Architecture et OS

Objectifs et plan du cours : L'architecture des machines et de leur système d'exploitation (OS) constitue un ensemble de connaissances dont on peut se passer au niveau superficiel (utilisateur ou programmeur occasionnel) mais qui devient rapidement indispensable quand on approfondit le sujet. Ce cours ne fait qu'introduire ces différentes notions qui sont, en allant du centre vers la périphérie :

1. **Processeur et le langage machine** : au cœur de toute machine informatique, il y a des circuits électroniques (processeurs, unités de stockage, etc.) combinés dans des circuits logiques dont on peut appréhender le fonctionnement sur des modèles simplifiés.
2. **Système d'exploitation et la ligne de commande** : le système d'exploitation d'un ordinateur gère ses différentes fonctions ; elles s'appréhendent en mode terminal par la ligne de commande.
3. **Réseaux et protocoles** : l'interconnexion des ordinateurs entre eux est aujourd'hui omniprésente. L'architecture des réseaux et les protocoles de communication sont des éléments de compréhension indispensables.
4. **Robotique** : systèmes embarqués, objets connectés et robots sont en interactions avec les ordinateurs avec lesquels ils ont un lien de parenté.

Introduction historique :
Il y a un siècle, il n'y avait pas d'ordinateurs. Aujourd'hui, on en compte plusieurs milliards. Mais si l'ordinateur apparaît comme une invention relativement récente, sa mise au point à nécessité plusieurs millénaires d'évolution et de découvertes successives. Voici les grandes lignes de ce développement :
Au début, il y a les nombres, et leur écriture, dont le zéro.
Les premiers calculateurs mécaniques apparurent au cours du 17^{e} siècle.
1642 : Blaise Pascal (1623-1662) met au point sa Pascaline, limitée aux additions et soustractions.
1673 : Gottfried Leibniz met au point une machine capable d'effectuer multiplications, divisions et racines carrées.
1834 : Charles Babbage invente la machine analytique permettant d'évaluer des fonctions, sur le principe du métier à tisser Jacquard, programmé à l'aide de cartes perforées. 1843 : Ada Lovelace écrit le premier programme informatique pour la machine analytique qui ne sera jamais construite.
1936 : Alan Mathison Turing(1912-1954) publie un article présentant sa machine (dite aujourd'hui « machine de Turing »), le premier calculateur universel programmable.
1938 : Konrad Zuse invente le premier ordinateur à utiliser le système binaire au lieu du décimal.
1943 : J.Mauchly et J.Presper Eckert créent le premier ordinateur ne comportant plus de pièces mécaniques (l'ENIAC, pour Electronic Numerical Integrator And Computer).
1945 : John von Neumann publie un texte fondateur sur l'architecture utilisée dans la quasi-totalité des ordinateurs.
1948 : La firme Bell Labs invente le transistor qui permet de réduire la taille des ordinateurs.
1955 : IBM France invente le mot ordinateur pour éviter de traduire computer par calculateur.
1958 : Invention du circuit intégré : les puissances augmentent, les tailles et prix diminuent.
1969 : Ted Hoff d'Intel crée le premier microprocesseur ; Intel commercialise son 4004 fin 1971.
1975 : Début de l'industrie de l'ordinateur personnel avec l'Altair 8800 sur lequel se forment Bill Gates, Paul Allen, Steve Wozniak, Steve Jobs, etc.

1. Architecture

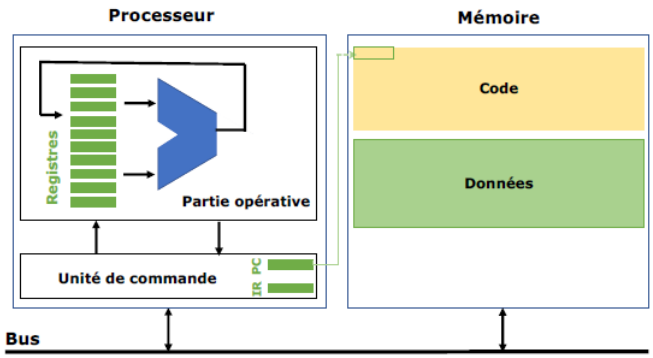
a. Le modèle de Von Neumann

Un ordinateur est principalement composé de deux circuits (le processeur et la mémoire), reliés entre eux par des « bus » de communication.

- Le processeur (CPU) est l'unité de traitement de l'information (instructions et données). Il exécute des programmes (suite d'instructions qui définissent un traitement à appliquer à des données).
- La mémoire centrale (RAM, dite aussi mémoire « vive ») est une unité de stockage temporaire des informations nécessaires à l'exécution d'un programme. Externe au processeur, elle stocke en particulier les instructions du programme en cours d'exécution et les données du programme (nombre, caractères alphanumériques, adresses mémoire, etc.).
- Les « bus » de communication sont les supports physiques des transferts d'information entre les différentes unités (bus de données, bus d'adresse, bus de commande).
- Les périphériques sont des unités connexes permettant de communiquer avec l'ensemble processeur-mémoire : clavier, écran, disque dur, réseau, imprimante, scanner, etc.

Le processeur exécute sans fin la suite des opérations suivantes, dite « boucle d'exécution » :

1. lire une instruction en mémoire (mise dans IR, *Instruction Register*)
2. décoder l'instruction
3. exécuter l'instruction
4. calculer l'adresse de l'instruction suivante (mise à jour PC,*Program Counteur*)

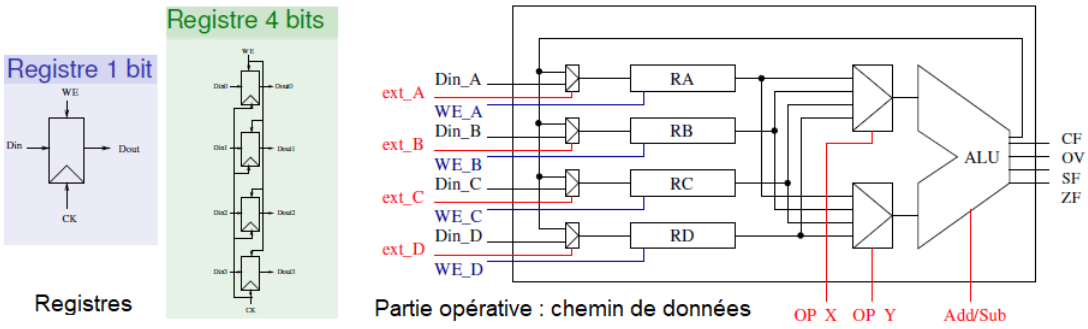


PC = Program Counteur

IR = Instruction Register

L'unité de commande récupère les instructions de la mémoire et les analyse, puis séquence les actions élémentaires pour leur réalisation. Elle orchestre donc les différentes actions à réaliser par le processeur et commande les interactions entre le chemin de données et la mémoire. Elle est composée d'un ou plusieurs automates séquentiels enchainant les actions à réaliser.

La partie opérative est au service de l'unité de commande. Elle contient les outils pour réaliser les actions élémentaires ordonnées par l'unité de commande, notamment une unité arithmétique et logique (ALU) et des registres internes. Un banc de registres désigne un ensemble de registres, identifiés chacun par un nom ou un numéro, et pouvant stocker un mot binaire de n bits. Un registre n bits est composé de la mise en parallèle de n registres 1 bit (registre 1 bit : cellule mémorisante élémentaire). Ces registres sont utilisés pour stocker les opérandes des opérations, ainsi que leurs résultats. Le chemin de données contient le banc de registres, l'ALU, et la connectique permettant de relier les registres en entrées et en sortie de l'ALU (multiplexeurs).



La mémoire peut être vue comme un tableau de n lignes ou mots de p bits.
L'index d'un mot définit son adresse :

adresse du 1er mot $= 0$, adresse du $2^e = 1$, ..., adresse du dernier mot $= n - 1$

Les mots sont en général des octets (soit $p = 8$) : on dit que l'unité adressable est l'octet.
Les cases mémoires peuvent parfois mémoriser des mots de seize, trente-deux ou soixante-quatre bits.
Les données et les instructions du programme en cours d'exécution sont stockées dans des zones différentes de la mémoire, notées « Données » et « Code » sur le schéma ci-dessus.
Il existe différents types de mémoire :

- La mémoire non réinscriptible ou ROM : assez lente d'accès, elle contient le code de démarrage ou l'ensemble du code dans certains petits systèmes embarqués
- La mémoire vive ou RAM : volatile et rapide d'accès, elle contient le code et les données manipulées par le processeur pendant l'exécution d'un programme
- La mémoire Flash : non volatile, réinscriptible (mais nombre d'écritures limitées) et rapide d'accès, elle est utilisée pour stocker du code ou les données non modifiables d'un programme ou bien dans les clés USB ou certains appareils numériques pour du stockage à long terme.

La capacité mémoire correspond au nombre de mots (octets) que la mémoire peut stocker.
Un ordinateur ayant une capacité de stockage de 16 Go (c'est peu aujourd'hui, un ordinateur individuel ayant couramment plus d'un tera-octet de mémoire) a $16 \times 2^{30} \times 8 = 137\,438\,953\,472$ circuit-mémoires 1 bit. Si cette mémoire est organisée en cases mémoires de soixante-quatre bits, il y en a 2 147 483 648 pouvant chacune mémoriser un mot de soixante-quatre bits.

1 kilo-octet	1 ko	2^{10} octets	1024 octets
1 mega-octet	1 Mo	2^{20} octets	1 048 576 octets
1 giga-octet	1 Go	2^{30} octets	1 073 741 824 octets
1 tera-octet	1 To	2^{40} octets	1 099 511 627 776 octets

Le processeur initie les transferts de données entre le processeur et la mémoire et indique à la mémoire : l'adresse du mot à transférer, la taille du mot à transférer, le sens du transfert (*store* ou *load*) et la donnée à écrire en cas d'écriture (*store*).
Lors d'un transfert de données : le décodeur d'adresse sélectionne la ligne correspondant à l'adresse demandée sur le bus d'adresse.
La commande indique l'opération : lecture ou écriture. Si il y a lecture, les données lues sont mises sur le bus de données ; si il y a écriture : les données à écrire sont présentes sur le bus de données .

b. Jeu d'instructions

La vue externe d'un processeur peut être définie par l'ensemble des instructions qu'il est capable de traiter. Ce jeu d'instructions est étroitement lié à l'architecture interne de l'ordinateur. Il précise aussi quels sont les registres du processeur manipulables par le programmeur.

Définition 5.1 (Jeu d'instructions) Le jeu d'instructions d'un processeur est défini par :

- l'ensemble des instructions qu'il peut effectuer
- le système d'encodage de ces instructions en binaire

Remarques : Il existe de multiples jeux d'instructions, autant qu'il y a de types de microprocesseurs. Les jeux d'instructions principaux des années 1980 à 2010 ont pour nom x86, PowerPC, SPARC, ARM, Alpha, VAX, 68000, MIPS.
Le x86, par exemple, regroupe les microprocesseurs compatibles avec le jeu d'instructions du processeur Intel 8086 (cette série est nommée IA-32 à partir du Pentium). Les processeurs fabriqués selon l'architecture MIPS ont été utilisés dans les systèmes SGI et plusieurs systèmes embarqués, comme les ordinateurs de poche, les routeurs Cisco et les consoles de jeux vidéo (Nintendo 64 et Sony PlayStation, PlayStation 2 et PSP).
Nous allons étudier un jeu d'instructions MIPS (*Microprocessor without Interlocked Pipeline Stages*), mais ce n'est qu'à titre d'exemple, pour essayer de comprendre ce que peut être un jeu d'instruction. Il existe en fait plusieurs jeux d'instructions MIPS : MIPS I, MIPS II, MIPS III, MIPS IV, et MIPS V ainsi que MIPS32 et MIPS64. MIPS32 et MIPS64, qui se basent sur MIPS II et MIPS V, ont été introduits comme jeux d'instructions normalisés. C'est ce MIPS32, réputé simple, que nous allons étudier.

DÉFINITION 5.2 (INSTRUCTION) En langage machine, une instruction est une commande formatée, donnée au processeur, qui définit :
- Le traitement à effectuer maintenant
- La prochaine instruction à exécuter

Remarques :
- Un traitement à effectuer est une opération (addition, opération logique, opération mémoire) accompagnée des opérandes sur lesquelles elle porte (la ou les opérandes sources, l'opérande destination s'il y en a une).
- La prochaine instruction à exécuter est celle implantée en mémoire à la suite de l'instruction courante (succession implicite) ou bien celle définie explicitement par des instructions de saut.

