DUT GEII
Module complémentaire ARS21

Ce module permet d'appronfondir les notions de réseaux informatiques étudiées dans le module ARS3, et notamment la configuration de base des stations en réseau IP/Ethernet, la compréhension des mécanismes mis en œuvre dans les réseaux locaux : adressage, protocoles ARP, éléments d'interconnection, routage statique. Il se conclue par la configuration d'un serveur web (Apache).
L'enseignement est essentiellement pratique : quelques séances de cours/exercices (TD), et une séquence de travaux pratiques (TP) évalués.
Ce document rassemble les rappels de cours, sujets d'exercices et de travaux pratiques.

Table des matières

1 Introduction à UNIX

1.1 Les outils UNIX

UNIX est livré avec un grand nombre de programmes utilitaires. Ces programmes sont très divers, mais surtout orientés vers le traitement de fichiers de textes et le développement de logiciels. Tout système UNIX inclut normalement un compilateur C (certains vendeurs tendent à abandonner cet usage).

Les utilitaires les plus important sont les suivants :
- Interpréteurs de commandes (nommés shells), permettant l'accès d'un utilisateur au système. Les shells sont assez sophistiqués et s'apparentent à de véritables langages de programmation interprétés.
- Commandes de manipulation de fichiers ;
- Commandes de gestion des processus ;
- Éditeurs de texte ;
- Outils de développement : compilateurs, débugueurs, analyseurs lexicaux et syntaxiques, etc.

Nous détaillerons certains de ces outils par la suite.

1.2 Les utilisateurs, l'accès au système

Chaque utilisateur humain du système doit disposer d'un *compte* protégé par un mot de passe. Lorsque l'on se connecte à la machine, on doit fournir son nom et son mot de passe. Le nom est un pseudonyme attribué une fois pour toute à un utilisateur par l'administrateur du site. Le mot de passe peut être modifié par l'utilisateur aussi souvent qu'il le désire.

Avec la généralisation des interfaces graphiques, l'accès à un système sous UNIX s'est diversifié et (théoriquement) simplifié. La procédure d'entrée d'un utilisateur dans le système se nomme le **login**. La sortie est donc le **logout**. Lors du login, l'utilisateur devra toujours fournir son nom d'utilisateur et un mot de passe. Après vérification du mot de passe, le système lance un interpréteur de commande (shell).

Chaque utilisateur dispose de ses propres fichiers, dont il peut autoriser ou non l'accès aux autres utilisateurs. Il dispose d'un certain nombre de *droits* (accès à certains périphériques, etc).

Il peut lancer l'exécution de processus (le nombre maximal de processus par utilisateur peut être limité sur certains sites). Les processus lancés par un utilisateur ont les mêmes droits d'accès que lui.

2 Les fichiers UNIX

Le système de fichier est géré par le noyau. Les fichiers UNIX correspondent soit à des fichiers de données sur disque dur, soit à des répertoires, soit encore à des fichiers spéciaux permettant la gestion de certaines ressources du système (par exemple, lorsqu'un périphérique d'entrées/sorties permet des opérations comme ouverture, écriture, lecture (terminal, imprimante), on y accède généralement par un fichier spécial (device)).

2.1 Répertoires

Les fichiers sont organisés en répertoires et sous-répertoires, formant une arborescence (figure 2).

Dans chaque répertoire, on trouve au moins deux fichiers, nommés `.` (point) et `..` (point point). Le premier (`.`) permet de référencer le répertoire lui même, et le second (`..`) d'accéder au répertoire parent (du dessus).

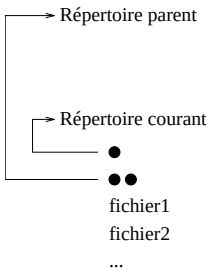


FIGURE 1 – Répertoire "." et "..".

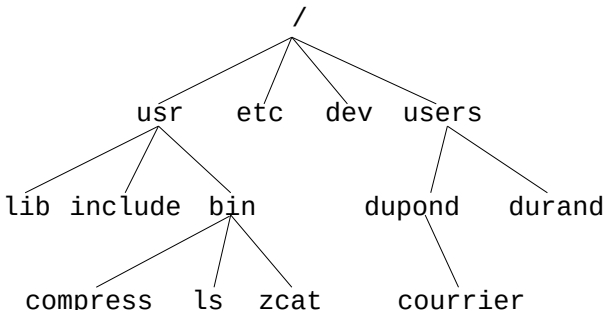


FIGURE 2 – Exemple d'arborescence de fichiers sous unix. La racine de l'arbre est le répertoire / (en haut). Ce répertoire contient ici 4 sous-répertoires.

A chaque instant, toute tâche possède un répertoire courant, ou répertoire de travail. La commande `pwd` affiche ce répertoire. La commande `cd` permet de changer le répertoire courant (voir plus loin page 8).

2.2 Chemins absolus et relatifs

Pour désigner un fichier quelconque, on peut utiliser soit un chemin absolu, soit un chemin relatif.

Un chemin absolu spécifie la suite des répertoires à traverser en partant de la racine, séparés par des caractères `/` (et non `\` comme sous DOS). Par exemple, le chemin

```
/usr/bin/compress
```

désigne le fichier `compress`, qui se trouve dans le répertoire `bin`, lui même dans le répertoire `usr` de la racine. Le premier caractère `/` indique qu'il s'agit d'un chemin absolu.

Il est souvent pratique d'utiliser un chemin relatif, à partir du répertoire courant. Par exemple, si l'on travaille dans le répertoire `dupond` de la figure 2, on peut accéder au fichier `durand` en spécifiant le chemin

```
../durand
```

Du même endroit, on peut accéder au fichier `compress` via le chemin

```
../../usr/bin/compress
```

(dans ce cas précis, il est plus simple d'utiliser le chemin absolu).

Tout chemin qui ne commence pas par un caractère `/` (prononcé *slash*) est interprété comme un chemin relatif au répertoire courant. On peut ainsi accéder aux fichiers du répertoire courant en donnant simplement leur nom.

2.3 Répertoire de connexion

A chaque utilisateur connu du système est associé un répertoire de connexion (*home directory*). L'utilisateur y place ses fichiers personnels, et peut y créer autant de sous-répertoires qu'il le désire. Dans l'exemple figure 2, le répertoire de connexion de M. Dupond est `/users/dupond`.

Après le login, l'interpréteur de commande a pour répertoire courant le répertoire de connexion de l'utilisateur.

Le répertoire de connexion contient aussi certains fichiers de configuration (voir section 3.3) permettant à l'utilisateur de personnaliser son environnement de travail. Ces fichiers sont normalement invisibles (car leur nom commence par un point, voir la commande `ls`).

A tout moment, on peut revenir au répertoire de connexion grâce à la commande `cd`.

On peut aussi spécifier un chemin à partir du répertoire de connexion d'un utilisateur en utilisant le caractère `~`. Par exemple,

```
~dupond/courrier
```

désigne le fichier

```
/users/dupond/courrier
```

2.4 Types de fichiers

Nous appelons *fichier* tout point dans l'arborescence des fichiers. Tous ne correspondent donc pas à des fichiers de données ordinaires. On distingue 4 types de fichiers :
- les fichiers ordinaires, qui contiennent des données. UNIX ne fait aucune différence entre les fichiers de texte et les fichiers binaires. Dans un fichier texte, les lignes consécutives sont séparées par un seul caractère '`\n`'.
- les répertoires, qui contiennent une liste de références à d'autres fichiers UNIX ;
- les fichiers spéciaux, associés par exemple à des pilotes de périphériques ;
- les tubes et sockets, utilisés pour la communication entre processus ;
- les liens symboliques (fichiers "pointant" sur un autre fichier).

2.5 Droits d'accès

A chaque fichier est associé un utilisateur propriétaire et un ensemble de *droits d'accès*. Les droits d'accès définissent les possibilités de lecture, écriture et exécution du fichier pour les utilisateurs.

2.6 Lecture, écriture et exécution d'un fichier

2.7 Utilisateurs et droits

Les utilisateurs d'un fichier donné sont divisés en trois ensembles :
- le propriétaire du fichier ;
- les utilisateurs du même groupe de travail que le propriétaire ;
- les autres utilisateurs ayant accès au système.

Un utilisateur appartenant à l'un de ces ensembles à accès ou non au fichier en lecture (r), en écriture (w) ou en exécution (x). Ces droits (ou permissions) d'accès ne peuvent être changés que par le propriétaire du fichier, grâce à la commande `chmod` (voir page 8).

La commande `ls -l` permet d'afficher les droits d'accès à un fichier ; par exemple :

```
$ ls -l polyunix.tex
-rwxr-----   1 emmanuel users   67504 Mar 25 23:29 polyunix.tex
```

indique que fichier `polyunix.tex` contient 67504 caractères et appartient à l'utilisateur `emmanuel`. La date et l'heure indiquées sont celles de la dernière modification du contenu du fichier.

Les caractères en début de ligne (`-rwxr-----`) indiquent le type et les droits d'accès sur ce fichier. Le premier caractère donne le type, ici `-` dénote un fichier ordinaire. Les neufs caractères restants sont divisés en trois groupes de trois, indiquant respectivement les droits du propriétaire du fichier, les droits des utilisateurs du même groupe que le proprétaire, et enfin les droits des autres utilisateurs. Le caractère `r` correspond au droit le lecture (read), `w` au droit d'écriture (write) et `x` au droit d'exécution.

Le fichier `polyunix.tex` montré ci-dessus est donc accessible en lecture, écriture et exécution par son propriétaire, en lecture par les utilisateurs du même groupe et pas du tout aux autres.

2.8 Le super-utilisateur

Afin de permettre l'administration du système, un utilisateur spécial, nommé super utilisateur (ou root), est toujours considéré par le système comme propriétaire de tous les fichiers (et des processus).

La personne qui gère le système est normalement la seule à connaître son mot de passe. Lui seul peut ajouter de nouveaux utilisateurs au système.

3 Commandes de base (shell)

Un *shell* est un interpréteur de commande en mode texte. Il peut s'utiliser en mode interactif ou pour exécuter des programmes écrits dans le langage de programmation du shell (appelés *shell scripts*).

En mode interactif, le shell affiche une invite en début de ligne (*prompt*), par exemple un caractère `$`, pour indiquer à l'utilisateur qu'il attend l'entrée d'une commande. La commande est interprétée et exécutée après la frappe de la touche "Entrée". Voici un exemple d'utilisation d'un shell ; les lignes débutants par `$`, sont entrées par l'utilisateur, les autres sont affichées en réponse :

```
$ pwd
/users/emmanuel/COURS/SYSTEME/POLYUNIX
$ ls
Makefile      polyunix.dvi  polyunix.tex
fig           polyunix.idx  polyunix.toc
hello.c       polyunix.ind  ps
$ ls fig
arbounix.fig  tabdesc.fig   tube.fig
$ ls -l *.c
-rw-r--r--   1 emmanuel users       84 Mar 25  1996 hello.c
```

Chaque ligne entrée par l'utilisateur est interprétée par le shell comme une commande, dont il lance l'exécution. Le premier mot de la ligne est le nom de la commande (par exemple `pwd` ou `ls`) ; il est éventuellement suivi d'un certain nombre d'arguments (par exemple `fig` ou `-l`).

