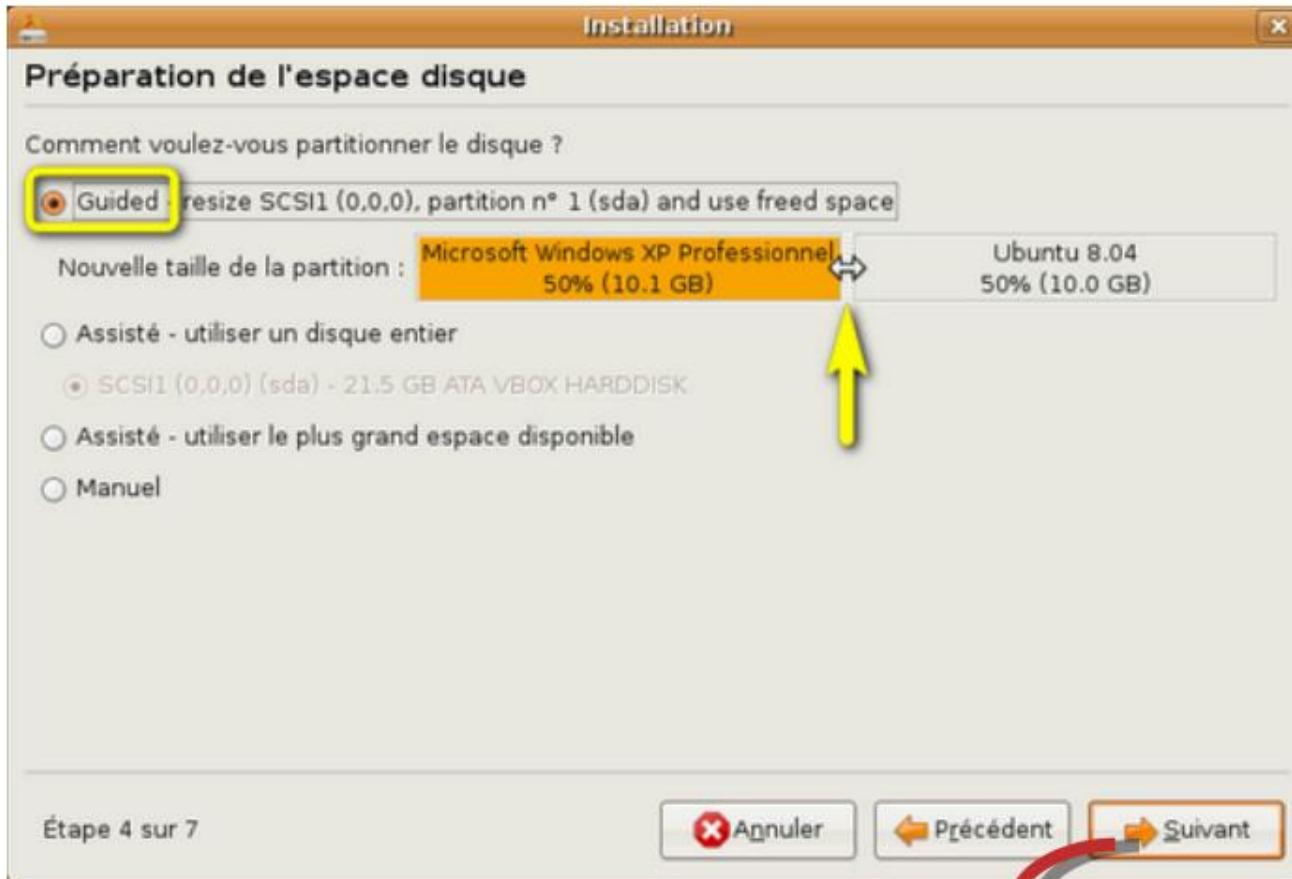
Clavier



Partitions



Partitions



Partitionnement manuel

Applications Raccourcis Système Live session user mar 22 avr, 23:38

Installation

Préparer les partitions

Device	Type	Mount point	Format?	Size	Used
/dev/sda					
/dev/sda1	ntfs		<input type="checkbox"/>	21130 MB	3200 MB
/dev/sda2	ntfs		<input type="checkbox"/>	63120 MB	69 MB
/dev/sda3	ext3		<input type="checkbox"/>	8743 MB	222 MB
/dev/sda5	ext3		<input type="checkbox"/>	34710 MB	458 MB
/dev/sda6	swap		<input type="checkbox"/>	1143 MB	0 MB