R-Type Instructions (Opcode 000000)

opcode (6)	rs (5)	rt (5)	rd (5)	sa (5)	function (6)

Instruction		Function
add	rd, rs, rt	100000
addu	rd, rs, rt	100001
and	rd, rs, rt	100100
break		001101
div	rs, rt	011010
divu	rs, rt	011011
jalr	rd, rs	001001
jr	rs	001000
mfhi	rd	010000
mflo	rd	010010
mthi	rs	010001
mtlo	rs	010011
mult	rs, rt	011000
multu	rs, rt	011001
nor	rd, rs, rt	100111
or	rd, rs, rt	100101
sll	rd, rt, sa	000000
sllv	rd, rt, rs	000100
slt	rd, rs, rt	101010
sltu	rd, rs, rt	101011
sra	rd, rt, sa	000011
srav	rd, rt, rs	000111
srl	rd, rt, sa	000010
srlv	rd, rt, rs	000110
sub	rd, rs, rt	100010
subu	rd, rs, rt	100011
syscall		001100
xor	rd, rs, rt	100110

I-Type Instructions (All opcodes except 000000, 00001x, and 0100xx)

opcode (6)	rs (5)	rt (5)	immediate (16)

Instruction		Opcode	Notes
addi	rt, rs, immediate	001000	
addiu	rt, rs, immediate	001001	
andi	rt, rs, immediate	001100	
beq	rs, rt, label	000100	
bgez	rs, label	000001	rt = 00001
bgtz	rs, label	000111	rt = 00000
blez	rs, label	000110	rt = 00000
bltz	rs, label	000001	rt = 00000
bne	rs, rt, label	000101	
lb	rt, immediate(rs)	100000	
lbu	rt, immediate(rs)	100100	
lh	rt, immediate(rs)	100001	
lhu	rt, immediate(rs)	100101	
lui	rt, immediate	001111	
lw	rt, immediate(rs)	100011	
lwc1	rt, immediate(rs)	110001	
ori	rt, rs, immediate	001101	
sb	rt, immediate(rs)	101000	
slti	rt, rs, immediate	001010	
sltiu	rt, rs, immediate	001011	
sh	rt, immediate(rs)	101001	
sw	rt, immediate(rs)	101011	
swc1	rt, immediate(rs)	111001	
xori	rt, rs, immediate	001110	

J-Type Instructions (Opcode 00001x)

opcode (6)	target (26)

Instruction		Opcode	Target
j	label	000010	coded address of label
jal	label	000011	coded address of label

Coprocessor Instructions (Opcode 0100xx)

opcode (6)	format (5)	ft (5)	fs (5)	fd (5)	function (6)

Instruction		Function	Format
add.s	fd, fs, ft	000000	10000
cvt.s.w	fd, fs, ft	100000	10100
cvt.w.s	fd, fs, ft	100100	10000
div.s	fd, fs, ft	000011	10000
mfc1	ft, fs	000000	00000
mov.s	fd, fs	000110	10000
mtc1	ft, fs	000000	00100
mul.s	fd, fs, ft	000010	10000
sub.s	fd, fs, ft	000001	10000

	000	001	010	011	100	101	110	111
000	R-type		j	jal	beq	bne	blez	bgtz
001	addi	addiu	slti	sltiu	andi	ori	xori	
010								
011	llo	lhi	trap					
100	lb	lh		lw	lbu	lhu		
101	sb	sh		sw				
110								
111								

OPCODE map

	000	001	010	011	100	101	110	111
000	sll		srl	sra	sllv		srlv	srav
001	jr	jalr						
010	mfhi	mthi	mflo	mtlo				
011	mult	multu	div	divu				
100	add	addu	sub	subu	and	or	xor	nor
101			slt	sltu				
110								
111								

FUNC map of R-type instructions

Codage des instructions :
Le jeu d'instructions définit le format de codage binaire des instructions (il ne faut pas oublier que tout doit être traduit en binaire pour la machine). La table ci-dessus donne le principe d'encodage de l'architecture MIPS32 où l'instruction à effectuer est associée à un mnémonique indiquant l'opération, par exemple `add`, `andi`, `lw`, `sw`, `j`, `beq`. Ces mnémoniques sont convertis en codes binaires selon le format correspondant.
L'`OpCode` est écrit sur les 6 premiers bits (bits 31 à 26) : il permet de distinguer les instructions au format I ou J, pour celles qui sont au format R, il vaut 0. Les différents opérandes qui suivent l'`OpCode` sont, soit des constantes, nommées « immédiats » (-1, 0xFFFF, 3, etc.), soit des registres du processeur, identifiés par un numéro ($0, $1, ..., $31), soit encore des codes de fonctions (pour les instructions au format R notamment, mais aussi pour les instructions coprocesseur).

Instructions arithmétiques et logiques

Ces instructions utilisent l'ALU pour réaliser un calcul sur des données. Le résultat est toujours stocké dans un registre. Les opérandes sources sont des registres et/ou des constantes entières codées sur 16 bits (pour une structure de mémoire en cases de 2 octets) et étendues sur 32 à l'exécution (pour un processeur 32 bits).
Exemples d'instructions du MIPS32 :

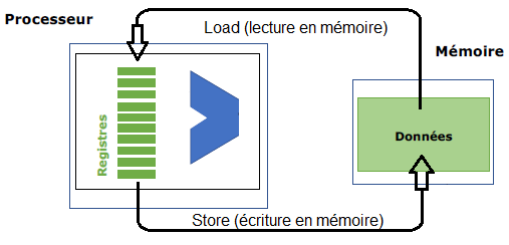
- Addition entre 2 registres : `add $4, $2, $3`
- Addition entre un registre et un immédiat : `addi $1, $2, 350`
- OU logique entre 2 registres : `or $2, $4, $3`
- OU logique entre un registre et un immédiat : `ori $2, $3, 0x00F0`
- Affectation de registre : mettre une valeur sur les 16 bits de poids fort : `lui $2, 0xABCD`

L'encodage MIPS32 suit un format spécifique dont un exemple est :

> L'instruction `addi $1, $2, 3` permet d'ajouter l'immédiat 3 (en binaire 0000000000000011) au registre n°2 (00010) et met le résultat dans le registre n°1 (00001). Cela est codé en binaire, selon le format I du MIPS32, 001000-00001-00010-0000000000000011. J'ai ajouté les tirets de séparation pour distinguer codage de l'opération, registre où stocker le résultat, registre où prendre la donnée et immédiat à ajouter ; mais pour l'unité de commande et le processeur, cette instruction est simplement 00100000001000100000000000000011, écrite sur 4 octets.
> Pour simplifier, on utilise parfois le système hexadécimal : il faut regrouper alors les bits par 4 et traduire chaque quartet par un caractère hexadécimal. L'instruction `addi $1, $2, 3` devient, en quartets 0010-0000-0010-0010-0000-0000-0000-0011 d'où, en hexadécimal `0x20220003`.

Instructions de transfert mémoire

Ces instructions lisent (*load*) ou écrivent (*store*) des données en mémoire.
La syntaxe des instructions d'accès mémoire est au format I, mais notée `opcode rt, immediate(rs)`
Le sens du transfert est défini par l'`opcode` : `lw` (*load word*) ou `sw` (*store word*).
L'adresse accédée est, en définitive, obtenue en ajoutant le contenu du registre `rs` à l'immédiat `immediate` sur 16 bits (l'ALU est utilisée pour ce calcul d'adresse). Le registre `rt` contient la valeur à écrire en mémoire (s'il s'agit d'une écriture) ou recevra la valeur lue en mémoire (s'il s'agit d'une lecture).



Exemples MIPS32 :

- `lw $4,0($3)` : lecture du mot de la mémoire contenu à l'adresse `0+$3` et rangement dans le registre `$4`.
- `sw $4,0($3)` : écriture du mot contenu dans le registre `$4` à l'adresse `0+$3` de la mémoire.

L'encodage MIPS32 de ces instructions suit le même principe que précédemment :
Le code opération de `lw` est 100011, celui de `sw` est 101011.
Ces instructions codées en hexadécimal sont `0x8c640000` (pour `lw $4,0($3)`) et `0xac640000` (pour `sw $4,0($3)`). Pour obtenir ces codes, je les ai écrits dans MARS, un logiciel adapté (voir plus loin).
Examinons ces écritures :

- `0x8c640000` s'écrit en binaire 1000-1100-0110-0100-0000-0000-0000-0000, soit 100011-00011-00100-0000000000000000, soit `lw, $3, $4, 0` (l'immédiat est à la fin, le registre de destination au milieu).
- `0xac640000` s'écrit en binaire 1010-1100-0110-0100-0000-0000-0000-0000, soit 101011-00011-00100-0000000000000000, soit `sw, $3, $4, 0` (l'immédiat est à la fin, le registre de lecture au milieu).

Instructions de rupture de séquence

Ces instructions permettent de casser l'exécution séquentielle par défaut du code en spécifiant quelle sera la prochaine instruction à exécuter.
Les ruptures de séquence sont les sauts, les branchements et les appels et retours de procédure. Le saut donne au compteur de programme une adresse absolue tandis que le branchement ajoute un déplacement positif ou négatif au compteur de programme (sa valeur est obtenue dans l'instruction suivant le branchement). Les sauts et les branchements peuvent être inconditionnels ou conditionnels en fonction du résultat d'un test qui peut être effectué de différentes manières.

En MIPS32, le format J est utilisé pour les instructions de saut inconditionnel avec adressage direct. La valeur de PC est systématiquement modifiée. Il y a deux formes : `j label` (met dans PC la valeur associée à l'étiquette label) et `jr Ri` met dans PC la valeur contenue dans le registre `Ri`
Pour les sauts conditionnels, l'adressage est relatif, le format est alors de type I. Selon le résultat d'un test d'égalité ou de comparaison entre les contenus de deux registres, il va y avoir branchement vers une étiquette ou bien déplacement en séquence (l'instruction suivante).

- `beq $2, $3, fin` (*branch if equal*) : si $2=$3 alors il y a branchement à l'étiquette `fin`, sinon déplacement en séquence, c'est-à-dire aller à l'instruction suivante.
- `bne $2, $3, fin` (*branch if not equal*) : si $2≠$3 alors il y a branchement à l'étiquette `fin`, sinon déplacement en séquence
- `bgez $3, label` (*branch if greater or equal than zero*) : si $3⩾ 0 alors il y a branchement à l'étiquette `label`, sinon déplacement en séquence. De même pour
- `bgtz` (*greater than*) , `blez` (*less or equal*) et `bltz` (*less*) où la comparaison est faite par rapport à zéro (`z`).

Le système d'étiquetage relatif nécessite deux compilations (traduction du langage de haut niveau en langage assembleur), et ce n'est que lorsque toutes les instructions sont en place que le déplacement relatif peut être implanté dans les zones immédiate prévues à cet effet (laissées à 0 lors de la 1re compilation) [1].

Instructions système

Ces instructions demandent un service au système : arrêt du programme, affichage d'une valeur et d'une façon générale toutes les entrées–sorties sont prises en charge par la routine système `syscall`.
Voici un exemple de programme assembleur qui demande l'affichage d'une chaîne de caractères (« Bonjour ») puis qui stoppe le programme.

- L'étiquette `str` référence l'adresse de la chaîne de caractères en mémoire
- $a0 est l'adresse de la chaîne passée comme argument à `syscall`

1. Pour des explications plus complètes voir par exemple `https://slideplayer.fr/slide/5182541/`

- `.data` : directive indiquant que ce qui suit est placé dans le segment de données de la mémoire
- `.asciiz` : directive indiquant que ce qui suit est une chaîne terminée par le caractère 0
- `.text` : directive indiquant que ce qui suit est placé dans le segment de texte de la mémoire

```
        .data
str:    .asciiz             "Bonjour"
        .text
main:   ori $v0, $zero, 4   # $v0 <- 4
        la $a0, str         # $a0 <- str
        syscall             # affiche "Bonjour"
        ...
        ori $v0, $zero, 10  # $v0 <- 10
        syscall             # appel système de
                            # terminaison de programme
```

Le langage assembleur

Les microprocesseurs sont conçus pour être programmables à l'aide d'instructions en langage machine, c'est-à-dire en code binaire (impossible à écrire directement ou à lire pour un humain). L'assembleur utilise des instructions en texte, appelé code source, qui sont compilées pour produire du code binaire. C'est un langage de bas niveau (son fonctionnement est très proche du langage machine) : chaque ligne ne contient qu'une instruction, celle-ci ne pouvant exécuter que des actions simples (contrairement aux langages de programmation de plus haut niveau). Un programme assembleur est exécuté ligne par ligne, de haut en bas. Le seul moyen de faire répéter une tâche à un programme est d'utiliser un saut (`jump`) ou un appel à une procédure (`syscall`).
On a déjà donné quelques exemples d'instructions pour l'architecture MIPS32. Le langage assembleur est une représentation exacte du langage machine, spécifique à chaque architecture de processeur.
En plus de coder les instructions machine, les langages assembleur ont des directives supplémentaires pour assembler des blocs de données et affecter des adresses aux instructions en définissant des étiquettes ou labels.
L'immense majorité des programmes sont maintenant écrits en langages de haut niveau (compilés ou interprétés à la volée, directement en langage machine) mais il reste des domaines où l'assembleur présente un réel intérêt (calculs complexes, routines, tâches très dépendantes du système). Dans ce cours, son seul intérêt est de nous permettre de comprendre ce qui se passe dans le processeur.

```
1   # programme valeur absolue
2     .data
3   n: .word 0xFFFF # allocation d'un entier initialisé à une (-1)
4     .text
5           lui $3, 0x1001      # chargement de l'adresse de l'entier dans $3
6           lw  $4, 0($3)       # lecture de la valeur dans $4 en mémoire
7           bgez $4, affiche
8           sub $4, $0, $4
9           bgez $4, affiche
10  affiche :
11          ori $4, $0, 1       # affichage de l'entier contenu dans $4
12          syscall
13          ori $2, $0, 10      # fin du programme
14          syscall
```

Registers

Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x10010000
$a0	4	0x00000000

Data Segment

Address	Value (+0)	Value (+4)
0x10010000	0x0000ffff	0x00000000

Execute

Déroulement pas à pas du programme

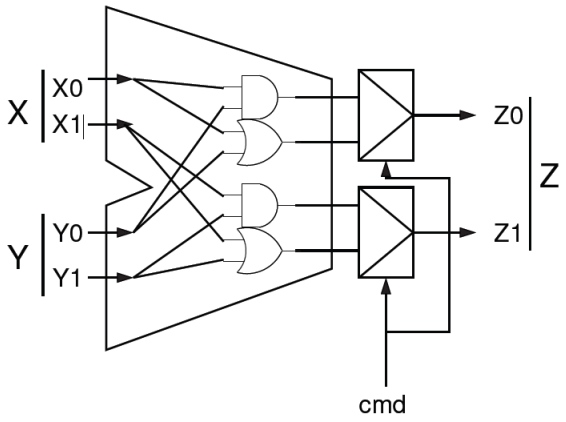
Text Segment

Address	Code	Basic		Source
0x00400000	0x3c031001	lui $3,0x00001001	5:	lui $3, 0x1001 # chargement de l'adresse de l'entier dans $3
0x00400004	0x8c640000	lw $4,0x00000000($3)	6:	lw $4, 0($3) # lecture de la valeur dans $4 en mémoire
0x00400008	0x04810002	bgez $4,0x00000002	7:	bgez $4, affiche
0x0040000c	0x00042022	sub $4,$0,$4	8:	sub $4, $0, $4
0x00400010	0x04810000	bgez $4,0x00000000	9:	bgez $4, affiche
0x00400014	0x34040001	ori $4,$0,0x00000001	11:	ori $4, $0, 1 # affichage de l'entier contenu dans $4
0x00400018	0x0000000c	syscall	12:	syscall
0x0040001c	0x3402000a	ori $2,$0,0x0000000a	13:	ori $2, $0, 10 # fin du programme
0x00400020	0x0000000c	syscall	14:	syscall

Le logiciel MARS[2] (*MIPS Assembler and Runtime Simulator*) est un IDE pour la programmation en langage assembleur dans l'architecture MIPS.
Dans l'onglet `Edit`, on écrit le programme en assembleur MIPS. La compilation est effectuée en cliquant sur l'item `Assemble` du menu `Run`. Á partir de là, le programme est disponible dans l'onglet `Execute` : dans le `Text segment` se trouvent les instructions compilées, dans le `Data segment` se trouvent les données mémoires et dans l'onglet `Register` se trouvent l'état des 32 registres généraux. L'exécution du programme pas-à-pas permet de visualiser l'évolution des mémoires et des registres. Á l'étape 2, par exemple (la figure ci-dessus), la valeur de l'adresse est inscrite dans le registre $3 (`0x10010000`).
Á l'étape suivante, le registre $4 va contenir la valeur -1 (`0x0000ffff`, visible dans le `Data segment`) qui a été stockée en mémoire à la compilation.