3.1 Les différents shells

Il existe plusieurs shells UNIX : C-Shell (`csh` ou `tcsh`), Bourne Shell (`sh` ou `bash`), Korn shell (`ksh`), ... L'interprétation des commandes simples est semblable pour tous ; par contre l'utilisation pour écrire des scripts diffère beaucoup (définition des variables, structures de contrôle, etc).

Les variantes `tcsh` et `bash` apportent un plus grand confort d'utilisation en mode interactif (historique, terminaison automatique des commandes, etc) ; `tcsh` est compatible avec `csh`, et `bash` avec `sh`. *De nos jours, le shell* `bash` *s'est imposé comme le standard sur le système Linux, nous en recommandons vivement l'utilisation.*

Le point commun à tous les shells est l'emploi d'une syntaxe concise mais obscure et difficilement mémorisable, rendant leur apprentissage difficile (mais leur usage assez divertissant à la longue !). Il

est difficile d'administrer finement un système UNIX sans posséder quelques bases sur `sh` et `csh`, car de nombreux scripts de configuration sont écrits dans ces langages. La tendance actuelle est de généraliser l'emploi d'interfaces graphiques, qui restent toutefois moins souples et puissantes que les scripts. D'autre part, les scripts complexes sont de plus en plus souvent écrits dans des langages interprétés plus puisants comme Python ou Perl.

Dans les sections suivantes, nous décrivons brièvement les commandes du shell les plus utilisées. Les commandes sont groupées par thème. Pour retrouver rapidement une commande, utilisez l'index à la fin de ce document.

Pour plus de détails sur ces commandes ou sur leurs options, se reporter au manuel en ligne (commande `man`).

3.2 Métacaractères du shell

Certains caractères, appelés *métacaractères*, sont interprétés spécialement par le shell avant de lancer la commande entrée par l'utilisateur.

Par exemple, si l'on entre `ls *.c`, le shell remplace l'argument `*.c` par la liste des fichiers du répertoire courant dont le nom termine par `.c`.

Les métacaractères permettent donc de spécifier facilement des ensembles de fichiers, sans avoir à rentrer tous leurs noms. Voici les plus utilisés :
– `*` remplacé par n'importe quelle suite de caractères ;
– `?` remplacé par un seul caractère quelconque ;
– `[ ]` remplacé par l'un des caractères mentionnés entre les crochets. On peut spécifier un intervalle avec `-` : `[a-z]` spécifie donc l'ensemble des lettres minuscules.
Exemples :

```
$ ls
ABCDEF   a        grrr     prog     prog.o
Q.R.S    aa       hel.l.o  prog.c   x.y.z
$ ls A*
ABCDEF
$ ls *.c
prog.c
$ ls *g*
grrr    prog    prog.c  prog.o
$ ls *.?
Q.R.S    hel.l.o  prog.c   prog.o   x.y.z
$ ls [hg]*
grrr     hel.l.o
$ ls *.[a-z].*
hel.l.o  x.y.z
```

On peut empêcher l'interprétation des métacaractères par le shell en plaçant l'argument entre apostrophes `'`.

3.3 Initialisation d'un shell

Lors de leur démarrage, les shell exécutent des fichiers de configuration, qui peuvent contenir des commandes quelconques et sont généralement utilisés pour définir des variables d'environnement et des alias.
– `csh` exécute le fichier `~/.cshrc` (le "rc" signifie *run command*) ;
– `tcsh` exécute `~/.cshrc`, ou à défaut (si ce dernier n'existe pas) `~/.cshrc` ;
– `sh` exécute `~/.profile` ;
– `bash` exécute `~/.bash_profile` ou à défaut `~/.profile`.
Rappelons que les fichiers dont le nom commence par un point ne sont pas affichés par la commande `ls` (sauf si l'on emploie l'option `-a`) ; les fichiers d'initialisation sont donc "invisibles".

3.4 Variables d'environnement

Le système unix défini pour chaque processus une liste de variables d'environnement , qui permettent de définir certains paramètres : répertoires d'installation des utilitaires, type de terminal, etc. Chaque programme peut accéder à ces variables pour obtenir des informations sur la configuration du système.

Depuis le shell `sh` (ou `bash`), les variables d'environnement sont manipulées par les commandes `env` (affiche la liste), `export VARIABLE=VALEUR` (donne une valeur à une variable), et `echo $VARIABLE` (affiche la valeur de la variable).

Pour plus de détails, voir la section sur les scripts (page **??**).

3.5 Commandes de manipulation des fichiers

cat [*fichier1 ...*]

Recopie les fichiers spécifiés l'un après l'autre sur la sortie standard (concaténation). Si aucun argument n'est spécifié, lit sur l'entrée standard (jusqu'à rencontrer un caractère fin de fichier CTRL-d).
La sortie standard est normalement l'écran, et l'entrée standard le clavier (voir plus loin section 3.5), donc `cat fichier` affiche simplement à l'écran le contenu du fichier spécifié.

cd [*chemin*]

Change le répertoire courant. Sans argument, ramène dans le répertoire de connexion (HOME).

chmod *mode fichier*

Modifie les droits d'accès au fichier. Les droits sont spécifiés sous la forme : ensemble d'utilisateurs +/- type de droit.
Exemples :

```
chmod a+r  boite  # donne le droit a tous (a)
                                  de lire (r) boite;
chmod og-w boite  # interdit (-) aux autres (o)
                     et au groupe (g) d'ecrire (w);
chmod u+x  boite  # donne le droit d'execution (x)
                    au proprietaire (u) du fichier boite.
```

On peut aussi exprimer les droits sous forme d'un nombre exprimé en base 8 (octal) : chaque groupe de droits (rwx) est codé sur 3 bits.

Valeur	droits
0	`--`
1	`-x`
2	`-w-`
3	`-wx`
4	`r-`
5	`r-x`
6	`rw-`
7	`rwx`

Ainsi,

```
chmod 777  boite  # donne les droits rwx a tous (a)
chmod 750  boite  # rwx pour le propriétaire, rx pour le groupe
                     et rien pour les autres.
```

cp [-ipra] *source... dest*

Si *dest* est un répertoire, copie le ou les fichier(s) *source* vers *dest*. Si *dest* est un nom de fichier, renomme *source*.
Principales options :

-i demander confirmation en cas d'écrasement de la destination ;

-p préserve les dates d'accès et de modification ;

-r copie récursive (descend les sous-répertoires

-a copie récursive avec préservation des dates et modes. Utile pour sauvegardes. Voir aussi sur ce thème la commande `tar` page 14).

echo [-n] *message*

Affiche les arguments, tels qu'il sont évalués par le shell. L'option -n supprime le saut de ligne.

ls [-aldF] $chemin_1$ $chemin_2$ ... $chemin_n$

$chemin_i$ est un nom de fichier ou de répertoire. Si c'est un fichier, affiche sa descrition ; si c'est un répertoire, affiche la description de son contenu.

Options :
- -a liste tous les fichiers (y compris les .* normalement cachés).
- -l format long (taille, date, droits, etc).
- -d décrit le répertoire et non son contenu.
- -F format court avec indication du type de fichier (ajoute * si exécutable, / si répertoire).
- -i affiche les numéros d'inode des fichiers.

mkdir [*chemin*]

Crée un répertoire. Le chemin peut être relatif (par exemple `mkdir ../exam`) ou absolu (par ex. `mkdir /users/emmanuel/cours`).

mv [-i] *source dest*

Si *dest* est un répertoire, déplace le fichier *source* vers *dest*. Si *dest* est un nom de fichier, renomme *source*. L'option -i permet de demander confirmation en cas d'écrasement de la destination.

pwd

Affiche le répertoire courant.

rm [-ri] *fichier ...*

Supprime le ou les fichiers spécifiés. L'option -i permet de demander confirmation pour chacun. L'option -r agit de façon récursive, c'est à dire détruit aussi les répertoires (pleins ou vide) et leurs sous-répertoires.

3.6 Redirections des entrées/sorties

Chaque programme sous UNIX dispose au moins de deux flux de données : l'*entrée standard*, utilisée en lecture, qui est normalement associée au clavier du terminal, et la *sortie standard*, utilisée en écriture, normalement associée à l'écran du terminal (ou à la fenêtre de lancement le cas échéant).

Tous les flux de données sont manipulés comme de simples fichiers : on utilisera par exemple la même commande pour lire un caractère au clavier, dans un fichier sur disque ou via une liaison réseau. Ceci simplifie grandement l'écriture des programmes et améliore leur réusabilité.

3.6.1 Redirections vers/depuis des fichiers

Il est très simple de *rediriger* l'entrée ou la sortie standard d'un programme lors de son lancement depuis un shell UNIX.

Pour la sortie, on utilisera la construction suivante :

```
ls > resultat
```

Dans ce cas, au lieu d'afficher sur l'écran, la commande `ls` va créer un fichier nommé ici `resultat` et y écrire. Rien n'apparait à l'écran (sauf s'il se produit une erreur).

Si l'on désire ne pas effacer l'ancien contenu du fichier, mais écrire à sa fin, on peut utiliser `>>` :

```
ls >> resultats
```

Enfin, pour rediger l'entrée standard, on utilise `<` :

```
cat < UnFichier
```

Il est possible de rediriger l'entrée et la sortie en même temps :

```
cat < UnFichier > Resultat
```

Notons enfin qu'il existe un troisième flux standard, utilisé pour l'écriture des messages d'erreur (nommé `stderr` en C, ou `cerr` en C++). Il se redirige avec `>&`.

3.6.2 Redirections vers des tubes

De façon similaire, il est possible de rediriger la sortie standard d'une commande vers l'entrée standard d'une autre commande grâce au tube (*pipe*) noté `|`. Les différentes commandes s'exécutent alors en parallèle.

```
# affiche le contenu du repertoire trie a l'envers
ls | sort -r
```

3.7 Contrôle de tâches

Normalement, le shell attend la fin de l'exécution d'une commande avant d'afficher le prompt suivant. Il arrive que l'on désire lancer une commande de longue durée tout en continuant à travailler dans le shell. Pour cela, il suffit de terminer la ligne de commande par le caractère `&` (et commercial).

Par exemple :

```
$ calcul &
$ ls -Ral / > ls-Ral.txt &
$
```

Ici, les deux commandes s'exécutent en parallèle, tandis que le shell attend notre prochaine instruction [1].

On dit que les processus s'exécutent en *tâches de fond*. Pour connaître la liste des tâches de fond lancées de ce shell, utiliser la commande `jobs` :

```
$ jobs
[1]    Running                 calcul
[2]    Running                 ls -Ral / > ls-Ral.txt
$
```

Le nombre entre crochets est le numéro de la tâche de fond (job). On peut envoyer un signal à une tâche en utilisant `kill` et `%` à la place du PID :

```
$ kill [-signal]  %numero_de_tache
```

1. Question : que fait la deuxième commande lancée ?

Pour remettre une tâche de fond au "premier plan", utiliser la commande `fg` (utile si la tâche réalise une entrée clavier par exemple).