New partition table New partition **Modifier la partition** Delete partition

Annuler les modifications des partitions

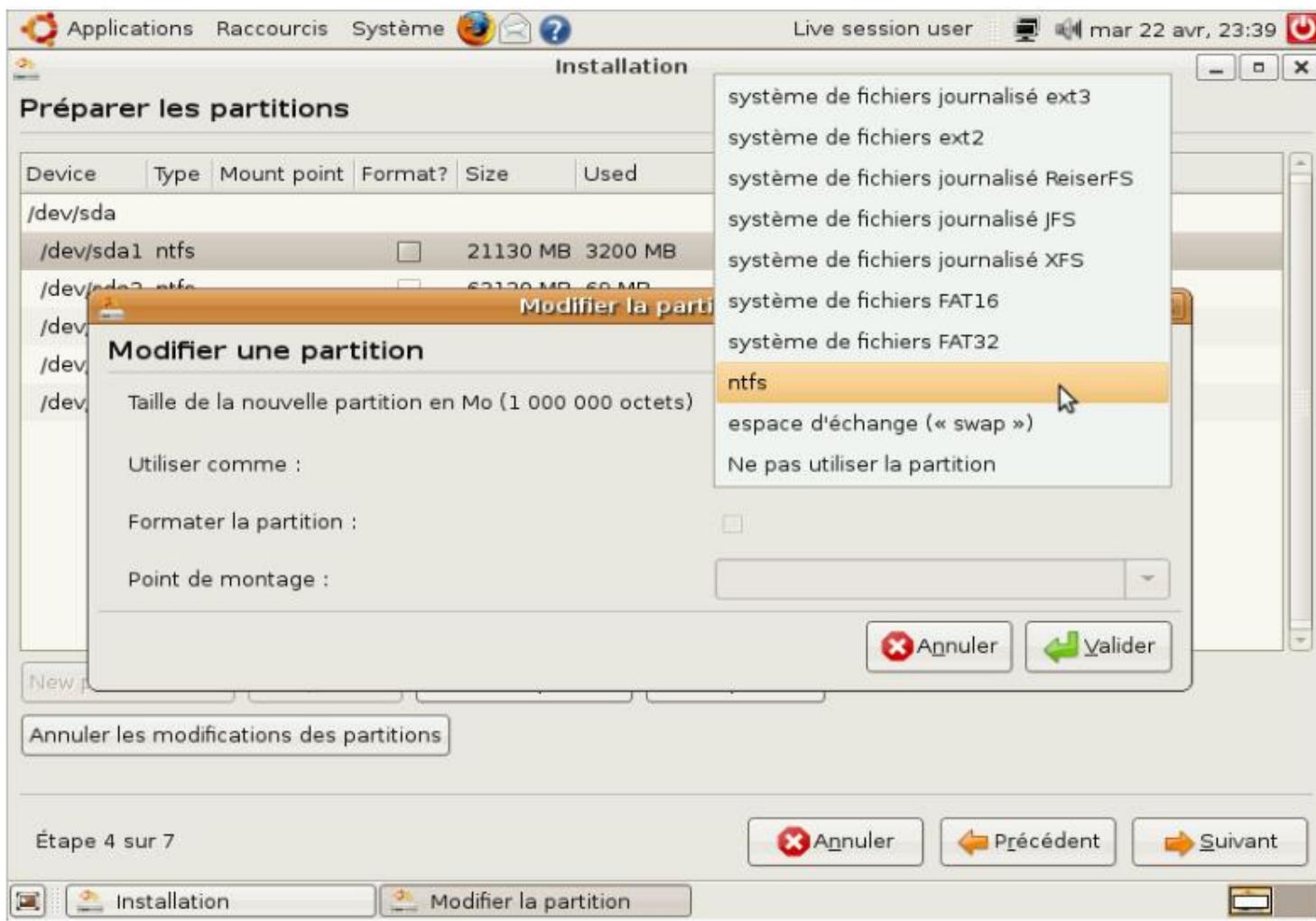
Étape 4 sur 7

Annuler Précédent Suivant

Installation

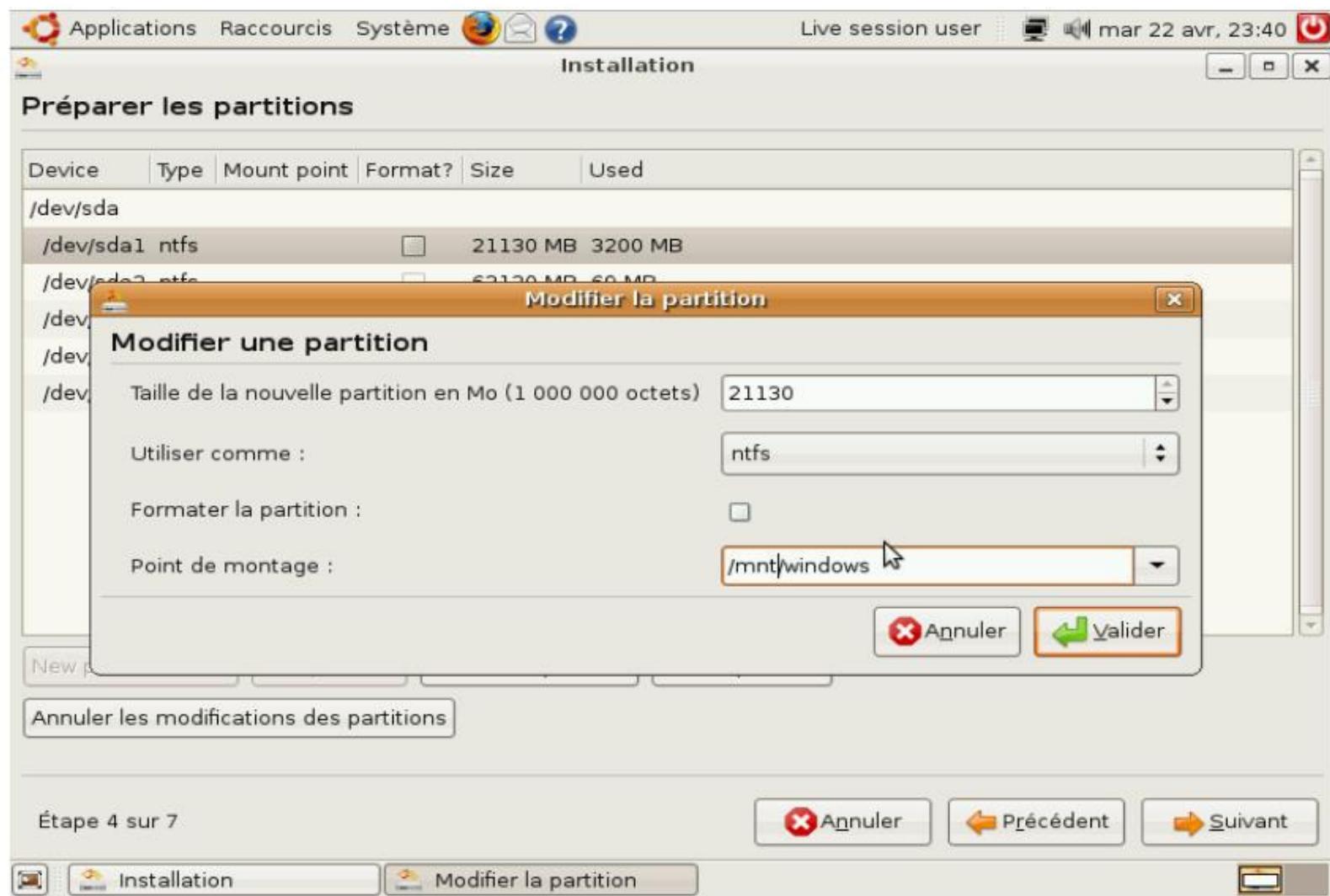
Partitionnement manuel

2



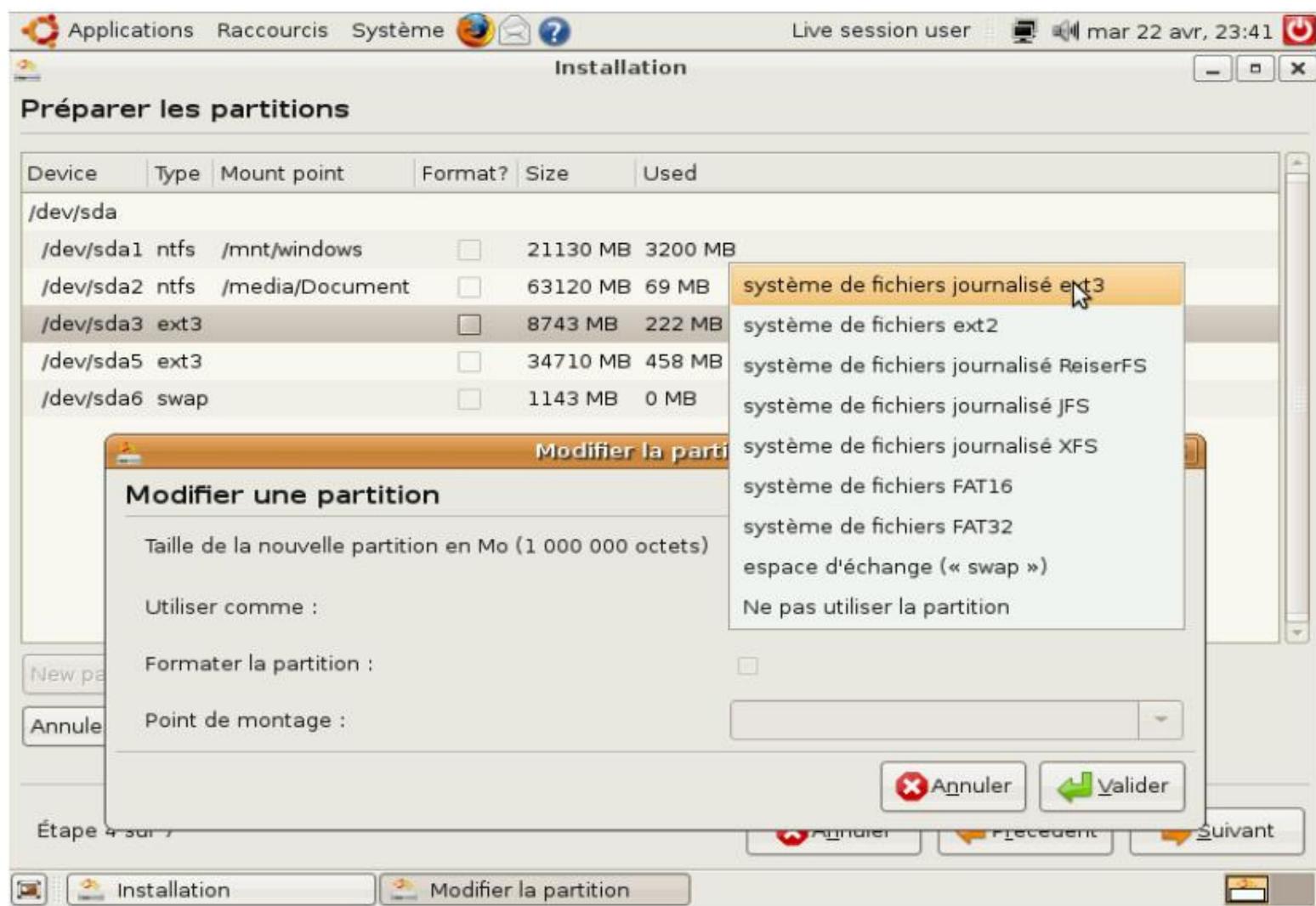
Partitionnement manuel

3

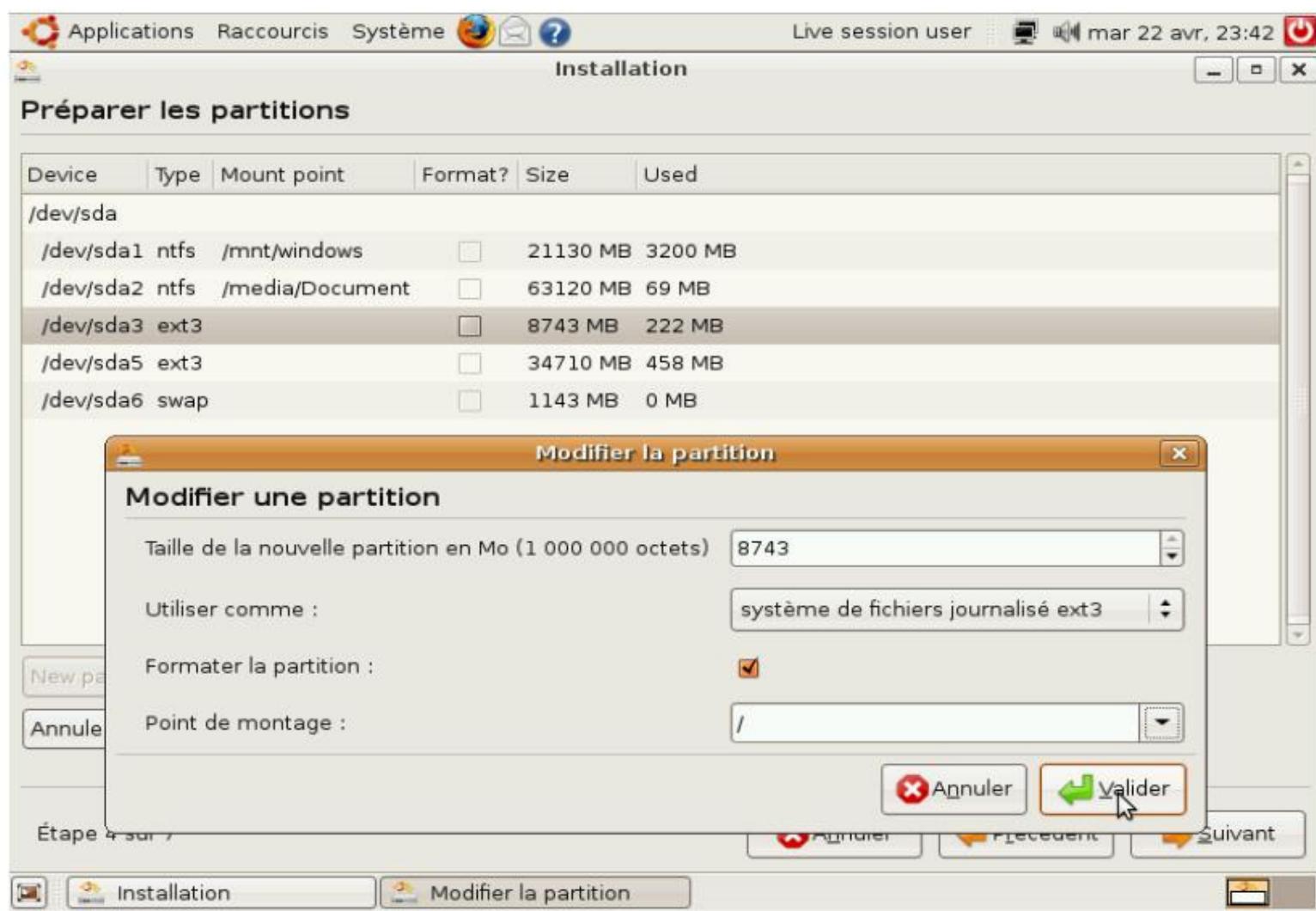


Partitionnement manuel

4



Partition /racine



Partition /home



Partition /swap

Le SWAP, qu'est ce que c est et à quoi sert?