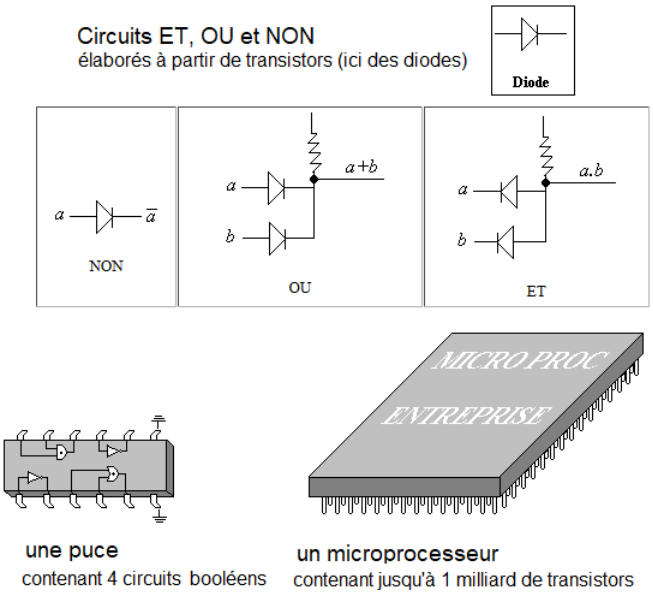
c. Unité arithmétique et logique

Les processeurs disposent d'instructions pour réaliser des opérations logiques.
On a vu qu'en MIPS32, l'instruction `or $3, $2, $1`, par exemple, permet de comparer les contenus des registres $1 et $2 et de mettre le résultat dans le registre $3. Ce « OU » logique est effectué dans l'ALU : les contenus des registres $1 et $2 sont placés sur les entrées X et Y de celle-ci tandis que la sortie Z recueille le résultat et le place dans le registre $3. La sortie de l'ALU présente plusieurs résultats possibles, selon les circuits logiques précâblés qu'elle contient. La sortie correspondant au « OU » logique est sélectionnée grâce à un multiplexeur commandé par l'instruction réalisée. Pour un « OU », la commande du multiplexeur pourrait être 1 tandis qu'avec la commande 0, le multiplexeur sélectionnerait un « ET » logique, réalisant l'instruction `and $3, $2, $1`.
La figure ci-dessous montre, pour des mots de 2 bits, la sélection des opérations « OU » et « ET » à la sortie de l'ALU par des multiplexeurs commandés par la valeur de `cmd` : si `cmd`=1 alors l'opération réalisée est un « OU » et sinon (`cmd`=0) l'opération réalisée est un « ET ».



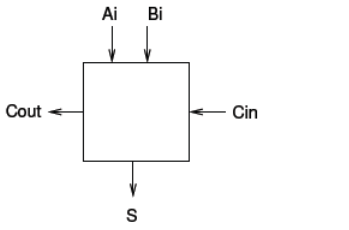
La réalisation physique des circuits est aujourd'hui électronique – un autre domaine que l'informatique – ce qui permet une très grande miniaturisation. Nous n'entrerons pas dans les détails, mais le composant de base en électronique est le transistor. Il possède trois broches (des pattes métalliques) sur lesquelles on peut appliquer une tension électrique (représentée par 0 ou 1). Sur ces trois broches, il y en a deux entre lesquelles circule un courant, et une troisième qui commande le courant. On l'utilise le plus souvent comme un interrupteur commandé par sa troisième broche : le courant qui traverse les deux premières broches passe ou ne passe pas selon ce qu'on met sur la troisième.
Sur l'illustration ci-dessous, ce sont des diodes qui réalisent les circuits logiques câblés simples « NON »,« ET » et « OU ». Dans une puce et à fortiori dans un microprocesseur, les câblages sont d'une autre nature puisqu'ils utilisent des transistors.

2. MARS est téléchargeable gratuitement sur le site de l'Université du Missouri `http://courses.missouristate.edu/kenvollmar/mars/`

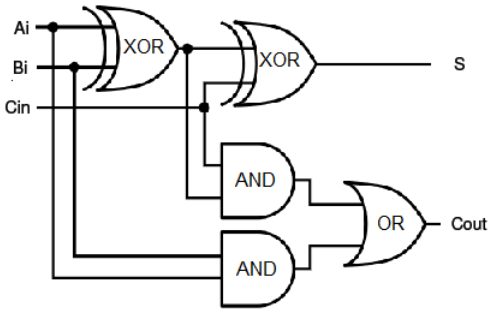


Réalisation d'un additionneur

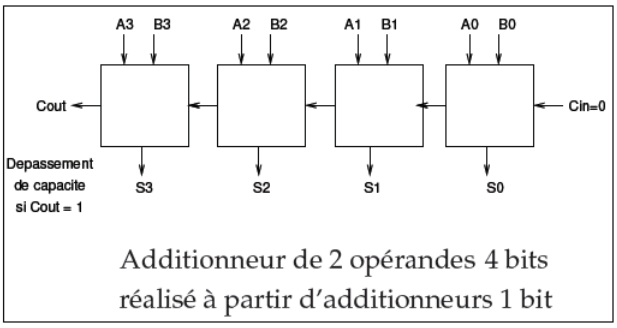
Un additionneur 1 bit permet d'additionner 2 bits (`Ai`, `Bi`) et éventuellement un autre bit (`Cin`), habituellement la retenue d'une autre addition. `Ai`, `Bi` et `Cin` constituent les entrées de l'additionneur et le résultat de l'addition en binaire est donnée par deux sorties `Cout` et `S` (`Cout` représente le bit de poids le plus fort et `S` le bit de poids le plus faible du résultat). La table de vérité de ce circuit comporte 8 lignes (voir ci-dessous) et sa réalisation pratique est obtenue par un câblage contenant 2 portes XOR, 2 portes AND et une porte OR.



Additionneur de 2 opérandes
un bit avec 1 bit de retenue



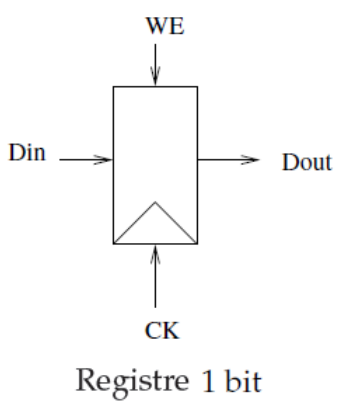
a_i	b_i	c_{in}	s	c_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



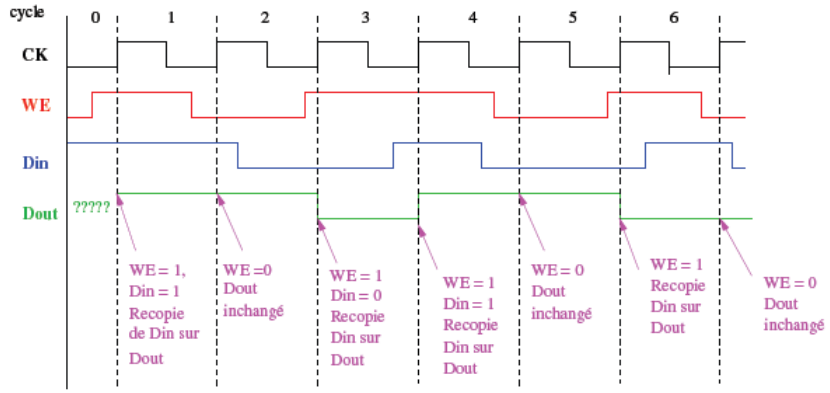
En chaînant plusieurs additionneurs un bit, on peut fabriquer des additionneurs capables de traiter des mots de longueurs arbitraires. Mais ce procédé est lent car la propagation de la retenue se fait au coup d'horloge : s'il y en a beaucoup, la durée d'une addition sera proportion plus longue. Les additionneurs efficaces réalisent un compromis entre vitesse et complexité (techniques de prédiction de la retenue). Ils sont au cœur de l'ALU.

Réalisation d'un registre

Les registres correspondent à de la mémoire très temporaire à disposition immédiate de l'ALU dans le processeur.
L'illustration montre l'interface d'un registre 1 bit : sur cette figure, `CK` désigne le signal d'horloge globale cadençant le processeur. `WE` signifie *Write Enable*. D'un point de vue macroscopique, le registre 1 bit mémorisant sur front montant a la fonctionnalité suivante : lorsque `CK` passe de 0 à 1 (front montant) si `WE` = 1 alors la valeur de `Din` est recopiée sur `Dout`. Sinon `Dout` est inchangé (garde la valeur précédente). Á tout autre instant (front descendant donc) ou si `WE` = 0 alors `Dout` est inchangé.
La plupart des circuits électroniques complexes sont synchronisés par une horloge : un signal d'horloge est un signal électrique oscillant qui rythme les actions d'un circuit. Sa période est appelée cycle d'horloge. Le chronogramme de l'illustration montre les signaux d'entrées et la valeur de sortie `Dout` d'un registre 1 bit sur plusieurs cycles d'horloge en fonction de ses entrées.



Registre 1 bit

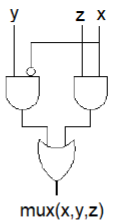


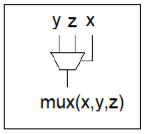
Chronogramme pour un registre 1 bit

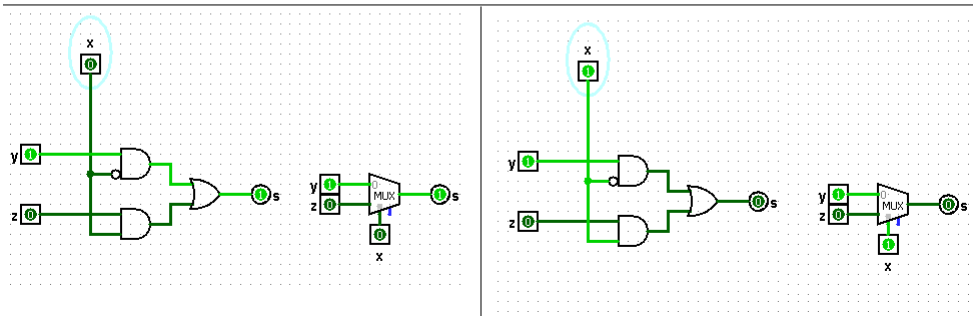
Simulations

Le logiciel LOGISIM [3] est un outil d'édition et de simulation de circuits. Construisons un multiplexeur 1 bit : cette fonction logique, notée `mux`, ayant été étudiée au chapitre 1, rappelons sa table de vérité et son expression algébrique. Á partir de celle-ci, on peut déduire les portes logiques nécessaires et le câblage entre ces portes. Le circuit peut être dessiné directement sur LOGISIM et testé : en mettant des valeurs 0 ou 1 dans les entrées `x` (la commande), `y` et `z` , on observe que la donnée de sortie respecte bien la table de vérité de `mux(x,y,z)`, pour les huit combinaisons des trois variables d'entrée.