Enfin, la commande `sleep n` suspend l'exécution durant n secondes (le processus shell passe à l'état bloqué durant ce temps).

3.8 Comment le système exécute une commande ?

Lorsque l'on entre une ligne de commande sous un shell, ce dernier demande au système d'exécuter la commande, après avoir éventuellement substitué les méta-caractères (*, ? etc.) et remplacés les alias.

A chaque commande doit alors correspondre un fichier *exécutable* (avec le droit `x`). Ce fichier exécutable porte le nom de la commande et est recherché par le système dans une liste de répertoires (spécifiée par la variable d'environnement `PATH`).

La commande `which` permet de savoir quel est le fichier correspondant à une commande.

which *commande*

Affiche le chemin du fichier exécutable correspondant à la commande. Exemple :

```
$ which csh
/bin/csh
```

Le fichier exécutable est soit un fichier binaire (du code en langage machine), qui est alors exécuté directement par le processeur, soit un fichier texte contenant un *script* (voir chapitre **??**).

4 Utilitaires UNIX

Cette partie décrit quelques commandes utilitaires UNIX très pratiques. Ces programmes sont livrés en standard avec toutes les versions d'UNIX.

Nous commençons par décrire l'utilisation de l'éditeur standard `vi`. Si vous avez accès à un autre éditeur de texte, vous pouvez passez cette section.

4.1 L'éditeur vi

`vi` est un éditeur de texte "plein écran" (par opposition aux éditeurs "ligne" qui ne permettaient que l'édition d'une ligne à la fois).

Comparé aux éditeurs modernes, `vi` est d'abord malcommode, mais il est assez puissant et toujours présent sur les systèmes UNIX [2]. Il est donc très utile d'en connaître le maniement de base. Nous ne décrivons ici que les commandes et modes les plus utilisés, il en existe beaucoup d'autres.

A un instant donné, `vi` est soit en mode **commande**, soit en mode **insertion** :
- **mode commande :** les caractères tapés sont interprétés comme des commandes d'édition. `vi` démarre dans ce mode, il faut donc lui indiquer (commande '`i`') que l'on veut `i`nsérer du texte ;
- **mode insertion :** les caractères sont insérés dans le texte édité. On peut quitter ce mode en pressant la touche "ESC" (ou "Echap" sur certains claviers).

2. Notons qu'il existe sous UNIX des éditeurs pour tous les goûts, depuis Emacs pour les développeurs jusqu'à gedit et autres jeditpour les utilisateurs allergiques au clavier (mais pas à la souris).

Appel de vi depuis le shell :

`vi` *fichier*	édite *fichier*.
`vi` +n *fichier*	commence à la ligne *n*.
`vi` -r *fichier*	récupération du fichier après un crash.

Mouvements du curseur (en mode **commande** seulement) :

Touche	**Action**
flèches	Déplace curseur (pas toujours bien configuré).
ESPACE	Avance à droite.
h	Recule à gauche.
CTRL-n	Descend d'une ligne.
CTRL-p	Monte d'une ligne.
CTRL-b	Monte d'une page.
CTRL-f	Descend d'une page.
*n*G	Va à la ligne *n* (*n* est un nombre).

Commandes passant en mode **insertion** :

Touche	**Commence l'insertion**
i	Avant le curseur.
I	Au début de la ligne.
A	A la fin de la ligne.

Autres commandes :

r	Remplace le caractère sous le curseur.
x	Supprime un caractère.
d$	Efface jusqu'à la fin de la ligne.
dd	Efface la ligne courante.
/*chaîne*	Cherche la prochaine occurrence de la chaîne.
?*chaîne*	Cherche la précédente occurrence de la chaîne.

Quitter, sauvegarder : (terminer la commande par la touche "Entrée")

:w	Écrit le fichier.
:x	Écrit le fichier puis quitte `vi`.
:q!	Quitte `vi` sans sauvegarder les changements.
!!*commande*	Exécute *commande* shell sans quitter l'éditeur.

4.2 Commandes diverses

compress [*fichier*]

Compresse le fichier (pour gagner de l'espace disque ou accélérer une transmission réseau). Le fichier est remplacé par sa version compressée, avec l'extension '.Z'. Décompression avec `uncompress` ou `zcat`.

date

Affiche la date et l'heure. Comporte de nombreuses options pour indiquer le format d'affichage.

diff *fichier1 fichier2*

Compare ligne à ligne des deux fichiers texte *fichier1* et *fichier2*, et décrit les transformations à appliquer pour passer du premier au second. Diverses options modifient le traitement des blancs, majuscules etc.
`diff` peut aussi générer un script pour l'éditeur `ed` permettant de passer de *fichier1* à *fichier2* (utile pour fabriquer des programmes de mise à jour ("*patchs*")).

file *fichier*

Essaie de déterminer le type du fichier (exécutable, texte, image, son,...) et l'affiche.

find [options]

Cette commande permet de retrouver dans un répertoire ou une hiérarchie de répertoires les fichiers possédant certaines caractéristiques (nom, droits, date etc..) ou satisfaisant une expression booléenne donnée.
find parcourt récursivement une hiérarchie de fichiers. Pour chaque fichier rencontré, find teste successivement les prédicats spécifiés par la liste d'options, jusqu'au premier qui échoue ou jusqu'à la fin de la liste.
Principales options :

-name nom le fichier à ce nom ;

-print écrit le nom du fichier (réussit toujours) ;

-exec exécute une commande. {} est le fichier courant. Terminer par `;` .

-type (d : catalogue, f : fichier ordinaire, p : pipe, l : lien symbolique).

-newer fichier compare les dates de modification ;

-o ou ;

-prune si le fichier courant est un catalogue, élague l'arbre à ce point.

Exemples :

```
#  Recherche tous les fichier nommes "essai"
#  a partir de la racine
find / -name essai -print

#  Recherche tous les fichier commencant par "ess"
#  a partir du repertoire courant
find . -name 'ess*' -print

#  Affiche a l'ecran le contenu de tous les fichiers .c
find . -name '*.c' -print -exec cat '{}' \;
```

(l'écriture de ce dernier exemple est compliquée par le fait que le shell traite spécialement les caractères `*`, `{`, et `;`, que l'on doit donc entourer de quotes `'`.)

head [-n] [*fichier*]

Affiche les n premières lignes du fichier. Si aucun fichier n'est spécifié, lit sur l'entrée standard.

lpr [-P*nom-imprimante*] [*fichier*]

Demande l'impression du fichier (le place dans une file d'attente). Si aucun fichier n'est spécifié, lit sur l'entrée standard.

Voir aussi la commande `lp`.

L'impression d'un fichier sous Unix passe par un spooler d'impression. Ce spooler est réalisé par un démon (c'est à dire un processus système qui s'exécute en tâche de fond).

lpq [-P*nom-imprimante*]

Permet de connaître l'état de la file d'attente associée à l'imprimante.

lprm [-P*nom-imprimante*] *numjob*

Retire un fichier en attente d'impression. On doit spécifier le numéro du job, obtenu grâce à la commande `lpq`.

lp [-d*nom-imprimante*] *fichier*

Comme `lpr`, sur certains systèmes.

md5sum [*fichier*]

Calcule et/ou vérifie la somme "MD5" d'un fichier ou flux. Voir RFC 1321.

man [n] *commande*

Affiche la page de manuel (aide en ligne) pour la commande. L'argument n permet de spécifier le numéro de la section de manuel (utile lorsque la commande existe dans plusieurs sections). Les numéros des sections sont : 1 (commandes utilisateur), 2 (appels systèmes), 3 (fonctions librairies C), 4 (devices), 5 (formats de fichiers), 6 (jeux), 7 (divers), 8 (administration) et n (new, programmes locaux).

Voir aussi le lexique page 23.

more [*fichier*]

Affiche un fichier page par page (aide en ligne, taper 'h'). Si aucun fichier n'est spécifié, lit sur l'entrée standard.

less [*fichier*]

less is more. Comme more, avec plus de fonctionnalités (taper 'h' pour en savoir plus).

tail [+n | -n] [*fichier*]

La forme `tail +n` permet d'afficher un fichier à partir de la ligne n. La forme `tail -n` affiche les n dernières lignes du fichier. Si aucun fichier n'est spécifié, lit sur l'entrée standard.

tar options [*fichier ou répertoire*]

La commande `tar` permet d'archiver des fichiers ou une arborescence de fichiers, c'est à dire de les regrouper dans un seul fichier, ce qui est très pratique pour faire des copies de sauvegardes d'un disque, envoyer plusieurs fichiers en une seul fois par courrier électronique, etc.

Pour créer une nouvelle archive, utiliser la forme

```
$ tar cvf nom-archive repertoire
```

qui place dans le nouveau fichier "nom-archive" tous les fichiers situés sous le répertoire indiqué. On donne généralement l'extension `.tar` aux fichiers d'archives.

Pour afficher le contenu d'une archive, utiliser

```
$ tar tvf nom-archive
```

Pour extraire les fichiers archivés, utiliser

```
$ tar xvf nom-archive
```

les fichiers sont crées à partir du répertoires courant.

Nombreuses autres options. Notons que les noms de fichiers peuvent être remplacés par `-` pour utiliser l'entrée ou la sortie standard (filtres), comme dans les exemples ci-dessous :

- Archivage d'un répertoire et de ses sous-répertoires dans un fichier `archive.tar` :
  ```
  $ tar cvf archive.tar repertoire
  ```
- Archivage et compression au vol :
  ```
  $ tar cvf - repertoire | gzip > archive.tar.gz
  ```
- Pour afficher l'index de l'archive ci-dessus :
  ```
  $ zcat archive.tar.gz | tar tvf -
  ```
- l'option `z` permet de (dé)comprimer directement. On utilse dans ce cas l'extension `.tgz` :
  ```
  $ tar cvfz  archive.tgz repertoire
  ```
- Copie complète récursive d'un répertoire `repert` dans le répertoire `destination` :
  ```
  $ tar cvf - repert | (cd destination; tar xvfp -)
  ```
  Les parenthèses sont importantes : la deuxième commande `tar` s'exécute ainsi dans le répertoire de destination.

  Cette façon de procéder est supérieure à `cp -r` car on préserve les propriétaires, droits, et dates de modifications des fichiers (très utile pour effectuer des sauvegardes).

uncompress [*fichier*]

Décompresse un fichier (dont le nom doit terminer par .Z) compressé par `compress`.

wc [-cwl] [*fichier* ...]