A quoi sert le SWAP aujourd'hui dans nos distributions unix, sur des machines possédant souvent 1024Mo de RAM?

Partition /swap

- Fonctionner le système d'une manière optimal
- Incrire toutes les données temporaires nécessaires aux fonctionnement des programmes en cours d'exécution
- Exécution de programmes particulièrement gourmands en ressources mémoires (serveur d'échange de fichiers, serveur de calculs ,vidéo, 3D...)
- Si un programme n'est pas utilisé pendant un certain temps, Linux peut décider de le placer en espace "swap".
- Libérer de l'espace en RAM, augmenter la taille du cache disque, et donc d'augmenter les performances des accès aux disques et ainsi accroître les performances globales du système
- La mise en veille prolongée

Partition /swap

- La taille de SWAP fasse entre x1.5 à x2 de la RAM
- 1024(1Go) Mo de la RAM, recommande 2048 Mo(2Go)
- On peut avoir plus, mais ... (blocage du système au cas de dépassement de mémoire)
- Sous Unix il est facile d'ajouter de redimensionner le SWAP (Gparted pour ubuntu ou une console)
- Swapfile(fichier d'échange), SWAP effectif

sudo su : se connecter en administrateur

more /proc/swaps : connaître l'espace du SWAP

more /proc/meminfo : Connaitre les informations mémoire du système

df -ah : Connaitre l'espace disque disponible des différentes partitions

sudo dd if=/dev/zero of=/file.swap bs=1024 count=512000 : la commande dd crée et copie des zéros dans le fichier file.swap contenant 512000 blocs de 1024 octets chacun, soit un fichier approximatif de 512 Mo (voir 524 Mo).

sudo mkswap /file.swap 512000: formater le fichier en tant qu'espace d'échange ou de "swap"

sudo swapon /file.swap: Activer le SWAP

sudo swapoff /file.swap: désactiver le SWAP

Partition /swap

Applications Raccourcis Système Live session user mar 22 avr, 23:44

Installation

Préparer les partitions

Device	Type	Mount point	Format?	Size	Used
/dev/sda					
/dev/sda1	ntfs	/mnt/windows	<input type="checkbox"/>	21130 MB	3200 MB
/dev/sda2	ntfs	/media/Document	<input type="checkbox"/>	63120 MB	69 MB
/dev/sda3	ext3	/	<input checked="" type="checkbox"/>	8743 MB	222 MB
/dev/sda5	ext3	/home	<input type="checkbox"/>	34710 MB	458 MB
/dev/sda6	swap		<input type="checkbox"/>	1143 MB	0 MB

Modifier la partition

Modifier une partition

Taille de la nouvelle partition en Mo (1 000 000 octets) : 1143

Utiliser comme : espace d'échange (« swap »)

Formater la partition :

Point de montage :

Étape

Annuler **Valider** **Suivant**

Installation Modifier la partition

Récapitulatif

Applications Raccourcis Système Live session user mar 22 avr, 23:44

Installation

Préparer les partitions

Device	Type	Mount point	Format?	Size	Used
/dev/sda					
/dev/sda1	ntfs	/mnt/windows	<input type="checkbox"/>	21130 MB	3200 MB
/dev/sda2	ntfs	/media/Document	<input type="checkbox"/>	63120 MB	69 MB
/dev/sda3	ext3	/	<input checked="" type="checkbox"/>	8743 MB	222 MB
/dev/sda5	ext3	/home	<input type="checkbox"/>	34710 MB	458 MB
/dev/sda6	swap		<input type="checkbox"/>	1143 MB	0 MB

New partition table New partition Modifier la partition Delete partition

Annuler les modifications des partitions

Étape 4 sur 7

Annuler Précédent Suivant

Installation

Identifiez-vous

 Installation X

Identité

Quel est votre nom ?

Pierre Dupont

Quel nom d'utilisateur voulez-vous utiliser pour vous identifier ?

pierre

Si plus d'une personne sont amenées à utiliser cet ordinateur, vous pourrez créer d'autres comptes après l'installation.

Choisissez un mot de passe pour protéger votre compte utilisateur.

Entrez le même mot de passe à deux reprises, afin d'éviter toute erreur de saisie.

Quel est le nom de cet ordinateur ?

totor

Ce nom sera utilisé pour identifier l'ordinateur sur un réseau.

Étape 5 sur 7

Annuler Précédent Suivant

Vos documents Windows

Installation

Importer les documents et paramètres

Veuillez sélectionner le ou les comptes à importer. Les documents et les paramètres de ces comptes seront disponibles dès la fin de l'installation.

Si vous ne souhaitez pas importer de comptes utilisateurs, ne sélectionnez rien et allez sur la page suivante.

- ▼ pierre Microsoft Windows XP Professionnel (sda1)
 - Internet Explorer
 - Wallpaper
 - User Picture
 - My Documents
 - My Music
 - My Pictures

Etape 6 sur 7

 Annuler

 Précédent

 Suivant

Installation

Prêt à installer

Votre nouveau système d'exploitation va maintenant être installé avec les paramètres suivants :

Langue : French
Disposition du clavier : France - Alternative
Nom : Pierre Dupont
Nom d'utilisateur : pierre
Emplacement : Europe/Paris

Si vous continuez, les modifications affichées seront écrites sur les disques. Dans le cas contraire, vous pourrez faire d'autres modifications.

ATTENTION : cela détruira toutes les données présentes sur les partitions que vous avez supprimées et sur celles qui seront formatées.

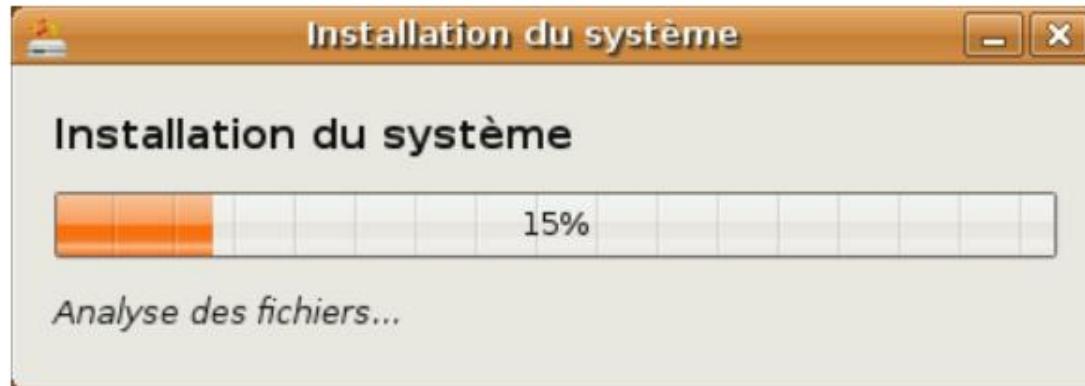
Les tables de partitions des périphériques suivants seront modifiées : SCSI1 (0.0.0) (sda)

Avancé...

Étape 7 sur 7

Annuler Précédent Installer

Fin installation



Après redémarrage

```
Ubuntu 8.04, kernel 2.6.24-16-generic
Ubuntu 8.04, kernel 2.6.24-16-generic (recovery mode)
Ubuntu 8.04, memtest86+
Other operating systems:
Microsoft Windows XP Professionnel
```