MUX			
x	y	z	mux(x,y,z) = (non(x) et y) ou (x et z)
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1





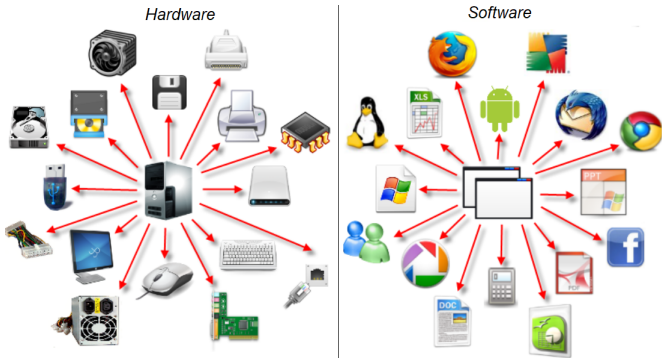


3. LOGISIM est disponible gratuitement sur le site `http://www.cburch.com/logisim/`

2. Systèmes d'exploitation

Du Hardware au Software

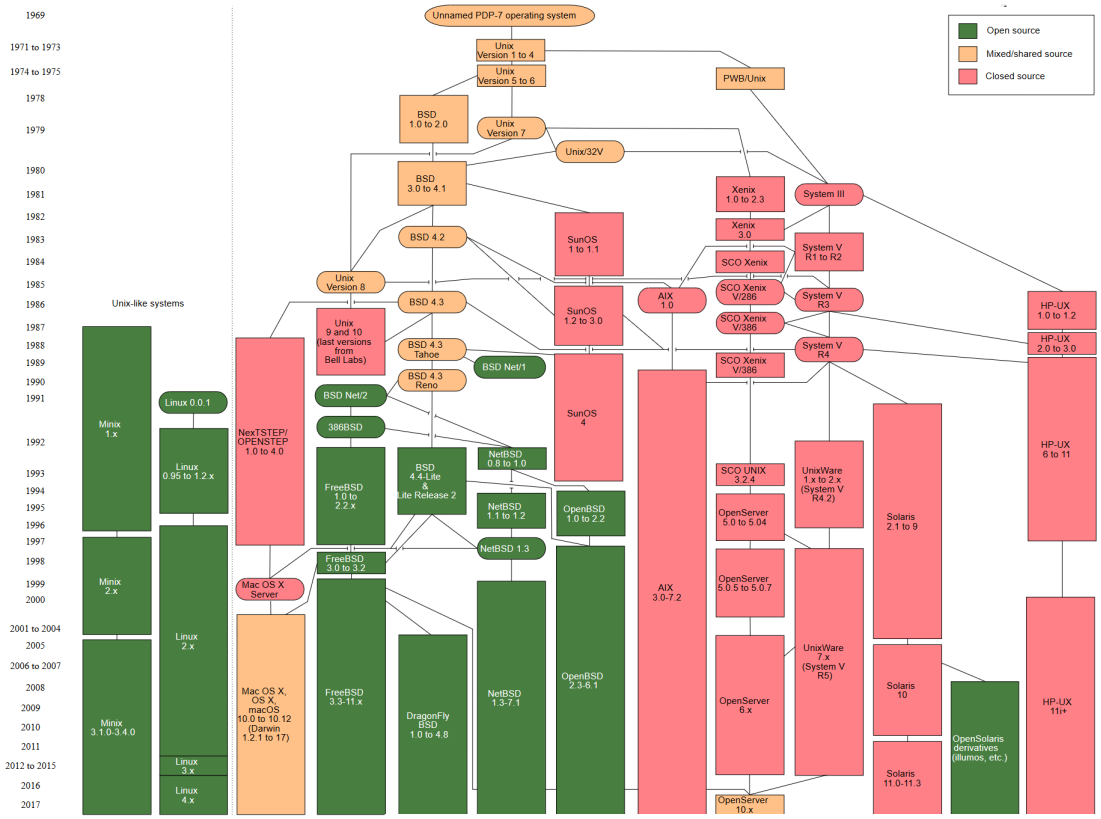
Les deux concepts *hardware* et *software* sont deux domaines différents. Le *hardware* désigne le matériel physique constituant l'ordinateur – le PC (de l'anglais *Personnal Computer*) – et les matériels externes tandis que le *software* est la partie logicielle. Le mot français logiciel a été introduit en 1969 pour remplacer l'anglicisme *software*. Le *software* d'un ordinateur désigne donc l'ensemble des programmes et des procédures utilisés par le *Hardware*.
Á la frontière entre ces deux domaines, on trouve le *firmware*. Le mot est forgé sur l'adjectif *firm* (résistant, ferme), un état intermédiaire entre *soft* (doux) et *hard* (dur). Un *firmware* (micrologiciel, microcode, logiciel interne, logiciel embarqué ou microprogramme) est un programme intégré dans un matériel informatique (ordinateur, photocopieur, automate, disque dur, routeur, appareil photo numérique, etc.) pour qu'il puisse fonctionner. Ce genre de programme permet à un matériel informatique d'évoluer (via des mises à jour), d'intégrer de nouvelles fonctionnalités, sans avoir besoin de revoir complètement le design du *hardware*. Le *firmware* des ordinateurs – le BIOS ou l'EFI (*Extensible Firmware Interface*) – est exécuté par le CPU (le processeur, de l'anglais *Central Processing Unit*). Ce sont les codes de gestion de la carte mère, d'une carte vidéo ou SCSI (de l'anglais *Small Computer System Interface* : standard définissant un bus informatique reliant un ordinateur à des périphériques).



Un système d'exploitation (OS, de l'anglais *Operating System*) est une couche logicielle permettant de faire l'interface entre le matériel et les applications des utilisateurs de la machine. Il est notamment responsable de la gestion des ressources de la machine comme le processeur, la mémoire et les périphériques comme les disques durs, les cartes réseaux (Wifi, Ethernet), la carte graphique, la carte son, etc. Le système d'exploitation gère les demandes et les ressources, en évitant les interférences entre les différents logiciels qui s'exécutent simultanément.
Le système d'exploitation est lancé après le *firmware* d'amorçage (en anglais *bootloader*) lors de la mise en marche de l'ordinateur. Il offre une suite de services généraux facilitant la création de logiciels applicatifs et sert d'intermédiaire entre ces logiciels et le *hardware*.
Il existe sur le marché des dizaines de systèmes d'exploitation différents, très souvent livrés avec l'appareil informatique. C'est le cas de Windows, Mac OS, Irix, Symbian OS, GNU/Linux (pour lequel il existe de nombreuses distributions) ou Android. UNIX est un des premiers systèmes d'exploitation : apparu dans les années 70, il fut le système d'exploitation le plus utilisé jusqu'à la fin des années 80. Le développement d'une multitude de versions propriétaires pour différentes architectures et l'émergence au début des années 90 de Windows, plutôt adapté aux utilisateurs non-informaticiens, ont causé une énorme perte de popularité de UNIX. De nos jours UNIX désigne surtout une famille de systèmes d'exploitation plutôt qu'un OS à part entière. On note donc deux familles de systèmes d'exploitation : Windows et UNIX. Parmi les OS UNIX qui ont résisté à la popularité de Windows, on peut citer BSD, GNU/Linux, iOS, Android et MacOS. Dans ce cours nous étudierons les principes de Unix, dans sa version GNU/Linux.

Unix

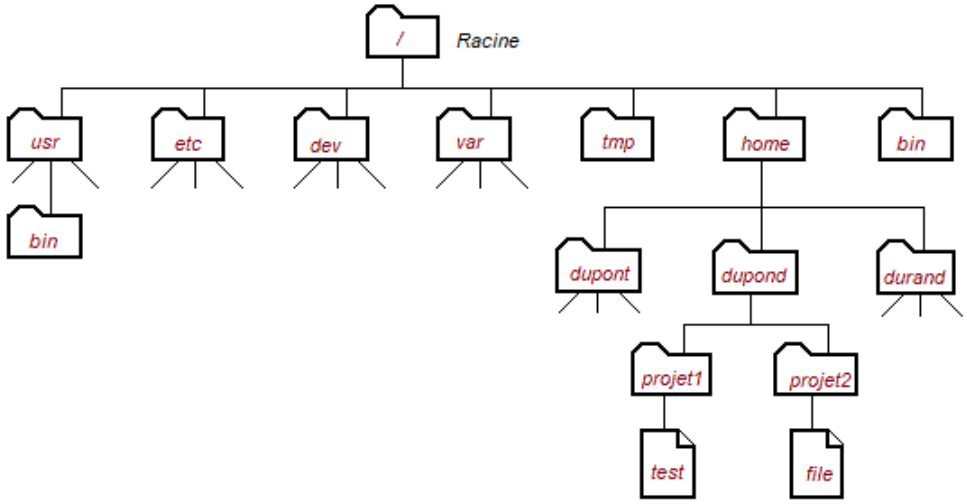
UNIX est une famille de systèmes d'exploitation multitâche et multi-utilisateur dérivé du Unix d'origine créé par AT&T (*American Telephone & Telegraph Company*) dans les années 1970 au centre de recherche de Bell Labs mené par Kenneth Thompson. Il repose sur un interpréteur ou superviseur (le `shell`) et de nombreux petits utilitaires, accomplissant chacun une action spécifique, commutables entre eux par un mécanisme de redirection et appelés depuis la « ligne de commande ». Une distribution Linux est une collection de logiciels pour la plupart *open-source* composée du système d'exploitation Linux (on parle aussi de noyau Linux) et d'une suite de librairies, d'utilitaires ou applications. Chaque distribution est maintenue par une communauté de développeurs permettant ainsi de produire des mises à jour régulières des différents logiciels associés à la distribution. En revanche, le noyau Linux est développé indépendamment de toute distribution. Il constitue donc un point commun entre elles. Parmi les distributions existantes, on note des distributions mères (ArchLinux, Debian, Gentoo, RedHat, Slackware, Android) qui se différencient par exemple par les outils utilitaires de gestion de paquets (aptitude pour Debian, Rpm pour RedHat, etc.). De ces distributions mères ont été développées des distributions filles qui fournissent les logiciels de leur distribution mère associée et d'autres utilitaires ou applications qui leur sont propres.



La notion de fichier sous UNIX ne se réduit pas uniquement au fichier classique : il permet également de désigner une ressource physique ou logique de l'ordinateur (appelé *device*), comme par exemple un terminal, une imprimante, un disque physique ou logique (partition du disque physique), la mémoire vive du système, un bus, une carte réseau, etc. Un fichier est donc caractérisé par son type qui définit l'ensemble des opérations que l'on peut lui appliquer. On peut classer les types de fichiers en deux grandes catégories :

- fichiers « disques » de différentes nature :
  - fichiers réguliers permettant de stocker des codes de programmes (binaires ou sources) ou bien des données classiques
  - répertoires permettant d'organiser le système de fichiers en arborescence et de désigner de manière unique un lien vers un fichier du système
  - liens (des noms de fichier)
- fichiers « spéciaux » désignant une ressource du système

Sous UNIX, les fichiers sont organisés selon une arborescence unique qui part d'une racine, désignée par le caractère `/` (et non `C:\` comme sous MS-DOS) et nommée *root directory*. Un fichier disque se désigne sans ambigüité grâce au chemin d'accès – le *path* absolu – qu'il faut parcourir depuis la racine pour l'atteindre. Le *path* absolu du fichier `test` est `/home/dupond/projet1/test`. Il est remarquable qu'il existe deux fichiers qui se nomment `bin`. Cependant, ce sont deux fichiers différents, l'un se désigne par `/bin` et l'autre par `/home/bin`.
La notation de chemin absolu peut devenir très lourde, notamment lorsque le lien à atteindre est très profond dans l'arborescence. C'est pourquoi il est possible de désigner des liens de manière relative (chemin relatif) au répertoire courant (voir plus loin cette notion). Par exemple, si le répertoire courant est `/home/dupond` alors il est possible de désigner le fichier `test` par le chemin relatif `projet1/test`.



`/usr` contient les logiciels installés avec le système (`/usr/bin` en particulier contient les exécutables).
`/etc` contient la plupart des fichiers de configuration.
`/dev` contient les fichiers spéciaux correspondant aux périphériques.
`/var` contient des données fréquemment réécrites.
`/tmp` contient les fichiers temporaires.
`/home` contient les répertoires personnels des utilisateurs.
Par exemple, l'utilisateur `dupond` a généralement pour répertoire `/home/dupond`.
`/bin` contient les commandes de base.

Le terminal

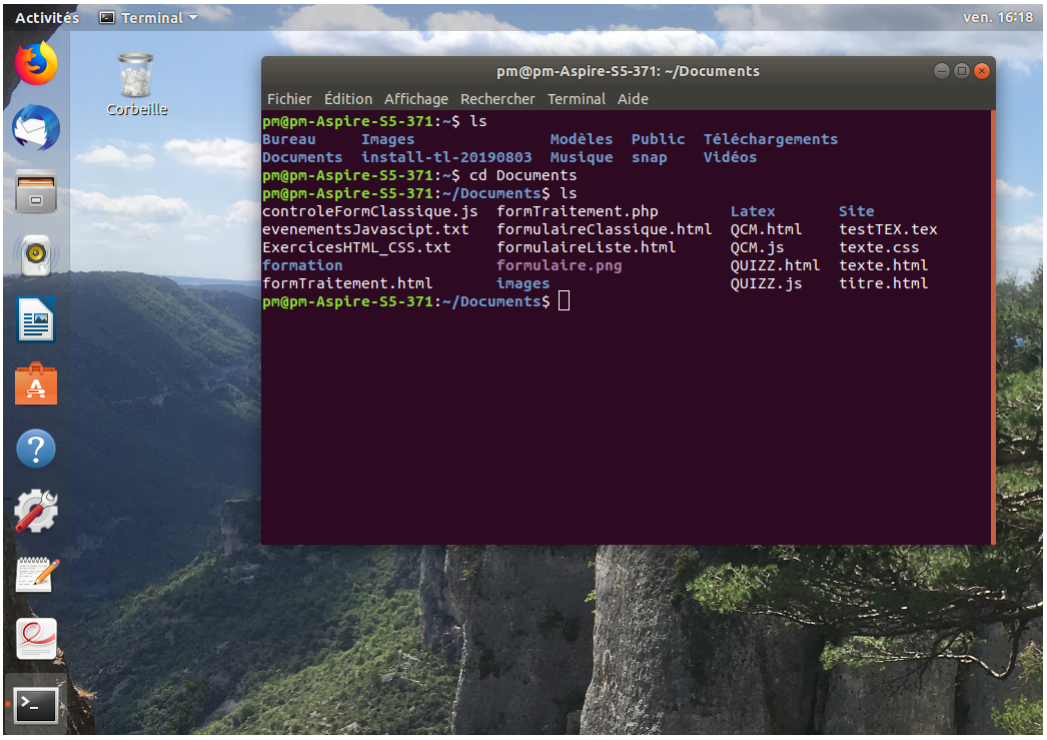
Comme la plupart des systèmes d'exploitation, Linux est un système multi-utilisateurs : il permet à plusieurs utilisateurs d'utiliser le même ordinateur tout en limitant les droits d'accès de chacun afin de garantir l'intégrité de leurs données. Pour pouvoir utiliser l'ordinateur, un utilisateur doit se connecter à la machine en ouvrant une session d'utilisation. Lors de la connexion, l'utilisateur doit renseigner :

- un *login* permettant de s'identifier de manière unique
- un moyen d'authentification permettant d'authentifier la connexion, c'est-à-dire de prouver que l'utilisateur est bien qui il prétend être. Il est possible de s'authentifier par mot de passe ou par clé

L'ouverture d'une session peut se faire de manière locale en se connectant physiquement sur la machine ou distante depuis une autre machine grâce à un réseau. Ceci permet par exemple d'administrer ou d'utiliser un ensemble de machines depuis un unique poste de travail.
Une fois connecté, le bureau de session de l'utilisateur s'affiche ainsi qu'un menu (habituellement sur le coté gauche de l'écran) avec des liens vers différentes applications installées sur la machine. Pour utiliser un terminal dans la session graphique, il faut utiliser l'application « Terminal ».