Affiche le nombre de caractères, mots et lignes dans le(s) fichier(s). Avec l'option -c, on obtient le nombre de caractères, avec -w, le nombre de mots (*words*) et avec -l le nombre de lignes.

which *commande*

Indique le chemin d'accès du fichier lancé par "commande".

who [am i]

Liste les utilisateurs connectés au système. La forme `who am i` donne l'identité de l'utilisateur.

zcat [fichiers]

Similaire à `cat`, mais décompresse au passage les fichiers (ou l'entrée standard) compressés par `compress`.

uuencode [fichier] nom

Utilisé pour coder un fichier en n'utilisant que des caractères ascii 7 bits (codes entre 0 et 127), par exemple pour le transmettre sur un réseau ne transmettant que les 7 bits de poids faible.

`uudencode` lit le fichier (ou l'entrée standard si l'on ne précise pas de nom) et écrit la version codée sur la sortie standard. Le paramètre `nom` précise le nom du fichier pour le décodage par `uudecode`.

uudecode [fichier]

Décode un fichier codé par `uuencode`. Si l'argument `fichier` n'est pas spécifié, lit l'entrée standard. Le nom du fichier résultat est précisé dans le codage (paramètre `nom` de `uuencode`).

4.3 Filtres de texte

Les *filtres* sont des utilitaires qui prennent leurs entrées sur l'entrée standard, effectuent un certain traitement, et fournissent le résultat sur leur sortie standard. Notons que plusieurs utilitaires déja présentés peuvent être vus comme des filtres (head, tail).

grep [option] *motif* $fichier_1$ ... $fichier_n$

Affiche chaque ligne des fichiers $fichier_i$ contenant le motif *motif*. Le motif est une *expression régulière*.

Options :
- -v affiche les lignes qui ne contiennent *pas* le motif.
- -c seulement le nombre de lignes.
- -n indique les numéros des lignes trouvées.
- -i ne distingue pas majuscules et minuscules.

sort [-rn] [*fichier*]

Trie les lignes du fichier, ou l'entrée standard, et écrit le résultat sur la sortie. L'option -r renverse l'ordre du tri, l'option -n fait un tri numérique.

tr [options] *chaine1 chaine2*

Recopie l'entrée standard sur la sortie standard en remplaçant tout caractère de *chaine1* par le caractère de position correspondante dans *chaine2*.

uniq [-cud] *fichier*

Examine les données d'entrée ligne par ligne et détermine les lignes dupliquées qui sont consécutives : -d permet de retenir que les lignes dupliquées. -u permet de retenir que les lignes non dupliquées. -c permet de compter l'indice de répétition des lignes.

4.4 Manipulation des processus

kill -*sig PID*

Envoie le signal *sig* au processus de numéro *PID*. Voir la table **??** (page **??**) pour une liste des signaux. *sig* peut être soit le numéro du signal, soit son nom (par exemple `kill -STOP 1023` est l'équivalent de `kill -19 1023`).

ps [-e][-l]

Affiche la liste des processus.
L'option -l permet d'obtenir plus d'informations. L'option -e permet d'afficher les processus de tous les utilisateurs.

ps affiche beaucoup d'informations. Les plus importante au début sont :
- `UID` : identité du propriétaire du processus ;
- `PID` : numéro du processus ;
- `PPID` : PID du père du processus ;
- `NI` : priorité (nice) ;
- `S` : état du processus (R si actif, S si bloqué, Z si terminé).

nice [-*priorite*] *commande*

Lance l'exécution de la commande en modifiant sa . Permet par exemple de lancer un processus de calcul en tâche de fond sans perturber les processus interactifs (éditeurs, shells etc.), en lui affectant une priorité plus basse.

Le noyau accorde moins souvent le processeur aux processus de basse priorité.

renice *priorite* -p pid

Modifie la priorité d'un processus.

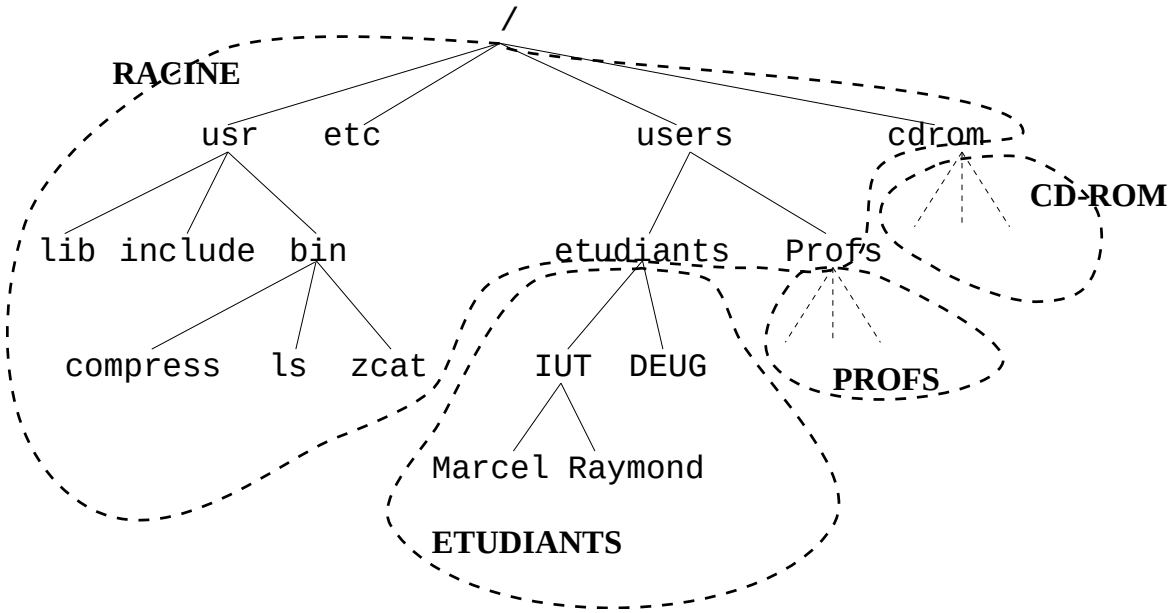


FIGURE 3 – Dans cet exemple, l'arborescence des fichiers est composée de quatre *systèmes de fichiers* : le système racine (point de montage /, deux systèmes pour les utilisateurs (/users/etudiants et /users/profs), et un système pour le CD-ROM (/cdrom).

4.5 Gestion des systèmes de fichiers

4.5.1 Principes

Tous les disques, disquettes et CD-ROMs connectés à un ordinateur sous UNIX sont accédés via une arborescence unique, partant du répertoire racine `/`. La complexité du système est ainsi masquée à l'utilisateur, qui peut utiliser les mêmes commandes pour accéder à un fichier sur disque dur ou sur CD-ROM.

Remarquez que ce principe est différent de celui employé par les systèmes MS-DOS et Windows, pour lesquels chaque *volume* (disque) possède une racine spécifique repérée par une lettre (`A:\`, `C:\`, etc).

Bien entendu, les fichiers sont organisés de façon différentes sur les disques durs, les disquettes, ou les CD-ROMS : chaque périphérique possède son propre *système de fichiers*.

Les différents systèmes de fichier, que l'on peut considérer comme des sous-arborescences associées à un périphérique spécial, sont *montées* sur l'arborescence racine. Chaque système de fichier doit posséder un point de montage, qui est au départ (avant montage) un simple répertoire vide (voir figure 3).

La commande `mount` permet d'associer au point de montage le système de fichier correspondant. Il faut lui préciser le pilote de périphérique (*device driver*) utilisé pour accéder à ce système. Nous n'étudions pas dans ce cours la gestion des pilotes de périphériques sous UNIX ; notons simplement que chaque pilote correspond à un pseudo-fichier dans le répertoire système `/dev`.

4.5.2 Commandes de base

Nous décrivons simplement trois commandes permettant d'observer l'état des systèmes de fichiers. Nous de décrivons pas leur utilisation pour administrer le système (ajouter un système de fichier, etc).

mount [*point d'entree*]

Permet d'afficher la liste des sytèmes de fichiers en service (montés). Le format d'affichage dépend légèrement de la version d'UNIX utilisée. Un exemple simple sous Linux :

```
$ mount
/dev/sda3 on / type ext2 (rw)
```

```
/dev/sda4 on /users type ext2 (rw)
/dev/sdb1 on /jaz type ext2 (rw)
```

On a ici trois systèmes de fichiers, gérés par les pilotes `/dev/sda3`, `/dev/sda4` et `/dev/sdb1` (trois disques durs SCSI), et montés respectivement sur la racine, `/users` et `/jaz`.

La commande `mount` est aussi utilisée pour monter un nouveau système de fichiers dans l'arborescence (lors du démarrage du système ou de l'ajout d'un disque).

df [*chemin*]

`df` (*describe filesystem*) affiche la capacité totale d'un système de fichiers (généralement en Kilo-Octets), le volume utilisé et le volume restant disponible.

Le format d'affichage dépend aussi de la version utilisée. Exemple sous Linux :

```
$ df /users
Filesystem  1024-blocks  Used Available Capacity Mounted on
/dev/sda4      417323  347789    47979     88%   /users
```

On a ici 417323 Ko au total, dont 347789 sont utilisés et 47979 libres.

du [-s] [*chemin*]

`du` (*disk usage*) affiche la taille occupée par le fichier ou répertoire spécifié par l'argument *chemin*. Avec l'option `-s`, n'affiche pas le détail des sous-répertoires.

Exemple :

```
$ du POLYUNIX
10      POLYUNIX/fig
19      POLYUNIX/ps
440     POLYUNIX
$ du -s POLYUNIX
440     POLYUNIX
```

Le répertoire `POLYUNIX` occupe ici 440 Ko.

5 Programmation en C sous UNIX

Nous décrivons dans cette section les bases de la programmation en langage C sous UNIX.

5.1 Le compilateur C

Par convention, les fichiers contenant du source C possèdent l'extension `.c`, les fichiers headers l'extension `.h` et les fichiers objets `.o`. Il n'y a pas de convention spéciale pour les exécutables qui doivent par contre posséder le droit d'exécution `x`.

Soit le fichier `hello.c` contenant le texte source suivant :

```
/* Exemple simple */
#include <stdio.h>
main() {
  printf("Hello, world !\n");
}
```

Le compilateur C peut s'appeler depuis une ligne de commande shell sous la forme :

```
$ cc hello.c
```

(`$` désigne le prompt du shell). Le programme `hello.c` est alors compilé et un fichier exécutable nommé `a.out` est créé. On peut lancer son exécution en tapant :

```
$ a.out
Hello, world !
```

Il est possible de spécifier le nom de l'exécutable produit grâce à l'option `-o` :

```
$ cc hello.c -o hello
```

La commande `cc` admet de nombreuses options sur la ligne de commande. Une option très utile est `-I` qui spécifie une répertoire où rechercher les fichiers inclus par la directive `#include`. Par exemple, si l'on a placé nos fichiers `.h` dans sous-répertoire `inc`, on pourra utiliser :

```
$ cc -Iinc exo.c -o exo
```

Sur certains systèmes, d'autres options sont nécessaire pour spécifier la variante de langage C utilisée (K&R ou ANSI) et l'utilisation ou non de la norme POSIX.