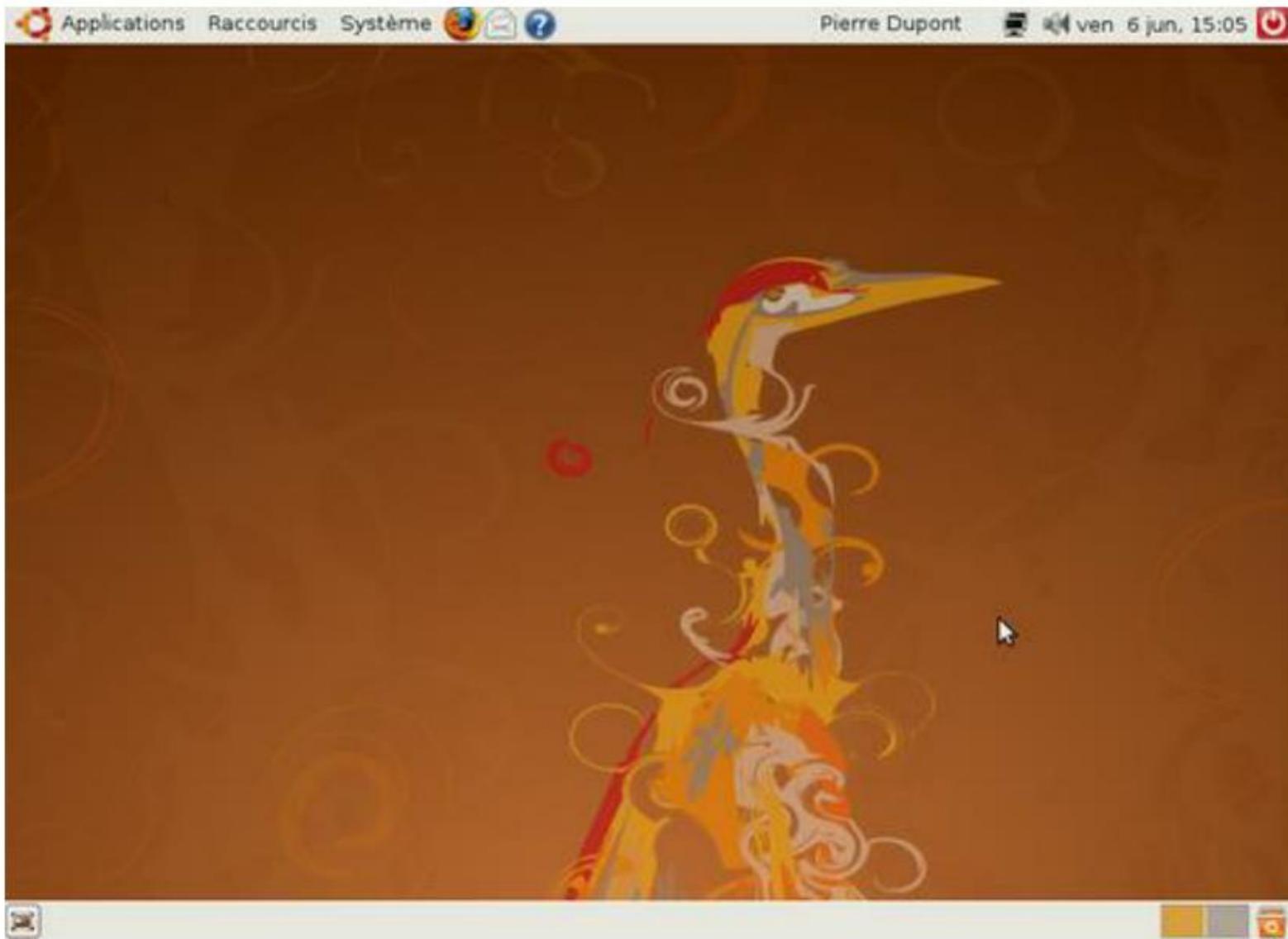
Use the **↑** and **↓** keys to select which entry is highlighted.
Press **enter** to boot the selected OS, **'e'** to edit the
commands before booting, or **'c'** for a command-line.

The highlighted entry will be booted automatically in 9 seconds.