L'utilisation et l'administration d'un ordinateur peuvent se faire de manière graphique grâce à des fenêtres et à l'utilisation d'une souris (comme le font la plupart des utilisateurs) ou bien en mode

texte à travers une console ou un terminal où les commandes sont directement données au clavier. Lorsqu'on lance l'application « Terminal », le système ouvre une fenêtre et exécute un programme (ou processus) particulier appelé interpréteur de commande ou `shell` reconnaissable par la présence d'une chaîne de caractères appelée invite ou *prompt*. Ceci invite l'utilisateur à entrer des commandes de son choix. Le *prompt* par défaut se termine par un caractère comme $, % ou # précédé par une chaîne de caractères tel que le *login* de l'utilisateur et le nom de la machine séparés par un @.



Dans le terminal, l'invocation d'une commande consiste à écrire le nom de cette commande, suivi d'un ou plusieurs autres mots appelés arguments ou paramètres et de taper ensuite sur la touche Entrée. Sur la capture d'écran ci-dessus, on voit une fenêtre Terminal ouverte : par défaut, le répertoire dans lequel on se situe est le répertoire utilisateur (nommé `pm@pm-Aspire-S5-371`), le tilde `~`) symbolise ce répertoire particulier, le $ qui suit est le *prompt* (l'invite de commande). J'ai entré une 1re commande qui est `ls`. Cette commande demandant les fichiers présents, j'ai obtenu l'affichage des dix fichiers (tous des répertoires) présents dans mon dossier. J'ai alors entré une 2e commande qui est `cd Documents` qui a pour effet de déplacer le répertoire courant dans le répertoire demandé (ici par un chemin relatif). On constate l'effet de cette commande sur l'indication affichée avant le *prompt*. J'ai alors entré une 3e commande qui est encore un `ls`, mais cette fois, n'étant plus dans mon répertoire utilisateur, j'obtiens une autre liste : celle des fichiers présents dans mon dossier `Documents`.

Les commandes

Le manuel est très utile pour obtenir la notice d'utilisation d'une commande.
Taper `man <commande>` pour obtenir la notice de `<commande>` : Son nom (`NAME`), sa syntaxe (`SYNOPSIS`), un descriptif plus ou moins détaillé (`DESCRIPTION`), son auteur (`AUTHOR`), l'adresse où signaler une erreur (`REPORTING BUGS`), le copyright et des liens (`SEE ALSO`). Par exemple `man ls` nous donnera, dans son descriptif, la liste des options possibles pour cette commande.
Je note en particulier que l'option `-l` donne des détails.

Signalons tout de suite une fonctionnalité intéressante : les flèches (haut et bas) permettent de naviguer dans l'historique des commandes passées. On peut aussi effectuer un copié-collé avec la souris. Un 3e raccourci utile est la complétion automatique obtenue, après avoir taper quelques caractères, en appuyant sur la touche de tabulation, notée parfois `Tab` : s'il n'y a qu'une complétion possible, elle s'affiche, sinon un 2e appui sur `Tab` donne la liste des possibilités et il faut entrer au moins un autre caractère et `Tab` à nouveau.

Premières commandes :

- `who` permet de savoir qui est connecté sur l'ordinateur (local et distant)
- `echo` permet d'afficher du texte sur la sortie standard (le terminal)
- `expr` permet d'évaluer une expression entre deux entiers avec un opérateur (+, -, *, /, %) ou un comparateur (<, <=, >, >=, =, !=). certains caractères ayant une signification particulière, il faut les échapper par un anti-slash : pour obtenir le résultat de 3×5, taper `expr 3 \* 5`
- `ls` affiche le contenu du répertoire courant, tandis que `ls <repertoire>` affiche le contenu du répertoire spécifié. Avec l'option `-l`, on obtient un affichage long qui donne, en particulier le type de fichier et ses autorisations (voir plus loin)
- `pwd` permet de connaître le chemin du répertoire courant ; au login, mon répertoire courant est le répertoire utilisateur (`/home/pm` en ce qui me concerne)
- `cd` permet de changer de répertoire (*directory*) : on indique le chemin absolu ou relatif du répertoire vers lequel on souhaite se diriger.
  Des raccourcis utiles : `..` pour le répertoire parent, `.` pour le répertoire courant, `-` pour le répertoire que l'on vient de quitter, `~` pour le répertoire utilisateur et rien pour le répertoire utilisateur (mon répertoire). Ainsi, pour aller dans mon dossier `Documents`, je peux taper `cd ~/Documents` ou seulement `cd Documents`.

Les différents fichiers (tout type confondu) sont associés à un utilisateur propriétaire. Afin d'éviter qu'un utilisateur n'accède à des fichiers dont il n'est pas propriétaire et sans autorisation, le système permet de définir des droits d'accès pour chaque fichier (autorisations).

DÉFINITION 5.3 (AUTORISATIONS) On peut effectuer trois types d'opération sur un fichier : la lecture (`r` comme *read*), l'écriture (`w` comme *write*) et l'exécution (`x` comme *execute*).
Un utilisateur donné est soit le propriétaire du fichier (`u` comme *user*), soit membre du groupe auquel le fichier appartient (`g` comme *group*), soit un autre utilisateur (`o` comme *other*).
Ainsi, pour un fichier donné, il existe 9 droits à positionner qui sont représentés par trois triplets : chaque triplet représentent dans l'ordre, les droits (`r`, `w`, `x`) du propriétaire, du groupe et des autres utilisateurs.

Remarques :

- Si le fichier est un répertoire : `r` autorise de lister les fichiers (commande `ls`), `w` autorise qu'on y crée ou supprime des fichiers et `x` qu'on puisse le prendre comme répertoire courant ou bien le traverser pour accéder à sa sous-arborescence.
- Pour afficher les droits des fichiers du répertoire courant, on peut taper `ls -l`. Si j'obtiens `drwxr-xr-x ...` pour la ligne correspondant à un fichier, cela signifie qu'il s'agit d'un dossier (`d` comme dossier, `-` pour un fichier et `l` pour un lien) dont le droit en écriture est restreint au seul propriétaire, les autres droits étant ouverts à tous.
- Seul le propriétaire d'un fichier et le super-utilisateur (`root`) peuvent changer les droits d'accès de ce fichier et seul le super-utilisateur peut créer ou modifier les groupes. Pour connaître les groupes auxquels on appartient, utiliser la commande `groups`. Lors de la création d'un fichier par un utilisateur, les droits sont fixés par défaut à `rwxr-xr-x`, mais on peut changer cette valeur par défaut en utilisant la commande `umask` suivie des droits omis au format numérique (voir ci-dessous).

Pour changer les autorisations sur un fichier dont on est propriétaire, utiliser la commande `chmod` (pour *change mode*) suivie d'un argument informatif et du nom de fichier. L'argument informatif est constitué par

- le ou les triplets concerné(s) : `u` (propriétaire), `g` (groupe), `o` (autres) ou, tous à la fois, `a` (*all*)
- si il s'agit d'un ajout de droits : caractère `+` ; s'il s'agit d'un retrait : caractère `-`
- les droits que l'on veut modifier : `r`, `w` ou `x`

Exemple : si je tape `chmod g+rx test`, je donne les droits (`+`) sur le fichier `test` en lecture (`r`) et en exécution (`x`) au groupe (`g`).

On peut modifier les 9 droits avec le format numérique : chaque triplet est exprimé en octal (voir le chapitre 1). Pour `chmod` une autorisation est représentée par le bit 1, une restriction de droit par 0. La combinaison par défaut `rwxr-xr-x`, par exemple, est notée 755. Avec `umask`, c'est le contraire, puisqu'on fixe des droits restreints par défaut (`umask 022` enlève le droit `w` au groupe et aux autres).

Commandes concernant les fichiers :

- `touch <fichier>` crée un fichier vide de nom `<fichier>` dans le répertoire courant
- `mkdir <dossier>` (pour *make directory*) crée un dossier vide de nom `<dossier>` dans le répertoire courant
- `mv <source> <cible>` (pour *move*) déplace le fichier `<source>` vers `<cible>` si c'est un dossier et le renomme sinon. Attention : s'il existe déjà, cette commande écrase le fichier `<cible>` ! Pour éviter cela, utiliser l'option `-i` (pour *interactive*) qui demande une confirmation
- `cp <source> <cible>` (pour *copy*) copie le fichier `<source>` vers `<cible>` ; contrairement à `mv`, cette commande ne supprime pas le fichier d'origine. Pour copier un dossier, utiliser l'option `-R`
- `rm <fichier>` (pour *remove*) supprime le fichier indiqué. Utiliser l'option `-i` pour demander une confirmation et l'option `-R` pour supprimer un dossier. Attention : il n'y a pas de retour en arrière possible, ni de poubelle ! Pour ne supprimer un dossier que s'il est vide, utiliser `rmdir <dossier>`
- `less <fichier>` lit le fichier indiqué si c'est un fichier texte (pour sortir `q`). Un fichier texte (les extensions habituelles : txt, html, js, etc.) est lisible directement par la machine, contrairement à un fichier binaire (formaté, par exemple un fichier d'extension doc, jpg ou pdf). De véritables éditeurs de texte sont disponibles en ligne de commande (comme `nano`, `emacs` ou `vim`) et apportent chacun des fonctionnalités particulières, tandis que `gedit` ouvre l'éditeur de texte dans une fenêtre séparée (équivalent du bloc-note de Windows) (pour sortir `Ctrl+maj+Q`).

Le type de fichier nommé « lien » est l'équivalent, en mode graphique, d'un raccourci. Créer un lien symbolique avec la commande `ln -s <chemin> <lien>` pour remplacer le chemin, absolu ou relatif, `<chemin>` par `<lien>`. Sans l'option `-s`, le lien créé est un lien physique. En réalité le nom d'un fichier est un premier lien physique ; en créer un 2^{e} implique qu'un même fichier possède deux noms, et supprimer un tel fichier (avec `rm`) ne supprime que le lien physique, pas le fichier qui subsiste grâce à l'autre lien.

Certaines commandes peuvent être lancées sur plusieurs fichiers à la fois.
Par exemple `mv image1.jpg image2.jpg Photos` déplace les fichiers `image1.jpg` et `image2.jpg` vers le dossier `Photos` s'il existe. Cette possibilité est enrichie par l'utilisation d'un mécanisme appelé *glob* qui manipule des motifs représentant des noms de fichiers :

- le caractère `*` représente n'importe quelle suite de caractères (sauf `/`)
- le caractère `?` représente n'importe quel caractère (sauf `/`)
- le *glob* `[i3G]` représente `i`, `3` ou `G`
- le *glob* `[0-3]` représente tous les chiffres de `0` à `3`
- le *glob* `[c-m]` représente toutes les lettres de `c` à `m`

Ainsi `mv im* Photos` déplace d'un coup toutes les images nommées `imagex.jpg` (`x` prenant n'importe quelle valeur) vers le dossier `Photos` (et d'autres fichiers, comme `impots.txt` s'il existe). De même, `cp *[0-9][0-9]*.txt Documents` copie tous les fichiers d'extension `txt` contenant 2 chiffres juxtaposés vers le dossier `Documents`.

Il y a de très nombreuses autres commandes. Par exemple, `file` donne, entre autres, le type de fichier ; `locate`, `find` et `grep` sont utiles pour retrouver un fichier existant ; `gzip` et `gunzip` pour compresser ou décompresser un fichier ; `tar` pour regrouper des fichiers en archive ou extraire les fichiers d'une archive ; etc. Il est aussi possible d'ajouter de nouvelles commandes et de nouveaux logiciels (absents de la distribution choisie) mais nous renvoyons le lecteur à des manuels/sites [4] plus complets.