5.2 La commande make

Lorsque l'on développe un programme un peu plus important que l'exemple donné plus haut, il devient vite fastidieux d'utiliser directement la commande `cc`, avec les bonnes options et sans oublier de recompiler tous les fichiers que l'on a modifiés depuis la dernière mise à jour.

Supposons que l'on travaille sur un projet comportant deux fichiers sources, `main.c` et `func.c`, qui utilisent tous deux un fichier `inc/incl.h` placé dans un sous-répertoire. L'exécutable que l'on fabrique est nommé `essai`.

En utilisant `cc`, il faut faire :

```
$ cc -c -Iinc -o main.o  main.c
$ cc -c -Iinc -o func.o  func.c
$ cc -o essai main.o func.o
```

L'option `-c` spécifie que l'on veut fabriquer un fichier objet (.o) qui sera lié ultérieurement pour obtenir un exécutable.

Si l'on modifie `main.c`, il faut refaire les étapes 1 et 3. Si l'on modifie le fichier inclut, il faut tout refaire. On dit que `essai` *dépend* de `main.o` et `func.o`, qui dépendent eux même de `main.c`, `func.c` et de `incl.h`.

Ceci signifie que notre exécutable final (`essai`) est à jour si sa date de modification est plus grande que celle de `main.o` et `func.o`, et ainsi de suite.

La commande *make* permet d'automatiser ce genre de chaînes de production. Son utilisation permet au développeur de s'assurer qu'un module donné est toujours à jour par rapport aux divers éléments dont il dépend. make se charge de déclencher les actions nécessaires pour reconstruire le module cible.

make [-f fichier-dep]

> make utilise un fichier spécial (appelé "makefile") indiquant les *dépendances* entre les divers fichiers à maintenir. Ce fichier est nommé par défaut `Makefile`.
>
> Syntaxe d'un fichier makefile :
>
> ```
> but : dependance1 [...[dependance_n]]
>         commande1
>         commande2
>         ...
> ```
>
> La ou les cibles doivent apparaître sur la première ligne et être suivies de ' :'. A la suite figure la liste des dépendances conduisant à la cible. Les règles de production sont décrites sur les lignes suivantes, qui doivent impérativement commencer caractère de tabulation.

Voici un exemple de fichier Makefile traitant l'exemple précédent :

```
all: essai
```

```
essai: main.o func.o
    cc -o essai func.o main.o

CFLAGS= -Iinc

func.o: inc/incl.h
main.o: inc/incl.h
```

La première ligne (all) indique que la cible par défaut est `essai` : lorsque l'on actionne `make` sans arguments, on veut fabriquer `essai`.

Ensuite, on indique que `essai` dépend des deux fichiers objets, et qu'il se fabrique à partir d'eux en actionnant `cc -o essai ...`.

`CFLAGS` est une variable spéciale indiquant les options à passer au compilateur C, ici `-Iinc`.

Enfin, on spécifie que les fichiers `.o` dépendent du fichier `.h`. Notons que make utilise un certain nombre de *règles par défaut*, par exemple celle disant que les fichiers `.o` dépendent des fichiers `.c` et se fabriquent en utilisant `cc`. Il n'est donc pas nécessaire de faire figurer la règle complète, qui s'écrirait :

```
main.o: inc/incl.h main.c
    cc -Iinc -c -o main.o main.c
```

5.3 Arguments sur la ligne de commande

L'exécution d'un programme nommé `exemple` est lancé depuis un shell par :

```
$ exemple
```

Si l'on entre

```
$ exemple un deux trois
```

le programme est lancé de la même façon, et il reçoit les chaînes de caractères `"un"`, `"deux"` et `"trois"` comme *arguments*. En langage C, ces arguments peuvent facilement être récupérés par la fonction `main()` du programme (qui, rappelons le, est la première fonction appelée lors du lancement de ce programme).

Il suffit de déclarer la fonction `main()` comme ceci :

```
void main( int argc, char *argv[] )
```

la variable `argc` reçoit le nombre de paramètres sur la ligne de commande. `argv` est un tableau de chaînes de caractères contenant les paramètres. Exemple : soit un exécutable nommé `essai`, si l'on entre

```
$ essai il fait beau
```

On aura :

```
argc = 4
argv[0] = "essai"
argv[1] = "il"
argv[2] = "fait"
argv[3] = "beau"
```

Notons que `argv[0]` contient toujours le nom du programme lancé.

5.4 Variables d'environnement

En langage C, on peut accéder à la liste de ces variables par l'intermédiaire du troisième argument de la fonction `main()`, qui est alors déclarée comme :

```
void main( int argc, char *argv[], char *arge[] )
```

`arge` est un tableau de chaînes de caractères définissant les variables d'environnement, sous la forme "`NOM=VALEUR`".

On peut aussi utiliser les fonctions `getenv()` et `setenv()`.

Attention, chaque processus hérite d'une copie des variables d'environnement de son père. Les modifications qu'il peut apporter à ces variables n'affecteront donc que ses descendants (les processus qu'il lancera), jamais le processus père.

5.5 Allocation mémoire

Pour allouer des blocs de mémoire dynamique sous UNIX, il est conseillé d'utiliser les fonctions de la librairie C standard : `malloc()` et `free()` en C, `new` et `delete` en C++.

```
#include <stdlib.h>
void *malloc( int nb_octets );
void *calloc( int nb_elem, int taille_elem );
void free( void *ptr );
```

La fonction `malloc()` alloue un bloc de mémoire contigüe de `nb_octets` octets. La fonction `calloc()` alloue un bloc de `nb_elem x taille_elem` octets et l'initialise à zéro. Les blocs alloués sont libérés après usage par la fonction `free()`.

5.6 Manipulation de fichiers

5.6.1 Fonctions de la librairie C

La librairie C standard permet de manipuler des fichiers (ouverture, lecture, écriture) via des pointeurs sur des flux de type `FILE`. Ces fonctions sont définies dans le fichier include `<stdio.h>`. Voici les principales fonctions utilisées (cf le cours de C) :

```
#include <stdio.h>
FILE *fopen( char *chemin, char *mode );
int fclose( FILE * );
int fprintf( FILE *, char *format, ... );
int fwrite( void *ptr, int size, int nbelem, FILE * );
int fread( void *ptr, int size, int nbelem, FILE * );
int fflush( FILE * );
int feof( FILE * );
```

Notez que ces fonctions existent aussi bien en C sous DOS que sous Macintosh ou UNIX. Elles doivent être utilisées si l'on désire écrire un programme *portable*. Dans ce cas, il est évident que l'on s'interdit l'usage de toutes les fonctionnalités propres à chaque système d'exploitation.

Note : les opérations d'entrées/sorties effectuées par les fonctions ci-dessus sont *bufferisées*; ainsi, l'appel à `fwrite()` ou `fprintf()` copie simplement les données dans un tampon (ou buffer), qui est vidé une fois plein vers le fichier concerné. Ceci permet d'éviter de multiplier inutilement les opérations d'entrées/sorties, souvent coûteuses. Une conséquence troublante pour le débutant est que lors de l'envoi de caractères vers l'écran, ceux ci n'apparaissent pas immédiatement, sauf si l'on force la vidange du tampon avec un retour chariot ou l'appel explicite à `fflush()`.

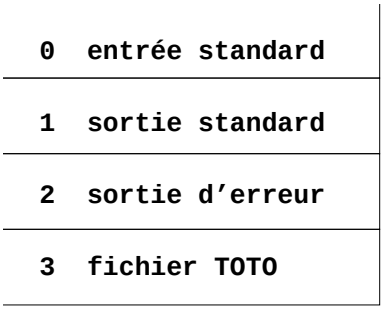


FIGURE 4 – Table des descripteurs d'un processus après ouverture d'un fichier TOTO. La prochain fichier ouvert se verrait attribuer le descripteur numéro 4.

5.6.2 Fichiers et descripteurs sous UNIX

Pour manipuler plus finement les fichiers UNIX, il est souvent nécessaire d'utiliser directement des appels système de plus bas niveau que ceux de la librairie C.

Le noyau UNIX maintient une table des fichiers ouverts par un processus. Le terme fichier désigne ici un *flux de données* unidirectionnel qui n'est pas nécessairement associé à un fichier disque.

Chaque processus dispose toujours à son lancement de trois flux de données :

1. l'*entrée standard*, associée normalement au clavier du terminal de lancement ;
2. la *sortie standard*, associée normalement à l'écran du terminal (ou à la fenêtre de lancement) ;
3. la *sortie d'erreur*, qui coïncide par défaut avec la sortie standard.

Nous avons vu dans la section 3.5 comment rediriger ces flux standard depuis le shell.

Un programme accède aux flux de données ouverts en spécifiant des *descripteurs*, qui sont simplement des nombre entiers correspondant à l'indice du flux dans la table. La table est remplie dans l'ordre d'ouverture des fichiers, en commençant à 3 puisque les trois premiers descripteurs sont toujours occupés par les flux standards (voir figure 4).

5.6.3 Appels systèmes manipulant les fichiers

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

int open( char *chemin, int mode, ... );
int close( int desc );
int read( int desc, void *ptr, int nb_octets );
int write( int desc, void *ptr, int nb_octets );
```

La fonction `open()` ouvre un fichier en lecture ou en écriture, crée un nouveau descripteur et le retourne.

`close()` ferme un descripteur.

`read()` et `write()` permettent de lire ou d'écrire une zone mémoire sur un descripteur.

5.7 Lancement d'une commande

Pour lancer l'exécution d'une commande externe depuis un programme C, on peut utiliser l'appel

```
int system ( char *comm );
```

Le processus qui appelle `system()` est bloqué jusqu'à la fin de la commande lancée.

Lexique pour comprendre le manuel UNIX

Le manuel en ligne d'UNIX est la principale source d'information sur les commandes du système. Il est rédigé dans un anglais technique très simple qui devrait vous être accessible sans efforts après acclimatation (de nombreuses pages sont maintenant traduites). Le lexique ci-dessous défini les termes informatiques les plus couramment utilisés dans ce manuel.

blank vide (ligne vide, écran vide).
by default par défaut, ce qui se passe si l'on ne spécifie rien d'autre.
byte octet (8 bits).
comma virgule.
concatenate "concaténer", c'est à dire mettre bout à bout.
directory répertoire.
file fichier.
handler traitant (d'interruption, de signal, ...), c'est à dire fonction gérant une situation spécifique.
head tête (début).
inhibit inhiber (empêcher, interdire).
job travail, tâche (processus contrôlé par un shell).
key touche (littéralement "clé").
parse analyser la syntaxe.
path chemin d'accès à un fichier ou répertoire.
pipe tube (communication interprocessus).
prompt demander (à l'utilisateur) ; ce qui s'affiche au début des lignes d'un interpréteur de commandes.
quote guillemet ' ; *double quote* = " ; *backquote* = ‘.
recursive récursif.
return retourner (d'une fonction) ; touche "entrée".
shell littéralement "coquille" ; désigne sous UNIX un interpréteur de commandes interactif.
skip sauter.
sort trier.
space espace.
status état.
syntax syntaxe.
tail queue (fin).
word mot.

Index

TD N° 1 - Révisions : programmation C

EXERCICE 1 - A l'aide du cours précédent, écrire un programme en langage C qui affiche un par un les arguments qu'on lui passe sur la ligne de commande.

Exemple : si le programme est nommé `exercice1`, si l'on entre la ligne de commande