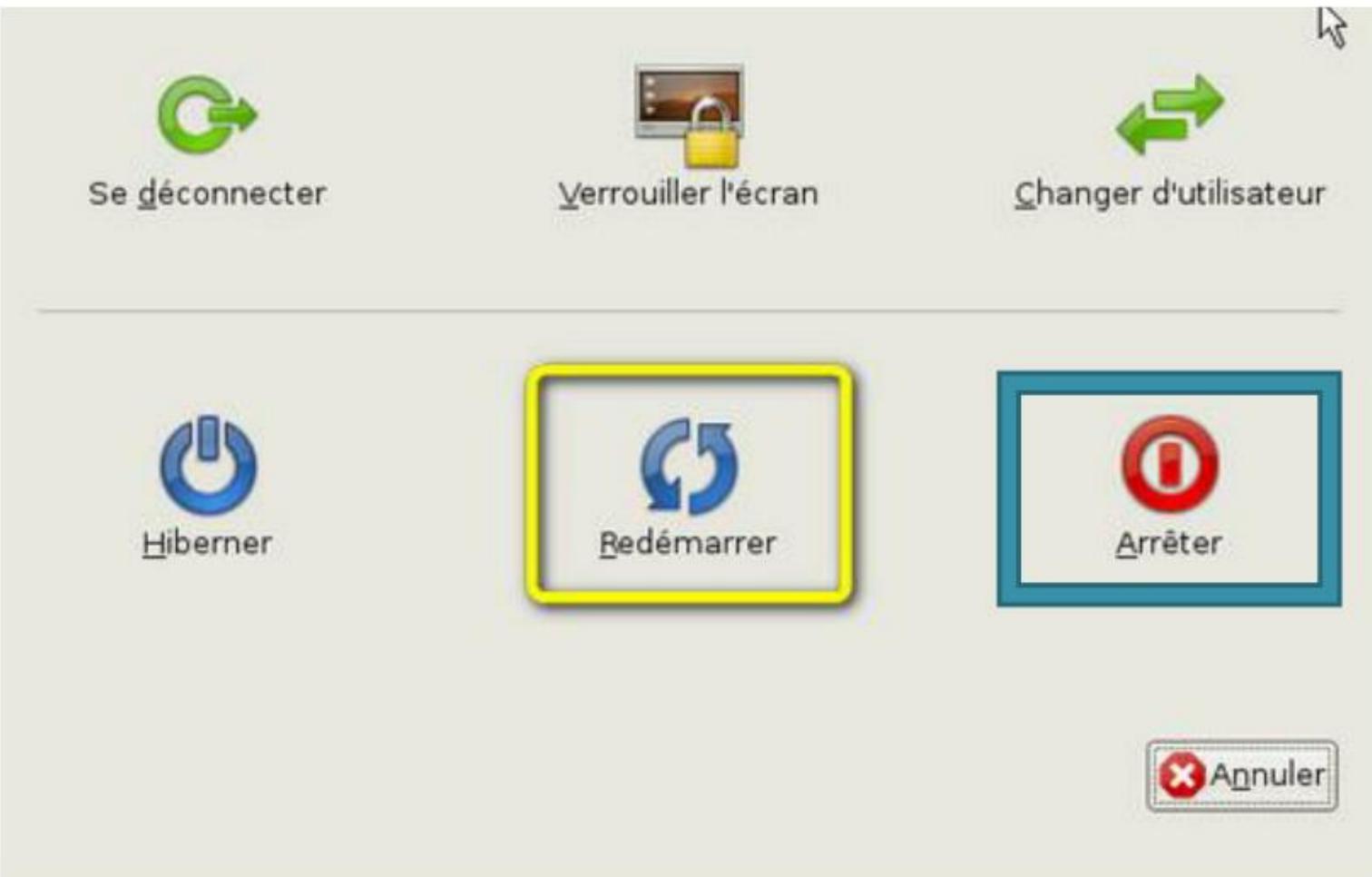
Accès



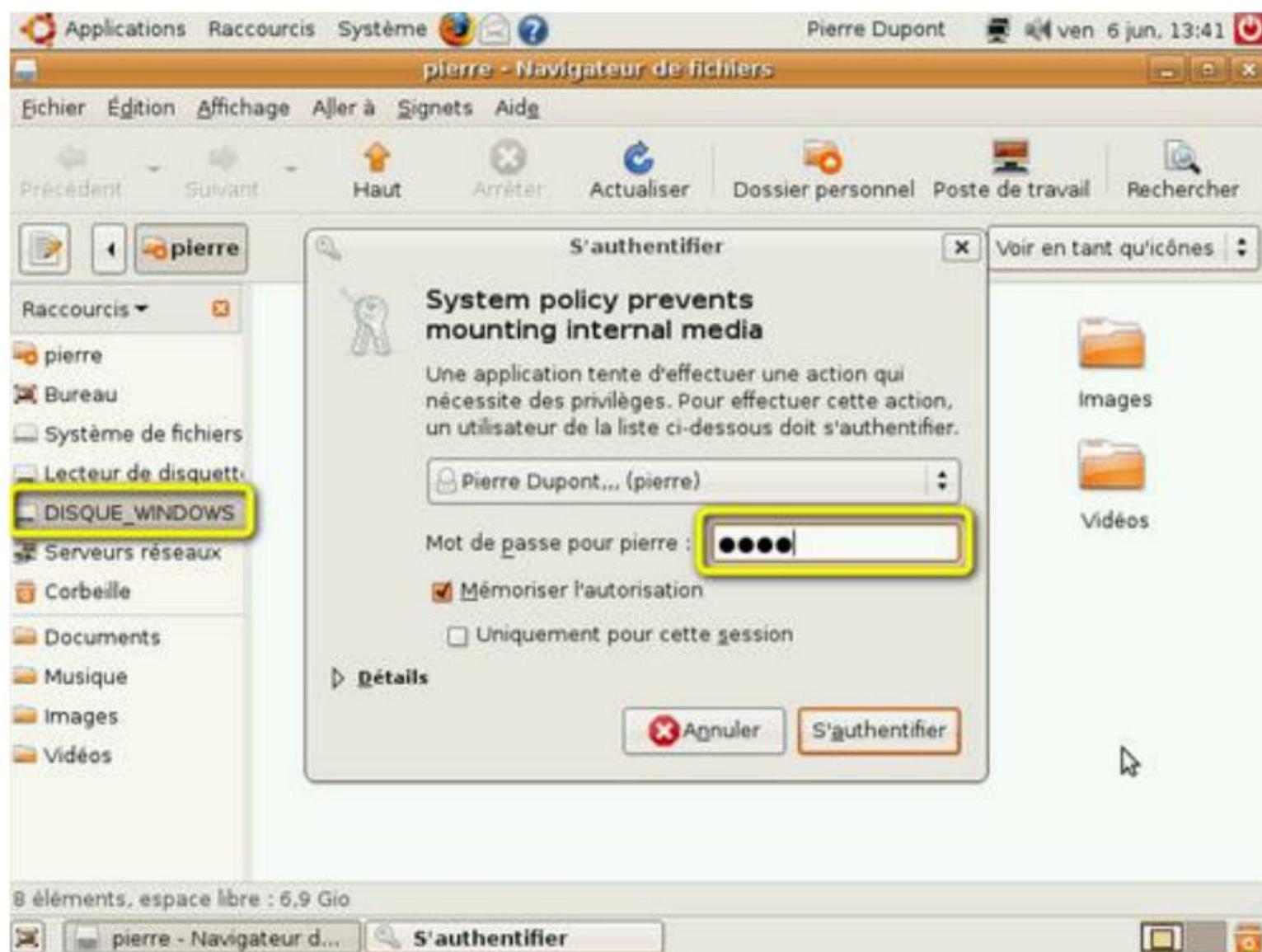
Ubuntu prêt



Arrêt / redémarrage



Accès partition Windows



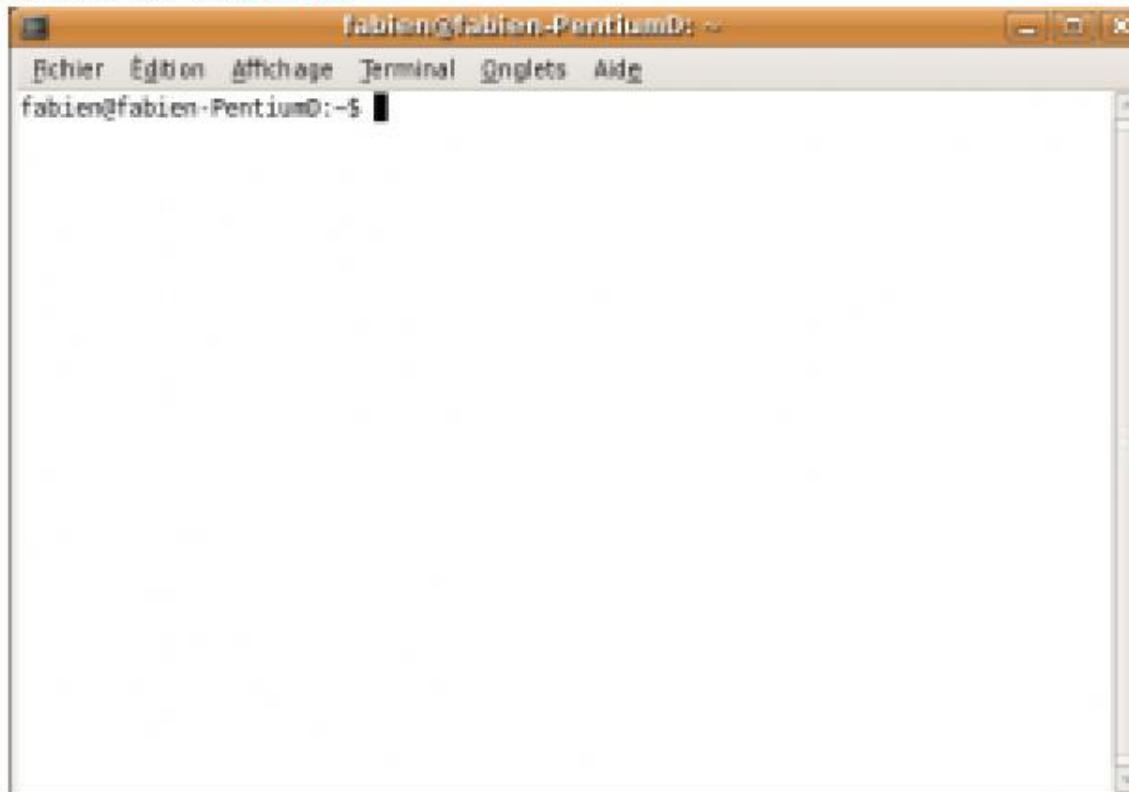




Revenons à Ubuntu ☺

Terminal/shell

- Les systèmes Unix/Linux furent à l'origine conçus pour fonctionner en mode texte, sans interface graphique : ligne de commande (CLI - Command Line Interface), .
- Cette ligne de commande est accessible via les *terminaux* qui jouent le rôle d'interface utilisateur/machine et fonctionnent avec un interpréteur de commandes : le *shell*.



Terminal

- Pour ouvrir un terminal :
 - *Applications → Accessoires → Terminal*
 - ou Alt+F2 et tapez **gnome-terminal**

- Pour utiliser un terminal:
 - *tapez une commande ou copiez-collez la,*
 - *ensuite faites Entrer (clavier).*

Shell

- Le rôle du Shell est d'assurer la correspondance entre ce que l'utilisateur tape et le système. Il en existe de nombreuses versions: **sh** (appelé «Bourne shell»), **bash** («Bourne again shell»), **csh** («C Shell»), **Tcsh** («Tenex C shell»), **ksh** («Korn shell») et **zsh** («Zero shell»)
- Son rôle consiste ainsi à lire la ligne de commande, interpréter sa signification, exécuter la commande, puis retourner le résultat sur les sorties.
- Le shell par défaut est précisé dans le fichier de configuration **/etc/passwd** dans le dernier champ de la ligne correspondant à l'utilisateur.
- Il est possible de changer de shell dans une session en exécutant tout simplement le fichier exécutable correspondant, par exemple : **/bin/bash**

Prompt

- **Invite de commande**

- Après initialisation du shell (lecture de sa configuration)
- le prompt est donné comme suit:
 - « nom de la machine » : « répertoire courant » « symbole »
 - symbole « \$ » : utilisateur normal
 - symbole « #» : administrateur, appelé «root»





Commandes de base

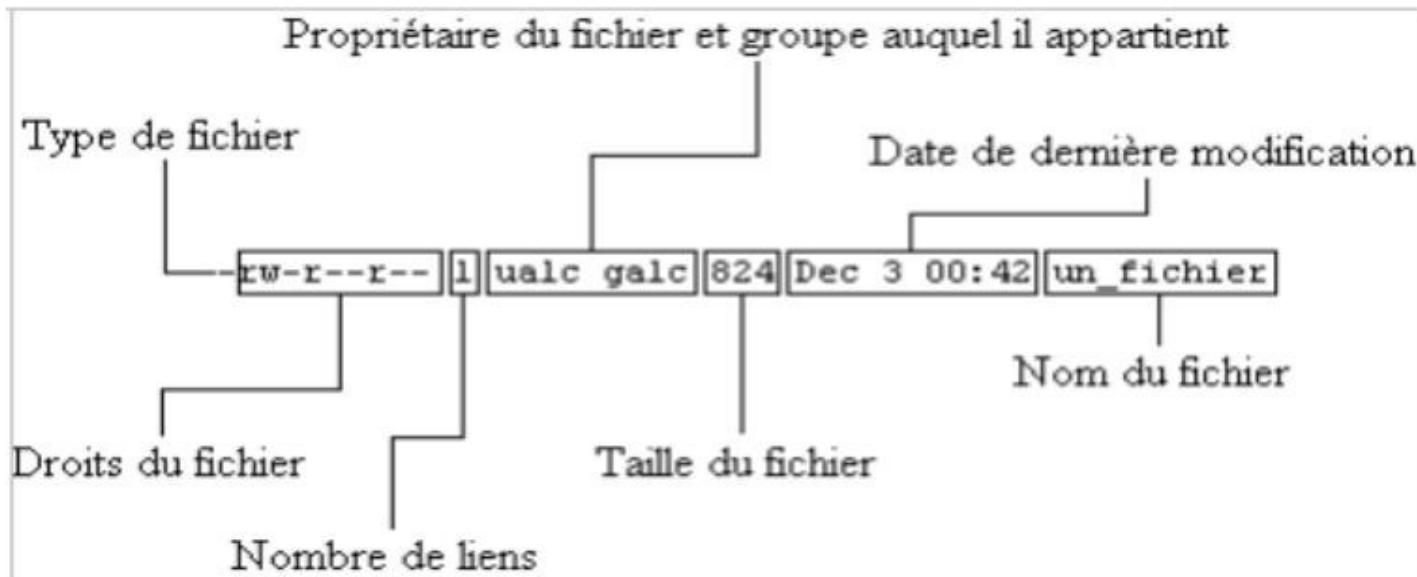
Répertoires/fichiers

But de la commande	Syntaxe	Équivalence MsDos
Se déplace vers le répertoire rep	cd rep	cd rep
Liste le contenu d'un répertoire (-R pour un listage récursif)	ls rep	dir
Copie un fichier (-R pour un répertoire)	cp source destination	copy
Déplace un fichier	mv source destination	move
Crée un répertoire	mkdir rep	mkdir rep
Efface un fichier (-rf pour effacer un répertoire)	rm fichier	del / deltree fichier
Crée un lien destination qui va pointer sur une source (-s pour un lien symbolique)	ln source destination	non disponible
Met à jour la date de modification du fichier, ou crée un fichier vide si le fichier n'existe pas.	touch fichier ou répertoire	non disponible

Lecture des fichiers

Syntaxe	But commandes
cat fichier	Affiche le contenu du fichier sur la sortie standard
more fichier	Lit le contenu d'un fichier page par page. (Il doit lire l'intégralité du fichier avant de l'afficher)
less fichier	Equivalent à more sauf qu'il n'est pas obligé de lire l'intégralité du fichier et permet de remonter dans l'affichage
tail fichier	N'affiche que les dernières lignes d'un fichier (-n permet de spécifier le nombre de lignes à afficher)
head fichier	Comme tail, mais affiche les N premières lignes d'un fichier (N=10 par défaut)
grep "chaine" fichier	Recherche l'occurrence d'une chaîne de caractères "chaine" dans un ou plusieurs fichiers

Attributs de fichiers



```
user@localhost$ ls -l
```

L'éditeur Vim

- Appel de l'éditeur
 - `vi fichier`
- Modes d'édition
 - **le mode insertion :**

Les caractères tapés s'insèrent directement dans le texte en cours (I)