4. Pour la distribution Ubuntu commencer par la page `https://doc.ubuntu-fr.org/`

Les scripts de commandes

On peut automatiser une séquence de commandes en écrivant un script de commandes (*shell-script*). Un tel script commence toujours par la ligne `#!/bin/bash`, appelée *shebang*, qui indique l'interpréteur devant être utilisé (ici, `bash`).
En *bash*, une ligne de commentaires (ou une fin de ligne) commence par `#`.
Les variables sont déclarées et affectées d'un coup, avec le signe `=`. Le contenu d'une variable est toujours considéré comme étant une chaîne de caractères. Pour lire le contenu d'une variable, il faut la préfixer d'un `$`.
Voici un premier *shell-script* :

```
#!/bin/bash
texte="Bonjour !"
echo $texte
```

Ce script produira l'affichage suivant sur le terminal : Bonjour !
Pour écrire le fichier de ce programme en mode terminal, on peut commencer par créer le fichier avec la commande `touch script.sh` (l'extension `sh` n'est pas indispensable, mais elle permet de se rappeler le type du fichier). Ensuite, il faut ajouter le droit d'exécution `x` à ce fichier, en lançant la commande `chmod u+x script.sh`. Le fichier, maintenant exécutable, ne sera pas reconnu si le dossier courant n'a pas été ajouté dans le `PATH`. La commande `export PATH=$PATH:.` permet d'ajouter le répertoire courant dans le `PATH`[5].
L'inconvénient de cette méthode est que la commande `export` n'est active que sur la durée d'une cession terminal. On peut créer un dossier dans le répertoire utilisateur, nommé par exemple `bin` (ou `sh`) qui contiendra tous les scripts exécutables de l'utilisateur sans avoir recours à la commande `export` : ouvrir le fichier nommé `/.bashrc` et lui ajouter la ligne suivante :

```
if [ -d $HOME/bin ]; then export PATH=$PATH:$HOME/bin fi
```

Le fichier `/.bashrc` est le fichier de configuration de l'utilisateur, c'est-à-dire un script exécuté au démarrage du `shell`. La ligne ajoutée à ce fichier transforme tous les fichiers exécutables du dossier `/bin` en commandes reconnues par le système.

Trois types de guillemets (*quote*) sont utilisables en *bash*, avec des significations différentes :

- les simples *quote* délimitent une chaîne de caractères non interprétée : `echo 'mon texte est $texte'` produira l'affichage de `mon texte est $texte`
- les doubles *quote* délimitent une chaîne de caractères où les noms de variables sont interprétés : `echo "mon texte est $texte"` produira l'affichage de `mon texte est Bonjour !`.
- les *back-quote* (accent grave) délimitent une commande à exécuter. Les noms de variables et les commandes y sont interprétées et on peut stocker la sortie standard de cette commande dans une variable. La ligne `var1=1;var2=‘expr 5 + $var1‘` conduit à avoir 6 dans `var2`. On peut remplacer `‘...‘` par `$(...)`. Ainsi, l'expression `var2=‘expr $x + $y‘` peut être remplacée par `$(expr $x + $y)`, ou même `$(($x + $y))` ou même encore `$((x+y))`.

Syntaxe d'une instruction conditionnelle, un *if* :

```
if <commande>; then
<instructions>
fi
```

Une commande en *bash* renvoie 0 si elle est vraie.
Un test est une commande particulière qui s'écrit `[ <test> ]` (attention : espaces nécessaires !)
On peut effectuer des comparaisons entre entiers. Par exemple, avec la syntaxe `[ n1 -eq n2 ]`, on teste l'égalité (remplacer `-eq` par `-ne`, `-ne`, `-lt`, `-le`, `-gt` et `-ge` pour tester, respectivement, l'inégalité, l'infériorité stricte ou large, la supériorité stricte ou large).
On peut aussi tester la valeur de deux opérandes quelconques, avec les symboles `=` et `\!=`
Les opérandes contenant une variable doivent être interprétées :
par exemple écrire `[ "$x" -eq "$y" ]` pour tester l'égalité numérique de `$x` et `$y`
Il existe aussi des tests portants sur les fichiers : `[ -e f ]` teste si le fichier `f` existe, `[ -d f ]` teste si c'est un dossier, `[ -w f ]` teste si le fichier existe et qu'on a le droit de le modifier, etc.

5. Le `PATH` est une variable d'environnement qui contient la liste des dossiers dans lesquels chercher les commandes

Syntaxe d'un boucle *While* :

```
while <commande>; do
<instructions>
done
```

Syntaxe d'une boucle *For* :

```
for ((i=0 ; 10 - $i ; i++)) do
<instructions>
done
```

Il y a d'autres syntaxes possibles pour ce type de boucle.
Consulter un guide de programmation *bash* ou un cours spécialisé [6] pour davantage de détails.

Pour donner des arguments en entrée d'un script de commandes, il faut les écrire à la suite du nom de la commande, précédés d'un espace. Dans le programme, pour accéder à des arguments, il suffit de les appeler avec la syntaxe `$n` où `n` est le numéro d'ordre de l'argument (`$1` est le 1er argument, `$2` est le 2e, etc.). D'autres variables prédéfinies sont utilisables : `$@` est une liste contenant les arguments, `$#` contient le nombre d'arguments. Supposons qu'on donne au programme suivant le nom de « ecrire » :

```
#!/bin/bash
echo $1
```

Après l'avoir rendu exécutable, la commande `ecrire Bonjour !` va écrire `Bonjour` sur le terminal. Le mot est considéré comme le 1er argument et le point d'exclamation comme 2e argument ; il n'est donc pas écrit puisque le programme n'écrit que le 1er.

Lors de l'exécution d'un script, trois flux de données sont automatiquement créés : l'entrée standard (le clavier), la sortie standard et la sortie erreur, toutes deux dirigées par défaut sur le terminal.
Il est possible de rediriger ces flux vers d'autres périphériques :

- l'entrée standard est redirigée vers le fichier `f` (les périphériques sont également des fichiers) avec la syntaxe `commande [arguments] < f`
- la sortie standard est redirigée vers le fichier `f` avec la syntaxe `commande [arguments] > f` ; si `f` est un fichier régulier, cela a pour conséquence d'écraser son contenu.
  Pour ajouter la sortie en fin d'un fichier régulier, remplacer `>` par `>>`.
- la redirection de la sortie erreur s'effectue de même, mais avec les symboles `2>` et `2>>`.

Exemples de redirections :
La commande `ls Musique > maMusique.txt` crée un fichier texte contient la liste des fichiers présents dans le répertoire `Musique` (au lieu d'afficher cette liste dans le terminal).
La commande `ls Chansons >> maMusique.txt` ajoute, à la fin de `maMusique.txt`, la liste des fichiers du répertoire `Musique`.
Pour créer le fichier `livre.txt` à partie de fichiers `chapitreN.txt` où `N` est un nombre, on peut lancer `cat chapitre*.txt > livre.txt` (`cat` est une commande qui affiche les fichiers fournis en argument sur la sortie standard).
La commande `mail -s 'test' ph.moutou@free.fr < message.txt` permet d'envoyer un mail à l'adresse `ph.moutou@free.fr` dont le sujet est `test`, le contenu du mail est écrit dans le fichier `message.txt`.

Pour enchaîner deux commandes, on peut faire usage du `;` (on l'a utilisé avec le `if ...; then`).
Cette construction lance la 2e commande même si la 1re a échoué.
Pour éviter cela, utiliser `&&` qui ne lance la 2e commande que si la 1re s'est correctement déroulée.
Á l'opposé, on peut utiliser `||` qui ne lance la 2e commande que si la 1re a échoué.
Il est possible d'envoyer la sortie de la 1re commande sur l'entrée de la 2e en les séparant par `|` : on crée ce qui est appelé un *pipe* (tube). L'envoi du mail précédent peut être réalisé sans redirection ainsi `cat message.txt | mail -s 'test' ph.moutou@free.fr`
Pour rechercher les fichiers `pdf` dans toute l'arborescence et lire la sortie dans `less` (qui possède une fonction de recherche d'un texte, en tapant `\texte`), il suffit de lancer la commande `locate *.pdf | less`.

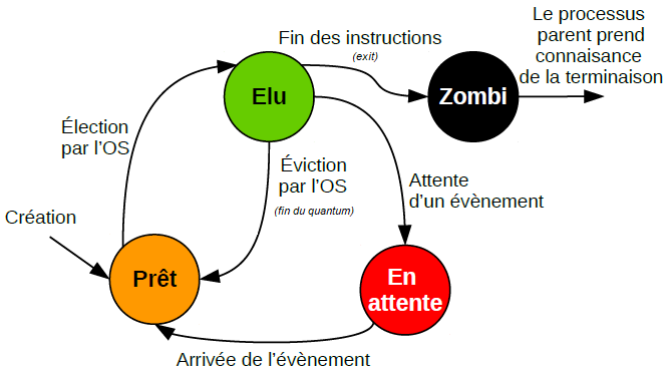
6. Suivre le cours de `https://openclassrooms.com/fr/courses/43538-reprenez-le-controle-a-laide-de-linux`

Les processus

Un processus, c'est un programme en cours d'exécution. Cela englobe les instructions du programme et aussi son contexte d'exécution :

- l'espace mémoire RAM manipulé pour pouvoir fonctionner. Cette espace mémoire regroupe aussi bien les données utilisées que les instructions du programme nécessaires à son exécution.
- un ensemble d'informations maintenues par le système d'exploitation comme : le PID (identifiant attribué par le système au moment de la création du processus), le PPID (identifiant du processus parent), le nom du propriétaire (identifiant de l'utilisateur qui a lancé le processus), le groupe du propriétaire, l'identifiant du terminal de rattachement (si le processus a été lancé avec un terminal), sa priorité, son répertoire courant, son état, ses signaux reçus, ses ouvertures de fichier, etc.

Chaque programme implique souvent plusieurs processus, et pourtant un processeur est capable d'exécuter plusieurs programmes en même temps. Afin de permettre à chaque processus d'évoluer dans son exécution, le système d'exploitation doit garantir l'accessibilité du processeur en un temps fini.
Le mécanisme de commutation : au bout d'une certaine période de temps (quantum), le système d'exploitation est chargé d'interrompre le processus en cours d'exécution, de choisir un autre processus éligible (défini par la politique d'ordonnancement qui tient compte de priorités) et de poursuivre l'exécution du processus nouvellement élu. Le processus évincé du processeur sera réélu ultérieurement par le système d'exploitation.



L'état prêt : le processus est en attente du processeur. C'est l'état de départ de tout processus et l'état incontournable pour obtenir le processeur. Le processus sort de cet état uniquement pour passer à l'état élu quand le système d'exploitation le choisi lors d'une commutation.
L'état élu : le processus utilise le processeur. Il sort de cet état pour une de ces raisons :

- le système d'exploitation décide de l'évincer car il est arrivé en fin de quantum (repasse à l'état prêt)
- le processus doit attendre un évènement particulier (passe à l'état en attente)
- le processus n'a plus d'instruction à exécuter (passe à l'état zombi)

L'état en attente : le processus attend un évènement particulier autre que l'obtention du processeur (par exemple le contenu d'un fichier suite à une requête d'entrée/sortie sur un périphérique). On ne peut sortir de cet état que lorsque l'évènement attendu est arrivé.
L'état zombi : le processus a terminé son exécution et attend que son parent prenne connaissance de sa terminaison.

Il est possible de consulter l'état des processus avec la commande `ps aux` (1re lettre du champ de colonne `STAT`). Du point de vue utilisateur, les états élu et prêt sont confondus et portent tous les deux la lettre `R` (*Running/Runnable*), l'état en attente est subdivisé en deux états (les lettres `S` et `T`) et l'état zombi est représenté par la lettre `Z`.
Le système d'exploitation apparait comme un gestionnaire d'interruptions : à ce titre, il traite des évènements d'origine matérielle (une lecture d'horloge qui permet de détecter les fins de quantum, une notification de fin d'entrée/sortie) ou logicielle (un appel système : fonctionnalité offerte aux applications comme créer/tuer un processus ou créer/lire/écrire un fichier).

La création d'un nouveau processus se fait par le système à la demande de l'utilisateur et se fait toujours à partir d'un processus existant appelé processus parent. D'un point de vue du *shell*, un nouveau processus est créé automatiquement lorsque l'on entre une commande dans le terminal (excepté pour quelques commandes internes comme par exemple `echo` ou `cd`). Une fois la commande lancée, le processus nouvellement créé exécute le code de la commande. Son processus parent est le processus de l'interpréteur de commande (le *shell*). À sa terminaison, un processus possède une valeur appelée code de retour à laquelle son processus parent peut accéder (code de retour nul si pas d'erreur).

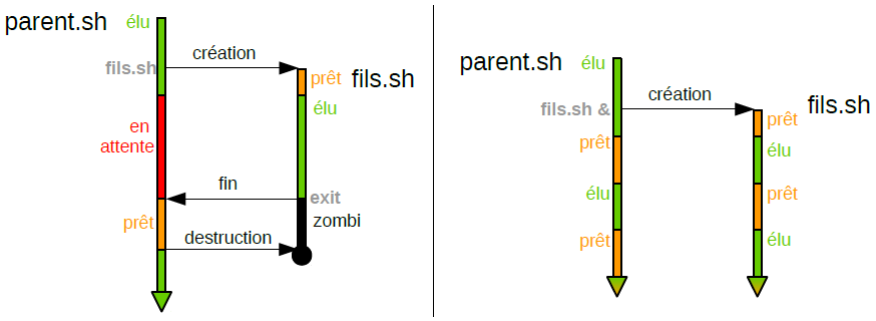
Les variables d'environnement sont un ensemble de variables dont les valeurs sont transmises de processus père en processus fils. Ainsi toute variable présente dans l'environnement du processus parent sera copiée dans l'environnement de chaque processus créé. Pour déclarer une variable dans l'environnement il faut utiliser la commande `export <variable>`.
Exemple : considérons les deux scripts `pere.sh` et `fils.sh` dont les codes sont (j'ai omis le *shebang* et compacté les scripts à l'aide de `;`) :

- pere.sh : `var=bonjour ; ./fils.sh ; export var ; ./fils.sh ; echo $var`
- fils.sh : `echo "var = $var" ; var="au revoir"`

Exécuter `pere.sh` conduit à l'affichage :

```
var=
var=bonjour
bonjour
```

On constate que le contenu de la variable d'environnement `var` n'est transmise au processus fils qu'après la commande `export var`.