```
$ exercice1 il fait beau
```

On aura :

```
argc = 4
argv[0] = "exercice1"
argv[1] = "il"
argv[2] = "fait"
argv[3] = "beau"
```

EXERCICE 2 - *Gestion des fichiers*

1- Écrire un programme C qui crée un fichier nommé "resultat" qui contienne la ligne de texte "Bonjour".

2- Modifier le programme précédent pour qui écrive dans le fichier les arguments de la ligne de commande, un par ligne, en les faisant précéder du numéro de ligne.

Exemple : si on lance

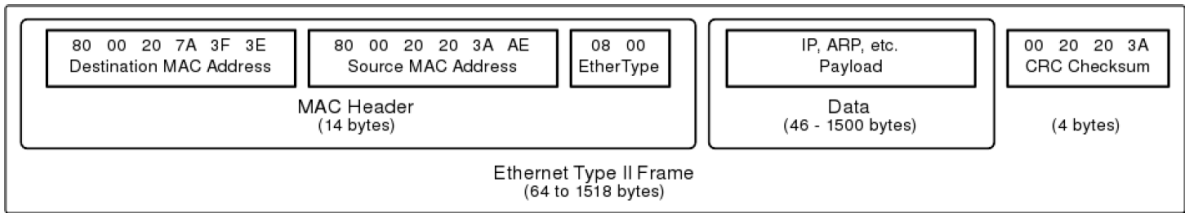
```
$ exercice2 il fait beau
```

On aura dans le fichier "resultat" :

```
1 il
2 fait
3 beau
```

TD N° 2 - Ethernet - ARP - IP

1 Rappel : structure de la trame Ethernet



2 Adresses Ethernet (MAC)

Quelques préfixes (codes constructeurs) d'adresses Ethernet. Voir `http://standards.ieee.org/regauth/oui/oui.txt` pour une liste plus complète.

`00:00:07`	Xerox
`00:00:17`	Tekelec
`00:04:AC`	IBM
`00:07:CB`	Freebox SA
`00:16:6F`	Intel Corporation
`00:1A:30`	Cisco Systems
`00:1B:E9`	Broadcom Corporation
`00:1E:74`	SAGEM Communication
`00:50:56`	VMWare, Inc.

3 Rappels sur le protocole IP

3.1 Introduction

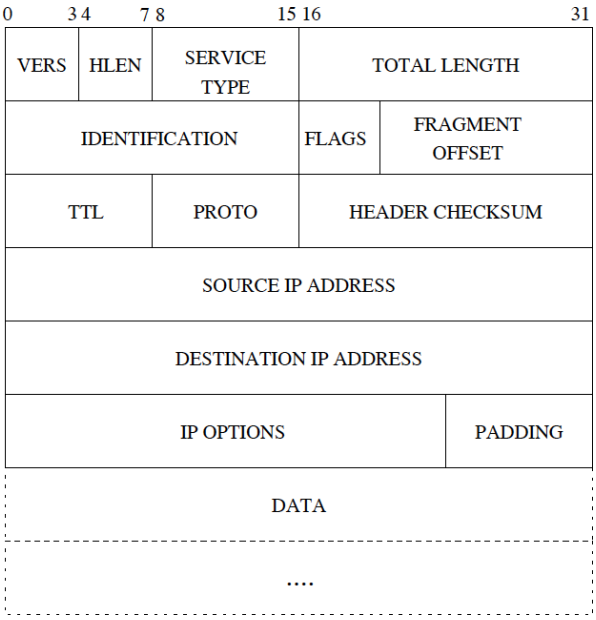
IP est l'acronyme de "*Internet Protocol*", est une famille de protocoles de communication au niveau 3 du modèle OSI (couche "Internet"). Son origine remonte à 1974. Deux versions sont actuellement utilisées : IPv4, le plus connu, avec des adresses sur 32 bits, est celui que nous étudierons dans ce cours. IPv6 permet de nouvelles possibilités (adresses sur 128 bits, qualité de service, ...) et est actuellement surtout utilisé dans les cœurs de réseaux.

IPv4 est décrit par la RFC 791 (de 1981).

Quelques caractéristiques en vrac du protocole IP :
- IP est le support de travail des protocoles de la couche de transport, TCP, UDP.
- IP ne donne aucune garantie quant au bon acheminement des données qu'il envoie. Il n'entretient aucun dialogue avec une autre couche IP distante, on dit aussi qu'il délivre les datagrammes (ou "paquets") "au mieux" (*best effort*).
- Chaque datagramme est géré indépendamment des autres datagrammes même au sein du transfert des octets d'un même fichier. Cela signifie que les datagrammes peuvent être mélangés, dupliqués, perdus ou altérés !

Ces problèmes ne sont pas détectés par IP et donc il ne peut en informer la couche de transport.

3.2 Structure de l'en-tête IPv4



`VERS` 4 bits qui spécifient la version du protocol IP (actuellement 4 ou 6) ;

`HLEN` 4 bits qui donnent la longueur de l'en-tête en "mots" de 4 octets. La taille standard de cette en-tête fait 5 "mots", la taille maximale fait : $15 \times 4 = 60$ octets.

`SERVICE TYPE` 8 bits. Actuellement utilisé pour le DSCP *Differenciated Service Code Point*, RFC 2474. Permet de spécifier le type le type de données transportées pour leur donner plus ou moins de priorité (exemple : voix sur IP).

`TOTAL LENGTH` donne la taille en octets du datagramme, en-tête plus données. S'il y fragmentation il s'agit de la taille du fragment (chaque datagramme est indépendant des autres). La taille des données est donc à calculer par soustraction de la taille de l'en-tête. 16 bits autorisent la valeur max 65535 octets. La limitation vient le plus souvent du support physique (MTU) qui impose une taille plus petite.

`IDENTIFICATION, FLAGS` **et** `FRAGMENT OFFSET` Ces mots sont prévus pour contrôler la fragmentation des datagrammes. Les données sont fragmentées car les datagrammes peuvent avoir à traverser des réseaux avec des MTU plus petits que celui du premier support physique employé.

`TTL` "*Time To Live*" 8 bits. Ce champ est un compteur décrémenté d'une unité à chaque passage dans un routeur. Couramment la valeur de départ est 32 ou même 64. Son objet est d'éviter la présence de paquets fantômes circulant indéfiniment. Si un routeur passe le compteur à zéro avant délivrance du datagramme, un message d'erreur est renvoyé à l'émetteur avec l'indication du routeur (via le protocole ICMP). Le paquet en lui-même est perdu.

`PROTOCOL` 8 bits pour identifier le format et le contenu des données, un peu comme le champ "type" d'une trame Ethernet. Il permet à IP d'adresser les données extraites à l'une ou l'autre des couches de transport. Valuers définies dans la RFC 790. Dans le cadre de ce cours, nous utiliserons essentiellement ICMP (valeur 1), UDP (valeur 17) et TCP (valeur 6).

`HEADER CHECKSUM` 16 bits pour s'assurer de l'intégrité de l'en-tête. A la réception de chaque paquet, la couche calcule cette valeur, si elle ne correspond pas à celle trouvée dans l'en-tête le datagramme est oublié ("discard") sans message d'erreur.

`SOURCE ADDRESS` Adresse IP de l'émetteur, à l'origine du datagramme.

`DESTINATION ADDRESS` Adresse IP du destinataire du datagramme.

`IP OPTIONS` 24 bits pour préciser des options de comportement des couches IP traversées et destinatrices. Les options les plus courantes concernent :
- enregistrements de routes ;
- enregistrements d'heure ;
- spécifications de route à suivre ;
- ...

Historiquement ces options ont été prévues dès le début mais leur implémentation n'a pas été terminée et la plupart des routeurs filtrants bloquent les datagrammes IP comportant des options.

`PADDING` Bits de remplissage (inutilisés donc) pour aligner sur 32 bits.

3.3 Adresses IPv4

Les adresses IPv4 sont représentées sur 32 bits et notées en décimal, 4 octets séparés par des points ; exemple : `192.168.0.13`

Chaque adresse peut être décomposée en une adresse de *réseau* (network, bits de poids forts) et une adresse de *machine* (host, bits de poids faible). La connaissance de la classe d'adresse ou du masque de réseau permet de savoir quelle taille (nombre de bits) a chaque partie.

Les bits de la partie machine ne peuvent valoir 0 (c'est alors l'adresse du réseau, comme `192.168.0.0`).

Si tous les bits de la partie machine sont à 1, cela indique une adresse de diffusion (sur ce réseau précis). Exemple : `192.168.0.255`

L'adresse `127.0.0.1` est réservée pour désigner la machine locale elle même. Ainsi, tout programme voulant envoyer un message à un autre progarmme sur la même machine peut utiliser les mécanisme de communication réseau en inquant l'adresse IP `127.0.0.1`.

Notons qu'une machine n'ayant pas (ou pas encore) d'adresse IP peut indiquer qu'elle a l'adresse `0.0.0.0`. C'est notamment le cas des client DHCP en attente de leur configuration.

3.3.1 Classes d'adresses

Classe	1er octet	Forme	Masque	Nb de réseaux	Nb de machines
A	1–126	N.H.H.H	255.0.0.0	$126 = 2^7 - 2$	$2^24 - 2$
B	128–191	N.N.H.H	255.255.0.0	$2^14 - 2$	$2^16 - 2$
C	192–223	N.N.N.H	255.255.255.0	$2^21 - 2$	$254 = 2^8 - 2$
D	224–239	réservé au multicast			
E	240–254	expérimental			

Dans la colonne « Forme », les octets N représentent la partie « réseau » de l'adresse, et les octets « H » la partie machine.

3.3.2 Plages d'adresses privées

Les adresses IP « privées » sont utilisées sur des réseaux locaux ou d'entreprise dans lesquels les machines n'ont pas besoin d'être atteinte depuis l'Internet public. Ces réseaux sont souvent connectés à l'Internet à travers un routeur à traduction d'adresse (NAT).

Les paquets utilisant des adresses privés ne sont pas routables par les routeurs d'Internet. En effet, plusieurs machines dans le monde peuvent utiliser la même adressse privée au même instant !

Les plages d'adresse réservées pour les réseaux privés sont spécifiées par la RFC 1918 :

Intervalle IP	Classe	Notation CIDR	Masque
10.0.0.0–10.255.255.255	A	10.0.0.0/8	255.0.0.0
172.16.0.0 –172.31.255.255	16 réseaux de classe B	172.16.0.0/12	255.240.0.0
192.168.0.0 –192.168.255.255	256 réseaux de classe C	192.168.x.0/24	255.255.255.0

3.3.3 Notation CIDR

La notation CIDR permet d'indiquer simplement le nombre de bits pour l'adresse réseau dans une adresse. Par exemple, `192.168.0.0/24` indique que ce réseau utilise 24 bits d'adresse pour la partie réseau et donc les 8 bits restants pour la partie machine.

EXERCICE 1 - Adresses MAC

1- Identifiez les fabricants des cartes réseau dont les adresses sont données ci-dessous :
– `00:16:6f:5f:f8:37`
– `00:07:cb:94:23:3d`
– `00:50:56:c0:00:08`
– `00:50:56:c0:00:01`
– `00:1a:30:0f:50:00`

EXERCICE 2 - Analyse de trame Ethernet

On considère la trame Ethernet suivante :