- **le mode commande**

Les caractères tapés sont considérés comme des commandes d'édition de texte (:)

Commandes vi

Commande	Description
:w	Sauvegarde les changements
:q	Quitte le document en cours
:q!	Quitte le document en cours sans sauvegarder les changements
:wq	Enregistre et quitte l'éditeur
:f file2	Enregistre sous un autre nom le document (équivalent de enregistrer sous)
/hop	Recherche l'occurrence hop et la met en surbrillance, "n" se positionne sur la prochaine occurrence
v	Permet de passer en mode visualisation pour faire une sélection par exemple
d	Équivalent de couper, après une sélection par exemple
dd	Coupe la ligne entière
y	Équivalent de copier, après une sélection par exemple
yy	Copie la ligne entière
p	Coller après le curseur
esc	Changement de mode

Mode Edition

Après l'utilisation de ces commandes, l'éditeur passe en mode édition :

Commande	Description
a	Ajoute après le curseur
A	Ajoute après la ligne courante
i	Insère avant le curseur
I	Insère au début de la ligne courante avant le premier caractère non-blanc
o	Ouvre une nouvelle ligne en-dessous de la ligne actuelle
O	Ouvre une nouvelle ligne au-dessus de la ligne actuelle

Gestion des processus

- Un processus est une instance d'un programme en train de s'exécuter, une tâche.
- Il possède un numéro unique sur le système pid
- Chaque processus appartient à un utilisateur et un groupe et à les droits qui leur sont associés
- Sous shell, un processus est créé pour exécuter chacune des commandes
- Le shell est le processus père de toutes les commandes.

Statut d'un processus

- Runing: le processus s'exécute
- Waiting: Attend quelque chose pour s'exécuter
- Ready: le processus a tout pour s'exécuter sauf le processeur
- Suspendu: Arrêté
- Zombie: état particulier

Le multitâche sous unix

- Unix est un système *multitâches*: il peut exécuter plusieurs programmes à la fois
- Le shell crée un nouveau processus pour exécuter chaque commande

Le multitâche sous unix

- Exemple:
- Xclock: lancement d'un processus en 1^{ier} plan (foreground)
- ctl+c: l'arrêt définitif d un processus
- Xclock&: lancement d'un processus en 2^{ième} plan (background)
- ^z:un signal qui suspend l'exécution sans détruire le processus correspondant
- Bg: pour afficher les processus en BG
- Fg: pour afficher les processus en FG
- Il reprend l'exécution là où il avait laissée

Le multitâche sous unix

- Le programme tourne au premier plan :
 - ^Z le suspend ;
 - ^C l'interrompt.
- Le programme est suspendu :
 - fg le passe au premier plan ;
 - bg le passe en arrière-plan.
- Le programme tourne en arrière-plan :
 - si c'est le seul dans ce cas, fg le passe au premier plan ;
 - sinon, c'est plus compliqué.
- Le programme ne tourne pas :
 - il n'y a rien à faire...

Voir les processus

La commande ps:

- PID (*process identifier*) : c'est le numéro du processus.
- TT : indique le terminal dans lequel a été lancé le processus. Un point d'interrogation signifie que le processus n'est attaché à aucun terminal (par exemple les démons).
- STAT : indique l'état du processus :
 - R : actif (*running*)
 - S : non activé depuis moins de 20 secondes (*sleeping*)
 - I : non activé depuis plus de 20 secondes (*idle*)
 - T : arrêté (suspendu)
 - Z : zombie
- TIME : indique le temps machine utilisé par le programme (et non pas le temps depuis lequel le processus a été lancé !).

Commande ps: options

- a (*all*) : donne la liste de tous les processus, y compris ceux dont vous n'êtes pas propriétaire.
- g (*global, général...*) : donne la liste de tous les processus dont vous êtes propriétaire.
- u (*user, utilisateur*) : donne davantage d'informations (nom du propriétaire, heure de lancement, pourcentage de mémoire occupée par le processus, etc.).
- x : affiche aussi les processus qui ne sont pas associés à un terminal.
- w : le tronque pas à 80 caractères (peut être utilisée plusieurs fois pour tronquer plus loin)
- agux : est en fait souvent utilisé pour avoir des informations sur tout les processus.

Commande top

La commande top affiche les mêmes informations, mais de façon dynamique : elle indique en fait par ordre décroissant le temps machine des processus, les plus gourmands en premier.

Tuer les processus

- ^c
- ^d
- ^z
- Kill pid: tuer un processus
- Kill -9 pid: imposer l'arrêt immédiat du processus

Autres commandes pour la gestion des processus

- **ps** : liste des processus et de leurs caractéristiques
- **htop** : liste dynamique des processus et de ce qu'ils consomment
- **pgrep** : récupération d'une liste de processus par expression régulière
- **pidof** : récupération du pid d'un processus recherché
- **fuser** : informations sur les file descriptor d'un processus
- **lsof** : idem
- **pmap** : afficher le mapping mémoire d'un processus
- **strace** : liste les appels système du processus
- **ltrace** : liste les appels de fonction de bibliothèques dynamiques du processus
- **pstack** : affiche la pile d'appel du processus
- **gdb** : pour tout savoir et même modifier l'action d'un processus.
- **kill** : envoyer un signal à un processus connaissant son pid
- **killall** : envoie un signal à tous les processus portant un certain nom
- **pkill** : envoie un signal aux processus matchant une expression régulière
- **ctrl-z** : envoie le signal STOP au processus en avant plan du shell en cours
- **fg, bg** : envoie le signal CONT à un processus stoppé du shell en cours



Gestion des Processus

Création

Création

```
#include<stdio.h>
#include<>sys/times.h>
Int main()
{int pid;
Char quisuisje="le pere";
Pid=fork();
If (pid==0)
{
    Quisuisje="le fils";
    Printf(" je suis le %s ",quisuisje );
}
else
{
    Printf(" je suis le %s ",quisuisje );
    Wait(NULL);
}
Return 0;
}
```

Création

Le Papa

Descripteurs de fichiers

1: (Stdout) Position = 0

Variables globales

```
const char* quisuisje = NULL;
```

```
int main()
{
    pid_t pid;
    quisuisje = "Le pere";
    pid = fork();
    if(pid == 0){
        quisuisje = "Le fils";
        printf("Je suis %s", quisuisje);
    }
    else{
        printf("Je suis %s", quisuisje);
        wait(NULL);
    }
    return 0;
}
```

Création

Le Papa

Descripteurs de fichiers

1: (Stdout) Position = 0

Variables globales

```
const char* quisuisje = "Le pere";
```

```
int main()
{
    pid_t pid;
    quisuisje = "Le pere";
    → pid = fork();
    if(pid == 0){
        quisuisje = "Le fils";
        printf("Je suis %s", quisuisje);
    }
    else{
        printf("Je suis %s", quisuisje);
        wait(NULL);
    }
    return 0;
}
```

Création

Le Papa

Descripteurs de fichiers

1: (Stdout) Position = 0

Variables globales

```
const char* quisuisje = "Le pere";
```

```
int main()
{
    pid_t pid;
    quisuisje = "Le pere";
    pid = fork(); //pid = 1000
    → if(pid == 0){
        quisuisje = "Le fils";
        printf("Je suis %s", quisuisje);
    }
    else{
        printf("Je suis %s", quisuisje);
        wait(NULL);
    }
    return 0;
}
```

Le Fiston

Descripteurs de fichiers

1: (Stdout) Position = 0

Variables globales

```
const char* quisuisje = "Le pere";
```

```
int main()
{
    pid_t pid;
    quisuisje = "Le pere";
    pid = fork(); //pid=0
    → if(pid == 0){
        quisuisje = "Le fils";
        printf("Je suis %s", quisuisje);
    }
    else{
        printf("Je suis %s", quisuisje);
        wait(NULL);
    }
    return 0;
}
```

Création

Le Papa

Descripteurs de fichiers

1: (Stdout) Position = 0

Variables globales

```
const char* quisuisje = "Le pere";
```

```
int main()
{
    pid_t pid;
    quisuisje = "Le pere";
    pid = fork(); //pid = 1000
    if(pid == 0){
        quisuisje = "Le fils";
        printf("Je suis %s", quisuisje);
    }
    else{
        → printf("Je suis %s", quisuisje);
        wait(NULL);
    }
    return 0;
}
```

Le Fiston

Descripteurs de fichiers

1: (Stdout) Position = 0

Variables globales

```
const char* quisuisje ="Le pere";
```

```
int main()
{
    pid_t pid;
    quisuisje = "Le pere";
    pid = fork(); //pid=0
    if(pid == 0){
        → quisuisje = "Le fils";
        printf("Je suis %s", quisuisje);
    }
    else{
        printf("Je suis %s", quisuisje);
        wait(NULL);
    }
    return 0;
}
```

Création

Le Papa

Descripteurs de fichiers

1: (Stdout) Position = 15

Variables globales

const char* quisuisje = "Le pere";

```
int main()
{
    pid_t pid;
    quisuisje = "Le pere";
    pid = fork(); //pid = 1000
    if(pid == 0){
        quisuisje = "Le fils";
        printf("Je suis %s", quisuisje);
    }
    else{
        printf("Je suis %s", quisuisje);
        → wait(NULL);
    }
    return 0;
}
```

Le Fiston

Descripteurs de fichiers

1: (Stdout) Position = 15

Variables globales

const char* quisuisje = "Le fils";

```
int main()
{
    pid_t pid;
    quisuisje = "Le pere";
    pid = fork(); //pid=0
    if(pid == 0){
        quisuisje = "Le fils";
        → printf("Je suis %s", quisuisje);
    }
    else{
        printf("Je suis %s", quisuisje);
        wait(NULL);
    }
    return 0;
}
```

Création

Le Papa

Descripteurs de fichiers

1: (Stdout) Position = 30

Variables globales

```
const char* quisuisje = "Le pere";
```

```
int main()
{
    pid_t pid;
    quisuisje = "Le pere";
    pid = fork(); //pid = 1000
    if(pid == 0){
        quisuisje = "Le fils";
        printf("Je suis %s", quisuisje);
    }
    else{
        printf("Je suis %s", quisuisje);
        wait(NULL);
    }
    return 0;
}
```

Le Fiston

Descripteurs de fichiers

1: (Stdout) Position = 30

Variables globales

```
const char* quisuisje ="Le fils";
```

```
int main()
{
    pid_t pid;
    quisuisje = "Le pere";
    pid = fork(); //pid=0
    if(pid == 0){
        quisuisje = "Le fils";
        printf("Je suis %s", quisuisje);
    }
    else{
        printf("Je suis %s", quisuisje);
        wait(NULL);
    }
    return 0;
}
```

fork()

- Un fork() est une fonctionnalité sous les systèmes Unix qui permet de **dupliquer** (créer) un processus.
- La valeur de retour de la fonction est :
 - 0, dans le processus fils;
 - Pid, l'identité du processus fils créé, dans le processus père.
 - -1, si la primitive échoue
- Exit(n): terminer un processus avec un statut n.