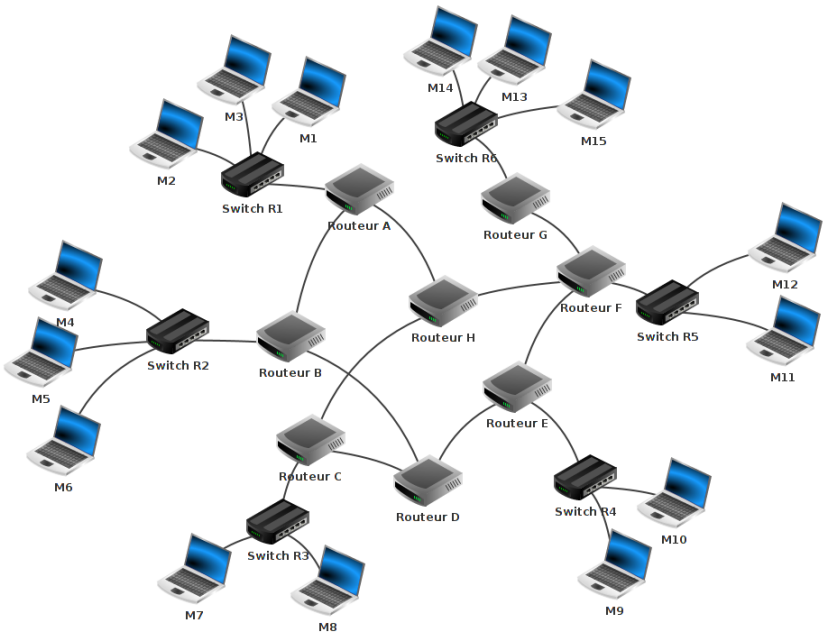


Lors du lancement d'une commande, un processus est créé implicitement par l'interpréteur si la commande n'est ni un mot clé (`if`, `for`, affectation de variable,...) ni une commande interne (`echo`, `cd`, `export`, `pwd`, `test`,...). Si un processus est créé de cette façon, le processus parent (celui qui a lancé la commande, terminal ou script) se met implicitement en attente de terminaison du processus fils. Ce mode de fonctionnement est illustré à gauche.
Il est possible de lancer une commande engendrant un processus sans attendre la fin de celui-ci en ajoutant le caractère & après la commande. Le processus père ne se bloque pas dans l'état « en attente », ce qui permet d'avoir deux processus qui s'exécutent de manière concurrente.

3. Réseaux

Un réseau est un ensemble d'équipements interconnectés pour réaliser une communication à distance. S'il a existé plusieurs familles de réseau (téléphoniques, informatiques, audiovisuels) utilisant des infrastructures séparées et des protocoles différents, ils ont tendance à converger aujourd'hui par l'utilisation d'un protocole commun : IP (*Internet Protocol*). Pour cette raison, nous ne présenterons ici que l'architecture TCP/IP (TCP pour *Transmission Control Protocol*).
L'architecture réseau définit les éléments (types de nœuds) d'un réseau : dans le réseau IP, il y a des hôtes et des routeurs. Les protocoles de communication entre les nœuds définissent le format des messages échangés entre les nœuds. Par exemple, dans le réseau TCP/IP, le protocole IP définit le format des paquets IP qui sont utilisés pour échanger des données entre un hôte et un routeur ou entre deux routeurs. Le protocole TCP, quant-à lui, définit le format des segments à échanger entre

deux hôtes. Les algorithmes utilisés par un protocole pour traiter les messages reçus dans les nœuds influent également sur les performances d'un réseau. Dans le réseau TCP/IP, le protocole UDP n'utilise pas l'algorithme de contrôle de congestion utilisé par TCP : en cas de congestion, un émetteur TCP réduit le débit de transmission tandis que l'émetteur UDP continue à envoyer les données.



Les réseaux de transmission de données par paquets ont une architecture en couches : la tâche de transmission des données est décomposée en plusieurs sous-tâches indépendantes (fonctionnalités) qui constituent les couches. Les données reçues ou envoyées sont traitées couche par couche, le chainage des fonctionnalités suivant un ordre précis. Dans une même couche, les nœuds intercommuniquent par un protocole qui définit le format des messages à échanger entre les nœuds ainsi que les algorithmes pour le traitement des messages. Les architectures les plus connues sont OSI (*Open System Interconnection*) qui comprend 7 couches et TCP/IP qui en comprend 4. Le modèle OSI, défini par l'*International Standard Organization* (ISO) est une référence qui n'a pas été implémenté : les trois couches de base se trouvent dans tous les nœuds du réseau tandis que les quatre couches hautes sont seulement aux extrémités qui se soucient de la communication de-bout-en-bout :

1. La couche physique transmet des éléments binaires sur un support physique en utilisant des signaux.

2. La couche liaison est responsable de la transmission des trames entre deux nœuds ayant un support physique direct.

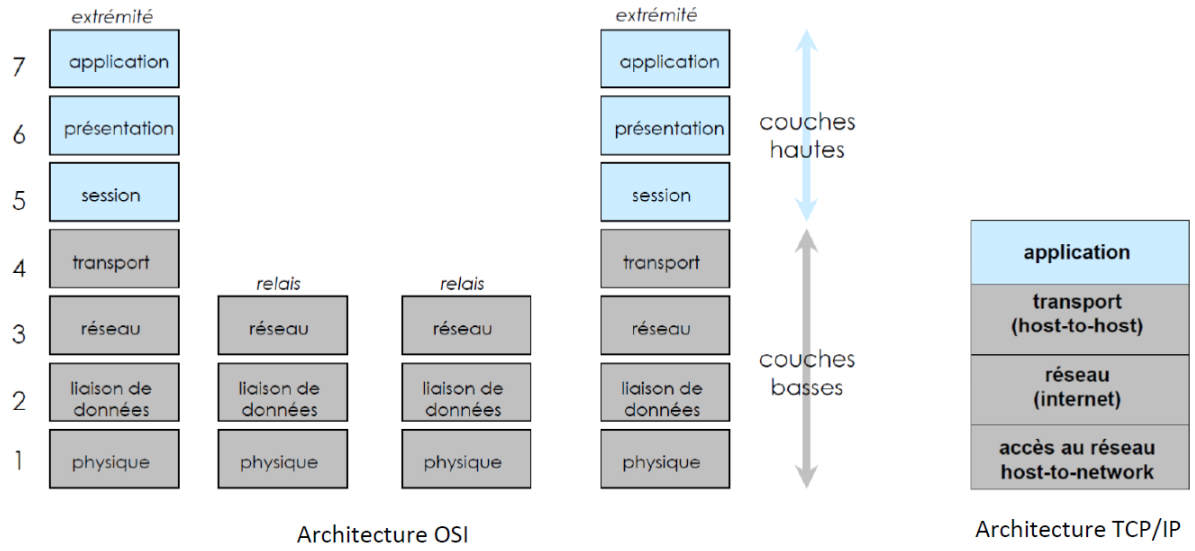
3. La couche réseau transmet des paquets d'une source à une destination en passant par plusieurs nœuds intermédiaires.

4. La couche transport s'occupe de la transmission de-bout-en-bout entre deux machines utilisateurs.

5. La couche session gère les sessions de communications de-bout-en-bout.

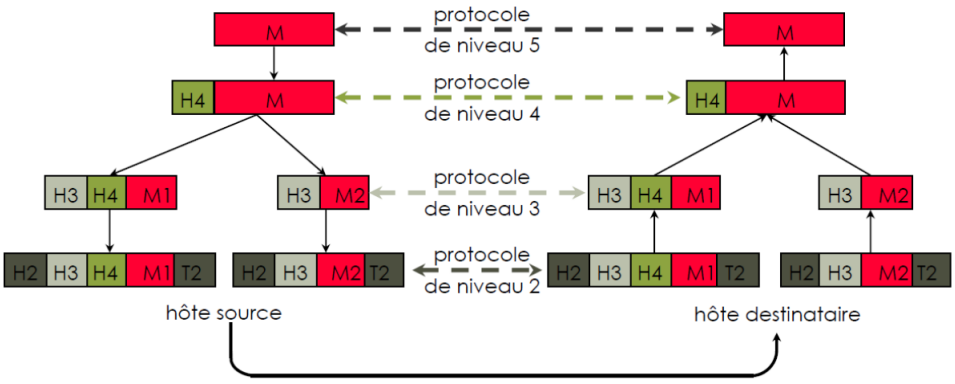
6. La couche présentation encode les données utilisateurs sous certains formats (MP3, ASCII,...)

7. La couche application fait l'interface avec l'utilisateur.



Le modèle TCP/IP, défini par l'*Internet Engineering Task Force* (IETF) n'a que quatre couches :

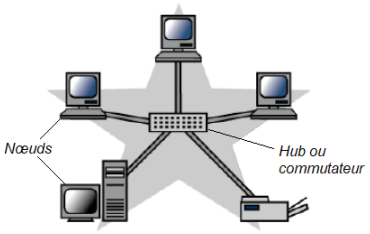
1. La couche physique permet l'accès au réseau. Cette couche correspond aux couches 1 et 2 du modèle OSI (transmission des trames entre nœuds et des bits sur le support physique).
2. La couche réseau est identique à celle du modèle OSI.
3. La couche transport est identique à celle du modèle OSI.
4. La couche application correspond à l'ensemble des couches 5 à 7 du modèle OSI.



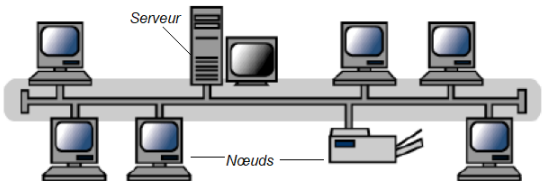
Si nous regardons à l'intérieur d'un nœud, l'interaction entre les couches adjacentes se fait par l'encapsulation de données à l'émetteur et la décapsulation des données au récepteur. L'encapsulation des données dans une couche prend les données reçues d'une couche supérieure et leur ajoute un en-tête spécifique (des informations de contrôle) puis passe ce paquet au niveau inférieur. Pour la décapsulation des données d'une couche, le paquet reçu de la couche inférieure est analysé en fonction de son en-tête puis, celle-ci étant omise, le paquet est transmis au niveau supérieur.

a. Réseaux Ethernet

Ethernet est la technologie prédominante pour l'interconnexion des ordinateurs dans un réseau local. Dans l'Ethernet partagé, les stations sont connectées directement à un support partagé. Dans les années 80 et 90, deux topologies les plus déployées de l'Ethernet partagé est la topologie en étoile (utilisant un *hub*) et les paires torsadées et la topologie en *bus* utilisant un câble coaxial. Aujourd'hui, nous n'utilisons plus les câbles coaxiaux et donc la topologie en *bus* disparaît. La topologie en étoile subsiste mais les *hub* sont souvent remplacés par des commutateurs. Dans le cas, on ne parle plus d'Ethernet partagé mais d'Ethernet commuté.



Réseau Ethernet en étoile

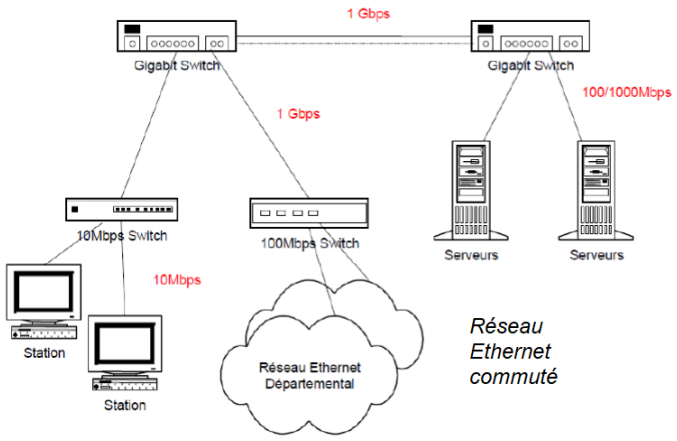


Réseau Ethernet en bus

trame Ethernet

	Préambule	SFD	Adresse MAC dest.	Adresse MAC src.	Longueur /PT	Données	CRC
(octets)	7	1	6	6	2	46 – 1500	4

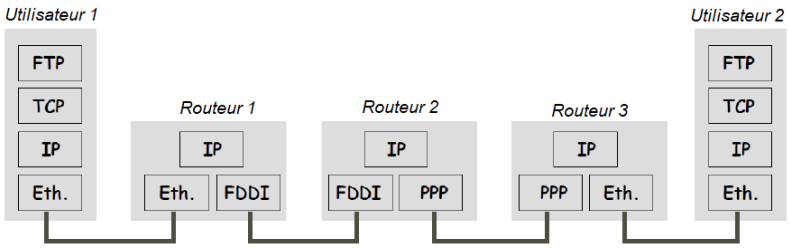
Dans l'Ethernet partagé, quand une machine veut envoyer des données à une autre machine du réseau, elle envoie la trame sur le support commun. Toutes les stations vont recevoir la même trame et en analyser l'en-tête. Le destinataire est le seul à retrouver son adresse dans l'en-tête (champ « Adresse MAC destination ») et en garde une copie, tandis que les autres stations ignorent cette trame reçue.
Dans les réseaux à support partagé, il est possible d'avoir les collisions : si deux stations émettent en même temps, les signaux se superposent. Les récepteurs n'arrivent plus à interpréter les trames corrompues par collision. L'algorithme *Carrier Sense Medium Access/Collision Detection* (CSMA/CD) d'accès au support MAC (*Medium Access Control*) a deux phases : CSMA et CD. La partie CSMA veille à ce qu'une station ayant une trame à émettre écoute le support avant d'émettre sa trame. Si le support est libre, la station peut envoyer la trame, sinon elle doit attendre. La partie CD veille à ce que, durant la transmission, le support soit écouté pour détecter d'éventuelles collisions. Si aucune collision n'est détectée jusqu'à l'envoi du dernier bit, la trame a été envoyée avec succès. En cas de collision, la transmission s'arrête et, après un temps d'attente aléatoire (l'algorithme de Backoff), est retransmise. Ce temps d'attente tient compte de la charge du réseau (estimée d'après le nombre de collisions consécutives). Pour éviter la transmission de trames corrompues par collision mais non détectées, les trames doivent avoir une longueur minimales de 64 octets et la durée d'envoi de la trame doit dépasser la durée maximale d'un aller-retour dans ce réseau.
Un *hub* est un équipement au niveau 1 (niveau physique) qui diffuse le signal reçu sur un port d'entrée vers tous les ports de sortie. Un commutateur est un équipement au niveau 2 (niveau liaison) qui comprend le format de la trame Ethernet et peut lire les adresses MAC dans l'en-tête de la trame. Il apprend automatiquement quelle station est connectée à quel port en analysant le champ d'adresse `MAC source` des trames reçues puis met à jour la table de commutation. En se basant sur sa table de commutation, un commutateur peut choisir le port de sortie pour délivrer la trame à la bonne station destinataire.