```
00:  0009 5b9a c494 0090 4b16 5d24 0800 4500
16:  0034 68db 0000 4011 74dd c0a8 0002 93d2
32:  088f 8002 03e1 cd9c f713 496f bd4c 8010
48:  0030 3e96 8d1c 0000 0101 080a 000e ffe0
64:  c4ef ...
```

qui est une trame Ethernet II encapsulant de l'IPv4.

1- Donnez les adresses MAC source et destination de ce message.

2- Isolez l'en-tête du datagramme IP véhiculé.
1. Quelles sont les adresses IP source et destination du message ?
2. Commentez, point par point, les différentes informations contenues dans l'en-tête IP ; en particulier, précisez le protocole des données transportées.
3. Précisez les ports source et destination sur le prototype des données tansportées
4. Quelle est la taille minimale du MTU du réseau traversé ?

EXERCICE 3 - ARP

1- Que se passe-t-il si une machine A veut envoyer un datagramme IP à la machine B dont A ne connait que l'adresse IP et pas l'adresse MAC ?

2- Que se passe-t-il si la machine B n'est pas sur le réseau physique de la machine A (donc reliée par un routeur par exemple) ?

3- Pourquoi est-ce judicieux de disposer d'un cache ARP sur chaque host ?

4- Décrire les champs d'un paquet ARP (RFC 826, voir aussi votre support de cours du module ARS3).

EXERCICE 4 - Décoder une trame Ethernet+ARP

Soit donné la trame ARP suivante, encapsulée dans une trame Ethernet (le CRC de la trame Ethernet n'est pas mentionné).

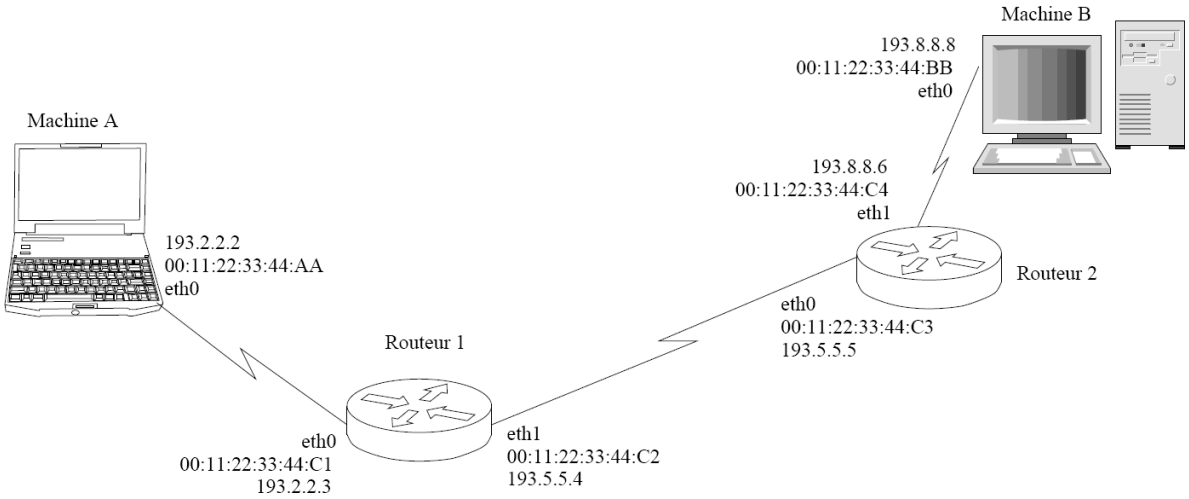
```
ADDR  HEX                                               ASCII
0000  FF FF FF FF FF FF 00 40 05 13 65 80 08 06 00 01   .......@..e.....
0010  08 00 06 04 00 01 00 40 05 13 65 80 80 DE 0C 01   .......@..e.....
0020  00 00 00 00 00 00 80 DE 0C 02 00 00 00 00 00 00   ................
0030  00 00 00 00 00 00 00 00 00 00 00 00               ............
```

1- Indiquer les champs de la trame ARP.

2- Indiquer la trame qui sera renvoyée par la machine ayant reconnu son adresse IP. Le code opération de la trame "ARP Reply" est 2.

EXERCICE 5 - Routage ARP

Considérez le réseau, représenté par la figure suivante, où la machine A souhaite envoyer un datagramme à la machine B. Les deux machines n'étant pas sur le même sous-réseau, le datagramme va donc devoir être routé via les deux routeurs R1 et R2.



1- Donnez les étapes successives nécessaires à cet acheminement, en précisant les adresses utilisées dans les en-têtes des trames Ethernet envoyées, ainsi que les requêtes ARP nécessairement effectuées.

2- Quel est l'état des tables ARP sur chaque machine une fois que B a reçu le datagramme (on suppose que ces tables étaient vierges au départ) ?

3- Dans l'état actuel, l'envoi d'un message de B vers A est-il possible ?

EXERCICE 6 - Classes d'adresses

Quelle est la classe des adresses suivantes ? Repérez certaines adresses particulières.

- 118.89.67.234
- 199.254.250.223
- 223.25.191.75
- 10.20.30.40
- 191.250.254.39
- 192.1.57.83
- 127.0.0.1
- 239.255.0.1
- 172.11.1.1
- 0.0.0.0
- 128.192.224.1
- 255.255.255.255

EXERCICE 7 - Si l'administrateur donne deux fois la même adresse IP à 2 machines différentes du réseau, que se passe-t-il ?

1. Les deux machines marchent très bien.
2. La première machine à obtenir l'adresse IP du réseau marche mais pas la deuxième.
3. Aucune machine ne marche.
4. Le débit est partagé entre les 2 machines.

EXERCICE 8 - Adresses réseaux

1- Un réseau a comme adresse 185.32.128.0 de masque 255.255.240.0. Quelle est l'adresse de diffusion (*broadcast*) ?

1. 185.32.255.255
2. 185.32.143.255
3. 185.32.159.25
4. 185.32.192.255

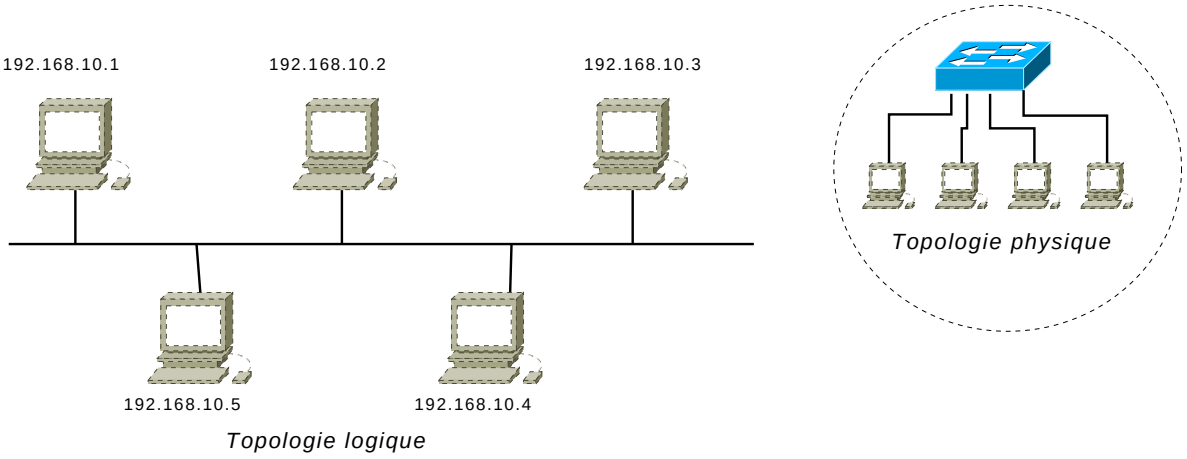
2- Une machine a comme adresse IP 150.56.188.80 et se trouve dans un réseau dont le masque est 255.255.240.0. Quelle est l'adresse du réseau ?

1. 150.56.0.0
2. 150.56.128.0
3. 150.56.176.0
4. 150.56.192.0

TD N° 3 - Routage statique

EXERCICE 1 - Adresses IP dans un réseau local

Soit le réseau local constitué de 5 machines ci-dessous auxquelles on a affecté une adresse IP (masque = 255.255.255.0) :



1- Quelles sont l'adresse et la classe du réseau ?

2- On souhaite rajouter 10 ordinateurs de plus à ce réseau. Donner une configuration d'adresses IP pour ces machines.

3- On veut encore connecter 250 ordinateurs de plus à ce réseau. Comment faire ?

4- Dans le réseau intranet résultant de la question précédente, la machine 1 souhaite communiquer avec la machine 4. Décrire tous les échanges au niveau Ethernet.

5- Même question si la machine 1 souhaite communiquer avec la dernière machine rajoutée au réseau.

6- On veut connecter à Internet ce réseau intranet.. Est-ce possible tel quel ? Sinon, pourquoi et quelles modifications apporter ?

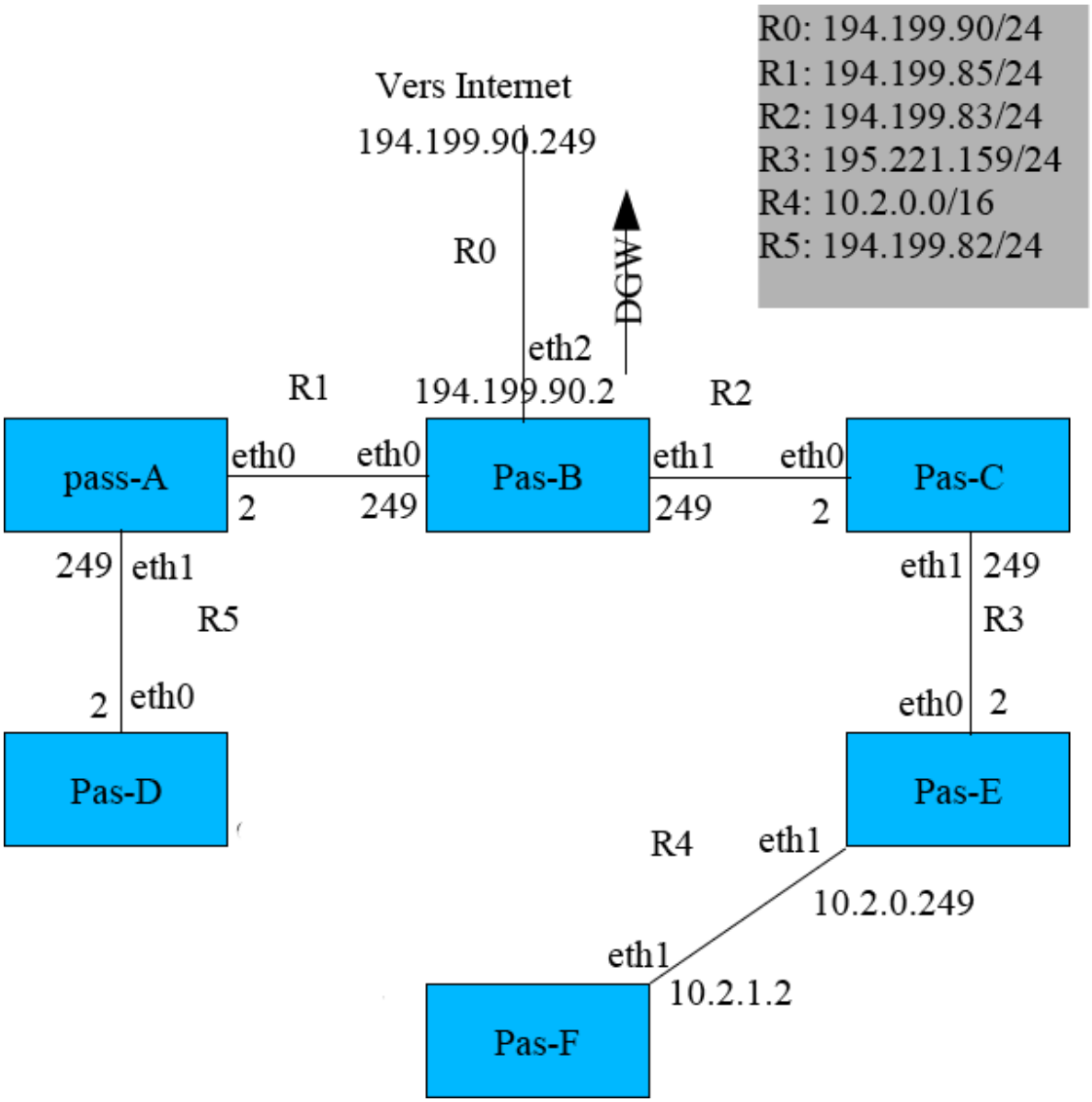
7- Quels sont les avantages et inconvénients de cette solution ?

EXERCICE 2 - Routage statique (*emprunté à P. Petit, Evry*)

On considère le réseau de la maquette ci-dessous. On vous demande de préciser la configuration du routage (routeur par défaut inclus) des machines du réseau de façon à ce que tous les postes communiquent entre eux et aient accès à internet.

La route par défaut de PAS-B vous est imposée.

La configuration du routeur vers internet (194.199.90.249) ne dépend pas de vous. On supposera qu'il est correctement configuré et qu'il connait vos réseaux internes.



EXERCICE 3 - Adressage IP, CIDR

Le standard CIDR (Classless Inter Domain Routing) étend la notion de réseaux IP de classes A, B, C. On indique l'adresse du réseau et le nombre de bits utilisé pour la partie réseau. les autres bits sont pour adresser les machines de ce réseau.

Ainsi, si l'adresse réseau est notée `192.168.255.48/25`, cela indique que les 25 bits de poids forts sont utilisé pour indiquer le réseau, et les 7 bits restants pour les machines.

1- Pour chacun des réseau suivant, indiquer le nombre max de machines que l'on peut adresser, et donner le masque de sous-réseau (subnet mask)
- 192.168.0.0/16
- 10.0.0.0/8
- 212.43.43.33/27

2- Sans calculatrice, donner un ordre de grandeur du nombre total d'adresses IPv4 publiques dans le monde.

3- En IPv6, les adresses sont sur 128 bits. Combien de machines peut-on connecter (avec IPv6 publiques) ? Combien cela représente-il d'adresses par être humain terrien ?

TD N° 4 - Web, CGI (révisions et préparation du TP)

EXERCICE 1 - *Formulaires Web*

Soit un formulaire HTML défini comme suit :