Le papa

```
#include <sys/types.h>
#include <stdio.h>
main()
{
    int pid;
    switch(pid=fork()){
        case -1:perror("Création de processus");
            exit(2);
        case 0: /* on est dans le processus fils*/
            printf("valeur de fork = %d au moment %f", pid,time());
            printf("je suis le processus %d de père %d\n", getpid(), getppid());
            printf("fin de processus fils\n ");
            exit(0);
        default : /*on est dans le processus père*/
            printf("valeur de fork = %d", pid);
            printf("je suis le processus %d de père %d\n", getpid(), getppid());
            printf("fin de processus père\n");}
}
```

Le papa

```
#include <sys/types.h>
#include <stdio.h>
main()
{
    int pid;
    switch(pid=fork()){
        case -1:
            perror("Création de processus");
            exit(2);
        case 0 :
            printf("... ", pid, time());
            printf("... ", getpid(), getppid());
            printf("fin de processus fils\n ");
            exit(0);
        default :
            printf("... = %d", pid);
            printf("... ", getpid(), getppid());
            printf("fin ..père\n ");}}
```

Le Fils

```
#include <sys/types.h>
#include <stdio.h>
main()
{
    int pid;
    switch(pid=fork()){
        case -1:
            perror("Création de processus");
            exit(2);
        case 0 :
            printf("... ");
            printf("... ");
            printf("fin de processus fils\n ");
            exit(0);
        default :
            printf("... = %d", pid);
            printf("... ");
            printf("fin ..père\n ");}}
```

Tests:

- Voir les processus : ps -ef
 - Dans le père
 - Dans le fils

Résumé

- Lancement du père
- Le fork
 - Un nouveau processus a été créé, le fils du processus qui a appelé fork()
 - Ce processus est une copie conforme de son père.
 - La fonction fork() retourne 0 pour le fils et retourne le pid du fils pour le papa.
- Maîtriser le fil d'exécution du père et du fils
- Variables et descripteurs de fichiers
- **La synchronisation**

Héritage

Héritage

- Le fils hérite:
 - les propriétaires réels et effectifs, (`getuid`, `geteuid`, `getgid`),
 - le répertoire de travail, (`pwd`)
 - la valeur de nice, `nice()`
- Le fils n'hérite pas:
 - les verrous sur les fichiers détenus par le père, `flock()`
 - les signaux reçus et émis par le papa
 - Temps
- `Sleep(n)`: endort le processus jusqu'à ce que *n* secondes se soient écoulées, ou jusqu'à ce qu'un signal non-ignoré soit reçu..

Fonction nice()

- Exécuter un programme avec une priorité d'ordonnancement modifiée.
- Plus `nice()` est élevée, plus le processus est gentil vis à vis des autres, leur laissant un accès plus fréquent à l'ordonnanceur. (par défaut=10)
- Le Super-User peut indiquer un ajustement négatif.
- La priorité peut être ajustée avec **nice** dans l'intervalle -20 (le plus prioritaire) à 19 (le moins prioritaire).

Struct tms

```
struct tms { clock_t tms_utime; /* user time */  
            clock_t tms_stime; /* system time */  
            clock_t tms_cutime; /* user time of children */  
            clock_t tms_cstime; /* system time of children */  
};
```

tms_utime: contient le temps CPU écoulé en exécutant les instructions du processus appelant.

tms_stime: contient le temps CPU passé dans les fonctions système exécutées pour le compte du processus appelant

tms_cutime: contient la somme des valeurs de tms_utime et tms_cutime pour tous les processus fils terminés ayant été attendus.

tms_cstime: contient la somme des valeurs de tms_stime et tms_cstime pour tous les processus fils terminés ayant été attendus.

Héritage

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/times.h>
char buf[1024];           /* pour récupérer le répertoire de travail */
struct tms temps;         /* pour récupérer les nombres de clics */
main(){ int i;
    nice(10); /*augmentation de 10 de la valeur du nice avant le fork*/
    for (i=0;i<10000000;i++); /* une boucle consommatrice de CPU */
    if (fork()==0) {printf("caractéristiques du fils \n ");
    printf("uid=%d euid= %d egid=%d\n ",getuid(),geteuid(),getegid());
    printf(" répertoire de travail :%s\n ",getcwd(buf,1024));
    printf("nice : %d \n",nice(0)+20);
    times(&temps);
    printf("clics en mode utilisateur : %d \n", temps.tms_utime);
    printf("clics en mode système : %d \n\n ", temps.tms_stime);}
    else{ sleep(5); /* pour partir après la terminaison du fils */
    printf("caractéristiques du père \n ");
    printf("uid=%d euid= %d egid=%d\n ",getuid(),geteuid(),getegid());
    printf(" répertoire de travail :%s\n ",getcwd(buf,1024));
    printf("nice : %d \n",nice(0)+20);
    times(&temps);
    printf("clics en mode utilisateur : %d \n", temps.tms_utime);
    printf("clics en mode système : %d \n\n ", temps.tms_stime);}}
```

tests

- Voir le processus :`ps -ef`
 - Dans le père
 - Dans le fils
- Introduire une variable: `int n=10;`
 - Afficher la valeur de n dans le père
 - Afficher la valeur de n+10 dans le fils
 - Afficher la valeur de n dans le père



Copie des données

Copie des données

La copie des données lors de la création d'un processus ne se fait plus juste après sa création, mais lors d'un accès à la donnée en écriture

Copie des données

```
#include <sys/types.h>
int n=1000;
main()
{int m=1000, pid;
printf("Adresse de n dans le père: %p\n ", &n);
printf("Adresse de m dans le père: %p\n ", &m);
printf("1 valeur de m et n dans le père : %d %d\n ", m, n);
switch(pid=fork()){
case -1: perror("fork");exit(2);
case 0 : /* on est dans le processus fils*/
    printf("Adresse de n dans le fils: %p\n ", &n);
    printf("Adresse de m dans le fils: %p\n ", &m);
    printf("2 valeur de m et n dans le fils : %d %d\n ", m, n);
    m*=2;n*=2;
    printf("3 valeur de m et n dans le fils : %d %d\n ", m, n);
    sleep(3);
    printf("6 valeur de m et n dans le fils : %d %d\n ", m, n);exit(0);
default: /*on est dans le processus père*/
    sleep(2);
    printf("4 valeur de m et n dans le père : %d %d\n ", m, n);
    m*=3;n*=3;
    printf("5 valeur de m et n dans le père : %d %d\n ", m, n);
    sleep(2);
    exit(0);
}
}
```

Copie des données

Le fils a changé la valeur de ses variables `n` et `m`. Ceci a changé la valeur de sa propre variable `n` et `m`, mais pas celle du père. Voici donc notre première conclusion: **les variables du père et celles du fils sont totalement distinctes** ; même si elles portent le même nom, il ne s'agit pas des mêmes variables. En revanche, vous aurez remarqué qu'au moment du fork, le fils avait hérité des valeurs de toutes les variables de son père.

tests

- Changer les variables
- Changer la valeur de sleep
- Ps -ef



Les processus Zombies

Zombie

- Un terme désignant un ***processus*** qui s'est achevé, mais qui dispose toujours d'un identifiant de processus (PID) et reste donc encore visible dans la table des processus. On parle aussi de processus défunt.

Zombie

La terminaison d'un processus:

- *le système désalloue les ressources que possède encore celui-ci mais ne détruit pas son bloc de contrôle.*
- *Le système passe ensuite l'état du processus à la valeur ZOMBIE (un Z dans la colonne « statut »).*
- *Un signal est envoyé au père afin de l'informer de ce changement.*
- *Dès que le processus père a obtenu le code de fin du processus achevé au moyen des appels systèmes (wait ou waitpid), le fils est définitivement supprimé de la table des processus.*

Zombie