Avec les commutateurs Ethernet, chaque port du commutateur est branché à une seule station. Les câbles étant aussi en *full-duplex*, il n'y a plus de collision dans un réseau Ethernet commuté et plus de contraintes sur la distance maximale du réseau. Un réseau Ethernet commuté peut être très grande et ses performances sont bien meilleures que celles d'un réseau Ethernet partagé.

b. Protocole Internet

On l'a dit, l'architecture IP définit deux types de nœuds : les hôtes et les routeurs. Un hôte est une machine utilisateur dans laquelle se trouvent les applications réseaux (navigateur web, messagerie, transfert de fichier) tandis qu'un routeur est un équipement d'interconnexion, conçu pour interconnecter deux ou plusieurs réseaux IP entre eux, potentiellement hétérogènes : les interfaces des routeurs peuvent utiliser des protocoles différents. Ainsi, un routeur IP peut relier un réseau Ethernet à un réseau FDDI ou PPP (pour une liaison louée).



Cette illustration montre l'architecture protocolaire de la communication entre deux hôtes (utilisateurs 1 et 2) s'envoyant un fichier (protocole applicatif : FTP). Le protocole FTP utilise le protocole TCP pour le transport ; les segments TCP passent au niveau IP pour former des paquets IP. Dans l'en-tête du paquet IP, l'adresse IP source, l'adresse IP destinataire et d'autres informations de contrôle sont insérées. Ensuite, le paquet IP est passé au niveau physique (le niveau MAC) pour être traité par le protocole Ethernet. Pour former la trame Ethernet, un en-tête Ethernet avec l'adresse MAC de l'utilisateur 1 (adresse MAC source) et de l'interface Ethernet du routeur 1 (adresse MAC destination) est ajouté. C'est cette trame Ethernet qui est envoyée par l'utilisateur 1 sur la liaison physique, à destination de l'utilisateur 2, via les routeurs 1, 2 et 3.
Le rôle principal du niveau IP est d'acheminer les paquets IP d'un nœud source vers un nœud destination en utilisant la table de routage dans chaque nœud. Pour construire leur table de routage, les routeurs IP échangent des messages de routage entre eux pour découvrir les chemins : chaque interface réseau doit être identifiée d'une manière unique par une adresse IP. Observons l'adresse IP d'une machine utilisateur (un hôte) : 132.227.98.5.
Un PC fixe a souvent une seule interface de type Ethernet. Chaque carte Ethernet a une adresse physique (encore appelée adresse MAC ou adresse Ethernet) codée sur 6 octets qui sert à identifier la carte réseau mais ne donne aucune information de localisation de la machine dans Internet. Ce n'est donc pas une véritable adresse, juste un identifiant de la carte Ethernet. Chaque interface réseau doit recevoir une adresse : cette adressage IP est réalisé manuellement par l'administrateur, ou dynamiquement par un protocole comme DHCP. La taille de cette adresse est de 32 bits (4 octets) écrite sous la forme décimale pointée : chaque octet est écrit sous la forme d'un nombre compris entre 0 et 255, espacé du suivant par un point. Pour éviter que l'utilisateur ne doive retenir l'adresse d'une machine sous cette forme, chaque machine possède également un nom symbolique : le nom DNS (par exemple Darwin, Hera, www.google.com, ...) choisi par l'administrateur. La correspondance entre le nom symbolique et l'adresse IP est maintenue par le système DNS (*Domaine Name System*) tandis que la correspondance entre l'adresse IP et l'adresse MAC d'une machine peut être obtenue par le protocole ARP (*Address Resolution Protocol*).

L'adresse IP est composée de deux parties : un préfixe réseau (*netid*, le localisateur) commun à tous les hôtes du même réseau IP et un suffixe machine (*hostid*, l'identificateur), qui identifie une machine du réseau.
Pour définir un plan d'adressage, un administrateur réseau identifie le nombre de machines que son réseau contiendra et en déduit la taille du suffixe machine dont il aura besoin et ainsi que la taille du préfixe réseau. Un préfixe réseau devant être unique, cet administrateur devra émettre une demande auprès d'un organisme international pour obtenir un préfixe réseau (bloc d'adresses CIDR d'une taille adéquate). Dans certains cas, si le réseau n'est pas directement visible sur internet, l'administrateur peut lui même choisir son propre préfixe réseau et utiliser un serveur NAT.

Exemple 1 – Une entreprise a 20 ordinateurs. Pour identifier chaque machine d'une manière unique, il nous faut 5 bits pour l'identificateur de machine (*hostid*) car $25 = 32 > 20$.
L'administrateur doit demander un bloc CIDR de 27 bits car $32 - 5 = 27$. Si l'entreprise obtient l'adresse réseau 125.93.64.96 avec un préfixe réseau sur 27 bits, il faut comprendre que cette adresse est composée de la partie réseau (*netid*) de 27 bits suivie d'une partie machine (*hostid*) nulle sur 5 bits : l'adresse réseau 125.93.64.96 sous forme binaire est 01111101.01011101.01000000.01100000.
Toutes les machines de ce réseau vont avoir la même valeur pour les 27 premiers bits dans leur adresse IP (les bits en rouge), seuls les 5 derniers bits sont différents (les bits en bleu) et permettent de différencier les machines de ce réseau. La 1re machine du réseau aura l'adresse 125.93.64.97 (01111101.01011101 .01000000.01100001) et la dernière adresse possible dans ce réseau est 125.93.64.126 (01111101.01011101.01000000.01111110). En effet, les adresses ayant les bits de la partie suffixe machine mis tous à 0 et tous à 1 ne peuvent pas être attribuées à une machine. Ces adresses sont réservées pour des buts spécifiques :

- si les bits du suffixe machine sont tous nuls, il s'agit de l'adresse du réseau
- si les bits du suffixe machine sont tous égaux à 1, il s'agit de l'adresse de diffusion dans le réseau. Un paquet IP avec l'adresse de diffusion comme l'adresse destination est destiné à toutes les machines du réseau.

Pour extraire l'adresse réseau à partir d'une adresse IP, il faut connaître son masque de sous-réseau (*subnet mask*) : il indique sur combien de bits est codé le préfixe réseau. Le masque peut être représenté de deux manières :

- sous la forme d'une adresse IP comme une suite de 32 bits mais avec les bits de la partie préfixe réseau tous mis à 1 et les bits de la partie machine mis à 0. Si on considère le réseau 142.68.2.0 ayant un préfixe réseau codé sur 24 bits, il reste 8 bits pour identifier les machines du réseau. Son masque de sous-réseau sera 11111111.11111111.11111111.00000000 soit 255.255.255.0 en notation décimale pointée. L'adresse du réseau est identifiée par l'adresse du réseau 142.68.2.0 et le masque 255.255.255.0.
- en indiquant à la suite de l'adresse IP, après un caractère de séparation (/), le nombre de bits sur lequel est codé le préfixe réseau : la notation 142.68.2.0/24 indique que le nombre de bits sur lequel est codé le préfixe réseau est 24.

Une fois obtenue l'adresse réseau, l'administrateur réseau doit souvent organiser son réseau en sous-réseaux. Les sous-réseaux sont ensuite interconnectés par des routeurs installés et configurés par cet administrateur réseau. Pour définir les sous-réseaux, l'administrateur prend les premiers bits de la partie d'identificateur de machine (bits de poids fort du suffixe *hostid*) pour identifier les sous-réseaux. S'il y a 4 sous-réseaux, il lui faut prendre 2 bits (de 0 à 3 cela fait $2^2 = 4$ numéros) et pour 5 sous-réseaux, il lui faut en prendre 3. La taille du suffixe « machine » qui sert à identifier les machines dans chaque sous-réseau en sera d'autant réduite.

Exemple 2 – En regardant l'adresse IP destination dans l'en-tête IP, le routeur sait comment acheminer le paquet vers la bonne interface de sortie pour arriver au réseau destinataire : il utiliser le masque de sous-réseau pour trouver l'adresse du sous-réseau auquel appartient une adresse IP.
L'adresse IP de la machine et son masque de sous-réseau sont convertis en format binaire ; une opération ET logique sur ces deux valeurs donne l'adresse de sous-réseau en binaire
L'adresse IP 142.68.2.6 avec un masque de sous-réseau 255.255.255.0 conduit à l'adresse réseau 142.68.2.0. Le calcul détaillé est le suivant :

```
      142.68.2.6 = 10001110.01000100.00000010.00000110
&& 255.255.255.0 = 11111111.11111111.11111111.00000000
                   ----------------------------------------
                   10001110.01000100.00000010.00000000 =142.68.2.0
```

c. Protocoles TCP et UDP

En utilisant l'adresse IP destinataire qui est mise dans l'en-tête de chaque paquet IP et la table de routage dans chaque routeur, ceux-ci arrivent à destination mais il est possible qu'ils soient dans le

désordre. Il peut aussi y avoir des pertes de paquets par les routeurs en cas de congestion. Les protocoles TCP (*Transmission Control Protocol*) et UDP (*User Datagram Protocol*) du niveau transport permettent d'identifier le processus d'application auquel un paquet IP appartient.
Le protocole UDP offre un service de transport de-bout-en-bout simple. Le protocole TCP offre un service de transport complexe avec le contrôle de flux et le contrôle de congestion. C'est aux développeurs des applications réseaux de choisir quel protocole utiliser. Par exemple, le protocole du Web — HTTP – est basé sur le protocole TCP tandis que le DNS utilise UDP. Une application de transfert de fichier comme FTP utilise aussi TCP tandis que le protocole RTP, utilisé par les applications multimédia est basé sur UDP. Pour utiliser les protocoles de transport dans une application réseau, les développeurs utilisent l'interface de connexion appelée *socket* (cela est possible dans la plupart des langages de programmation évolués) TCP ou UDP.

UDP

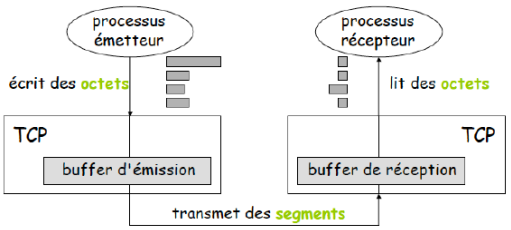
Pour indiquer le processus applicatif auquel le paquet UDP appartient, le protocole utilise un numéro de port (un point d'accès à une application au niveau applicatif). Ainsi, on peut identifier une communication UDP de bout-en-bout entre deux processus applicatifs à l'aide des cinq valeurs suivantes :

- valeurs de l'en-tête UDP : les numéros de port source et destination
- valeurs de l'en-tête IP : les adresse IP source et destination et le numéro du protocole (17 pour UDP)

Un numéro de port est codé sur 16 bits (il peut donc y avoir $2^{16} = 65536$ ports). Les ports de numéro inférieur à 1024 sont des numéros de port standardisés et utilisés par des applications bien référencées (*well-known ports*). Ainsi, le port par défaut associé à un serveur DNS utilise le port 53 est un exemple de port bien connu basé sur UDP. Deux autres champs dans l'en-tête UDP indiquent la longueur du message (sur 16 bits) et un code erreur (sur 16 bits).
Le protocole UDP ne permet pas de détecter les messages perdus ou le dé-séquencement ; les applications doivent implémenter leurs propres mécanismes au niveau applicatif.

TCP

TCP est un protocole de niveau transport très complet et fiable : si des données se perdent, il le détecte et met en place des mécanismes pour les retransmettre. La plupart des applications réseaux utilisent TCP : le Web (HTTP), le courrier électronique (SMTP, POP3, IMAP), le transfert de fichier (FTP), etc.
TCP utilise aussi les numéros de port, comme UDP, pour identifier le processus applicatif auquel le segment TCP appartient. Il y ajoute un numéro de séquence et un numéro d'acquittement de 32 bits. Les segments TCP sont de taille variée : le numéro de séquence de chaque segment est le numéro du premier octet du segment ; le numéro d'acquittement indique le numéro du prochain octet attendu. Un acquittement numéro n indique aussi que le récepteur a bien reçu tous les octets depuis le début de la connexion jusqu'à l'octet numéro $n-1$ inclus et il attend l'octet numéro n. Comme TCP numérote chaque octet au lieu de chaque message, ce protocole est dit « orienté octet ».



L'illustration décrit la transmission de données dans une connexion TCP de bout-en-bout. Le processus émetteur écrit des octets de données à envoyer dans un tampon mémoire (*buffer*) d'émission. Le protocole TCP forme des segments et les envoie à la machine destinataire. Les segments reçus sont remis dans l'ordre dans le *buffer* de réception et le processus récepteur lit des octets à sa guise.