```
#include <stdio.h>
#include <sys/types.h>
main()
{
if (fork() == 0)
{
    printf("fin du processus fils de numéro %d \n ", getpid());
    exit(2);
}
sleep(30);
}
```

tests

- Ps -l: après le sleep
- Ps -l: avant le printf

tests

- Ps –ef
- Changer la valeur de sleep



Synchronisation

wait() et waitpid()

- **Les primitives qui permettent l'élimination des processus Zombies et permettent la synchronisation d'un processus sur la terminaison de ses descendants avec récupération des informations relatives à cette terminaison.**
- **L'appel système wait() permet à un processus appelant de suspendre son exécution en attente de recevoir un signal de fin de l'un de ses fils.**
- **La primitive waitpid permet de sélectionner parmi les fils du processus appelant un processus particulier.**

wait()

- Recherche de processus fils dans la table des processus : si le processus n'a pas de fils, le wait renvoie une erreur, sinon elle incrémente un compteur.
- S'il y a un zombie, il récupère les paramètres nécessaires (accounting) et libère l'entrée correspondante de la table des processus.
- S'il y a un fils mais pas de zombie, alors le processus se suspend (état endormi) en attente d'un signal.
- Lorsque le signal est reçu, la cause du décès est stockée dans la variable "status". 3 cas à distinguer : processus stoppé, processus terminé volontairement par exit, processus terminé à la suite d'un signal.
- Enfin le wait permet de récupérer le PID du processus fils : $\text{PID}=\text{wait}(\text{status})$.



Zombie

Cas normal: père attend le fils

```
#include <sys/types.h>
#include <stdio.h>
main(){
    int PID, status;
    if (fork() == 0)
        {printf("processus fils %d\n", getpid());
        exit(10);}
    PID = wait(&status);
    printf("processus père %d\n", getpid());
    printf("sortie du wait \n ");
    sleep(15);
    /* fils est bien terminé père toujours en place signal et infos reçus */
    printf("PID = %d status = %d\n", PID, status);
    exit(0);
}
```

Cas d'un Zombie: le père n'attend pas son fils et est toujours en vie après la mort de son fils

```
#include <sys/types.h>
#include <stdio.h>
main(){int PID, status;
if (fork()==0) {printf("processus fils %d\n", getpid());
exit(10);}
printf("processus père %d\n", getpid());
for (;;) /* le processus père boucle */
}
```

Cas où le père reçoit le signal de terminaison de son fils et n'exécute le wait qu'après

```
#include <sys/types.h>
#include <stdio.h>
main(){int PID, status;
if (fork()==0)
{printf("processus fils %d\n", getpid());
exit(10);}
printf("processus père %d\n", getpid());
sleep(15); /* le père endormi n'attend pas ... */
printf("sortie du 1 er sleep \n ");
PID=wait(&status);
printf("sortie du wait d\n "); /* il n'y a plus de Zombie */
sleep(15); /* tout a été pris en compte et traité */
printf("PID = %d status = %d\n", PID, status);
exit(0);}
```

Les pipes

Pipes

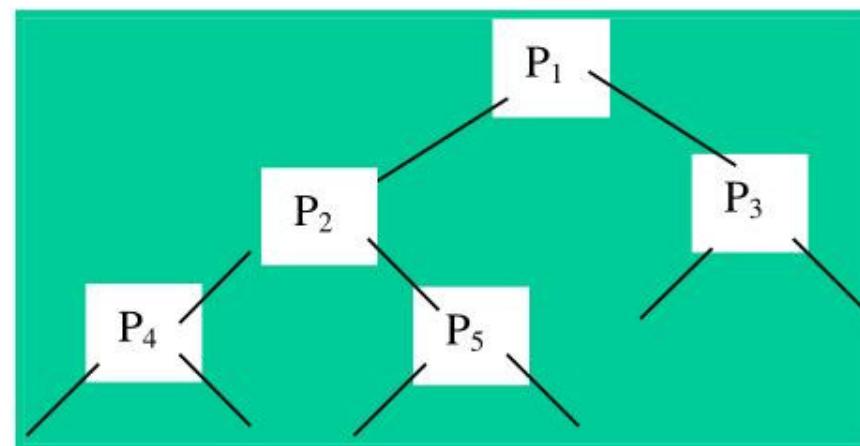
- Les pipes permettent de faire communiquer des processus d'une même machine : ce qu'un (ou plusieurs) processus écrit peut être lu par un autre.
- On utilise un pipe (tuyau) pour faire communiquer un processus et un de ses descendants ou des descendants

Pipes

- La nature d'un pipe est de type FIFO.
- Un pipe est unidirectionnel: un processus peut soit lire soit écrire dans un pipe
- L'information disparaît après lecture
- La taille du pipe est limitée de 4k à 20k dépendant du matériel.
- Le pipe est un objet du type fichier associé à 2 descripteurs de fichier et à 2 entrées dans la table des fichiers ouverts
- Le pipe n'a pas de noms dans l'arborescence de Unix et donc son ouverture ne peut se faire à l'aide de open mais plutôt de pipe
- La communication n'a lieu qu'entre processus de la même famille par héritage des descripteurs.

Pipes

Les pipes créés par le processus P2 ne permettent la communication qu'entre les processus P2, P4, P5 et leurs descendances. Les processus P1 et P3 ne peuvent accéder à ces pipes.

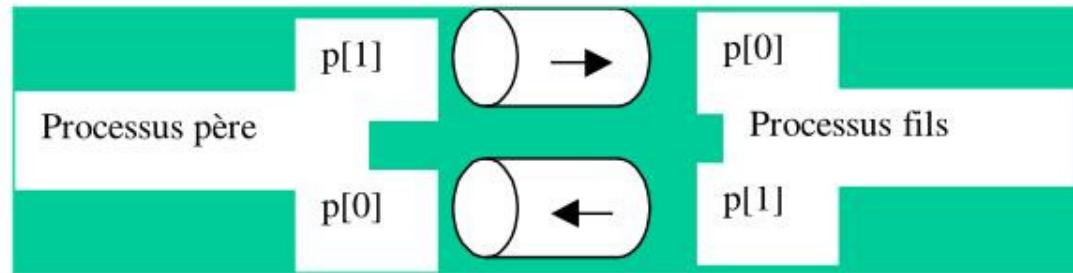


Pipes

- Creation:

```
int pipe(p)
```

```
int p[0];
```



L'appel système `pipe()` retourne 2 descripteurs de fichier, `p[0]` ouvert pour la lecture et `p[1]` ouvert pour l'écriture.

Pipes

- **Lecture**

Effectuée par un `read()` standard

`read(p[0],buf,5);` lecture de 5 caractères du pipe dans un buffer.

- **Ecriture**

Effectuée par un `write()` standard

Code

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/conf.h>
#include <stropts.h>
#include <stropts.h>
main(){int fils1, fils2, n, m, p[2]; char buf[5];
pipe(p); /* création de pipe */
if (fils1=fork()==0) /* création du premier fils */
{
    printf("je suis le fils producteur \n");
    printf("j'écris 5 caractères dans le pipe \n");
    write(p[1],"ABCDE",5);
    printf("fin d'écriture dans le pipe \n");
    exit(3);}
else /* le père crée le fils consommateur */
{
    if (fils2=fork()==0) /* création du deuxième fils */
    {
        printf("je suis le fils consommateur \n");
        read(p[0],buf,5); /* lecture du pipe */
        printf("voici les caractères lus \n");
        write(1,buf,5);/*affichage des caractères sur output standard*/
        printf("\n");
        exit(3);}
    else{   printf("processus père c'est fini .... \n");
        wait(&n);
        wait(&m);}
}
}
```

Tests:

- Remplir le pipe jusqu'à dépasser sa taille.
- Fermer le pipe avant de quitter le programme et essayer de le rouvrir.
- Lire d'un pipe vide.
- Lire deux fois les même données du pipe.

Les pipes nommés

Un pipe nommé a les mêmes caractéristiques qu'un pipe ordinaire plus:

- Il a un nom sous Unix comme un fichier ordinaire
- Il est créé de la même manière qu'un fichier spécial avec mknod
- Il est accessible par les processus n'ayant pas de lien de parenté
- Il subit les mêmes règles qu'un fichier ordinaire

Les pipes nommés

- **Création**

```
#include <sys/types.h>
#include <sys/stat.h>
int mknod (path, mode)
char *path; /* chemin */
int mode; /* type + permissions */
```

- Le mode est indiqué par trois sortes de flags : type de fichier (S_FIFO=0010000 pour un FIFO), permission, exécution particulière.

Les pipes nommés

- **Ouverture** : open (/uo/tcom/mypipe, O_RDWR, 2) ouverture bloquante pour la L/E, open (/uo/tcom/mypipe, O_RDWR | O_NDELAY, 2) ouverture non bloquante la L/E
- **Lecture** : read(d, buf, 2000) lecture d'un FIFO
- **Ecriture** : pour écrire dans le pipe il faut l'ouvrir avec open.
- **Suppression** : rm or unlink.
- **TP:** mknod mypipe

Ecriture non bloquante : PI

Lecture bloquante : P2

```
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
main(){int d, i;
char buf[2000];
d=open("/home/étudiant1/mypipe", O_RDONLY, 2);
if (d<0) {printf("problème de lecture du pipe \n");
exit(0);}
n=read(d, buf, 2000); /* lecture du pipe dans buf */
/* affichage du buffer sur l'écran */
printf("affichage du contenu du pipe \n");
write(1, buf, n);
printf("\n");}
```

Tests

- **Pour ce programme exécutez les tests suivants et observez ce qui se passe :**
 - Lancez en premier P1 et ensuite P2, est ce que la lecture et réalisée.
 - Lancez en premier P2 et ensuite P1 que se passe-t-il pour la lecture.
 - Lancez P1 ensuite P2 deux fois en lisant 2000.
 - Lancez P2 avec lecture de 4000 et P1 avec écriture de 2000.

Ecriture dans un pipe nommé plein

```
#include <stdio.h>
#include <fcntl.h>
main(){int d, i;
char buf[2000]; /*cette ouverture est bloquante, "blocage si mypipe est
plein*/
d=open("/home/étudiant1/mypipe",
O_RDWR, 2);
for (i=0;i<5121;i++) write(d, "P", 1); }
```