```
<form action="http://rt.org/krypto" method="get">
  <input type="text" name="valeur" value="" />
  <input type="submit" value="OK" />
</form>
```

L'utilisateur entre le texte "2008" dans le champ, puis clique sur le bouton OK.

1- Quelle est l'URL appelée par le navigateur ? Comment est envoyée la valeur saisie dans le champ de texte ? Quelle méthode HTTP est-elle invoquée ?

2- On remplace `method="get"` par `method="post"`. Quelle URL est-elle appelée ? Comment est envoyée la valeur saisie dans le champ de texte ?

EXERCICE 2 - *CGI*

1- Soit la requête `http://example.com/truc?x=3&y=3`. On suppose que `truc` est un script CGI. Expliquez précisément comment sont passés les paramètres de cette requête au programme CGI.

2- Que doit faire un programme CGI pour envoyer des données au client Web ?

3- Écrire en shell BASH un script CGI qui affiche une page Web (simplifiée, pas forcément validante) indiquant la date du serveur.

EXERCICE 3 - *Apache*

1- On a un serveur Apache s'exécutant sur notre machine UNIX d'adresse IP publique 1.2.3.4, en écoute sur le port 3000. S'agit-il d'un port UDP ou TCP ?

2- Indiquez l'URL à saisir dans un navigateur Web s'exécutant sur une machine connectée au réseau public pour afficher la page d'accueil de ce serveur Web.

3- Quel mécanisme simple d'Apache peut-on utiliser pour protéger les accès à un répertoire avec un mot de passe ?

4- Quelles sont les principales informations enregistrées par défaut par Apache dans son journal (log) ?

5- On suppose que le serveur Apache enregistre son journal dans le fichier `/var/log/apache2/access.log`.

Le site web ne comporte qu'une seule page (la page d'accueil), qui, outre du texte en XHTML, contient 2 images PNG.

Indiquer une méthode permettant de déterminer le nombre de fois où la page d'accueil a été visitée par des navigateurs de la famille "Mozilla".

6- Sous quelle identité UNIX les programmes CGI s'exécutent-ils ?

TP N° 1 - Utilisation d'UNIX

Ce TP s'effectuera *individuellement*, avec un PC sous Linux (image GTR-4, salles Q203 ou P202). Chaque étudiant rédigera un compte-rendu lisible sur papier remis à l'enseignant en fin de séance.

Objectifs

- Prise en main d'un système Linux ;
- commandes de base du shell ;
- édition et compilation d'un programme C.

Attention, UNIX fait la différence entre les majuscules et les minuscules. La plupart des commandes doivent s'écrire en minuscules.
Séparez toujours la commande de ses arguments par un ou plusieurs espaces (par exemple, écrire `ls -l` et non pas `ls-l`).

EXERCICE 1 - Commandes de base

1- Révisez l'utilisation des commandes `cd`, `mkdir`, `ls`, `rmdir`, `rm`, `cp`, `man`, `date`, `pwd`, `mv`, `echo`.

Sur votre compte-rendu, indiquez en une phrase ce que fait chaque commande et donnez un exemple d'utilisation.

2- *Commande ls*

En utilisant la commande ls et ses différentes options (voir `man ls`), visualisez le contenu de votre répertoire courant de la façon suivante :

1. Liste simple.
2. Liste montrant les fichiers cachés (ceux dont le nom commence par "."). On remarquera la présence des 2 fichiers "." et "..".
3. Liste avec descriptif complet de chaque référence (droits, nombres de liens, dates, taille user group ...).
4. Liste avec descriptif complet et avec un format plus compréhensible concernant la taille des fichiers.
5. Liste récursive (descend dans les sous-répertoires).
6. Liste par ordre chronologique (la commande "touch" peut servir à changer la date de modification d'un fichier).
7. Liste par date d'accès au lieu de la date de création. Pour constater un changement, utiliser la commande cat "nom de fichier" pour modifier la date du dernier accès.
8. Liste simple du contenu avec spécification du type de fichier (répertoire `/`, lien symbolique `@`, exécutable `*`).

3- *Commande man*

On peut chercher un mot clef interactivement lors de la visualisation du manuel d'une commande (qui se fait en réalité par l'intermédiaire du programme less).

La recherche est lancé en appuyant sur la touche `/` (voir le manuel de less pour plus de détails).

– Chercher dans le manuel de less le mot "pattern".

EXERCICE 2 - Compilation programme C

1- Créez un répertoire `TP1`. Dans ce répertoire, créez un fichier `hello.c` qui affiche "Bonjour toto" à l'écran (vous pouvez remplacer toto par votre nom). Créez un fichier `Makefile` qui vous permette de compiler votre programme, avec les options `-g -Wall -pedantic` (utiliser `CFLAGS`, voir résumé de cours).

2- Tester votre programme `hello`. Comment le lancer pour que le message soit ajouté au fichier `salutations` de votre répertoire de connexion au lieu d'être affiché à l'écran ?

3- Écrire un second programme, `un.c`, qui permette de créer un fichier texte nommé `fichier.txt` contenant la ligne "1".

Indications : revoir les fonctions `fopen()` et `fprintf()`.

4- Modifier votre programme sous le nom `creation.c` pour qu'il crée N fichiers nommés `f1.txt, f2.txt, ..., fN.txt` contenant chacun un nombre $(1, 2, 3, \ldots, N)$, N étant spécifié sur la ligne de commande (voir votre support de cours).

Indications : revoir les fonctions `atoi()` et `sprintf()`.

5- A l'aide du programme précédent, créez 1000 fichiers dans votre répertoire courant. Avec une seule commande shell, supprimez les fichiers dont les numéros sont entre 100 et 199.

TP N° 2 - Configuration réseau, services, SSH

Ce TP s'effectuera *individuellement*, avec un PC sous Linux (image GTR-4, salles Q203 et P202). Chaque étudiant rédigera un compte-rendu lisible remis à l'enseignant en fin de séance.

Objectifs

- Mise en réseau d'un client UNIX (révision) ;
- notion de "service" ;
- utilisation de la commande SSH.

Rappel : commandes de bases pour le réseau :
- `lspci` : affiche la liste des périphériques connectés au bus PCI.
- `ethtool <interface>` ou `mii-tool` : affiche l'état de l'interface (config. Ethernet, statut) ;
- `ifconfig` : associe une adresse IP à une interface réseau (`eth0` ou `eth1`) ;
- `dhclient <interface>` configure une interface réseau (`eth0` ou `eth1`) à l'aide du protocole DHCP ;
- `ping <ip>` utilise le protocole ICMP pour vérifier une conenxion (requête/écho) ;
- `wireshark` est un analyseur de trames ;
- l'adresse du ou des serveur(s) de noms (DNS) est dans `/etc/resolv.conf`.

EXERCICE 1 - *Configuration et tests IP de base*

1- Combien votre PC possède-t-il d'interfaces réseau ?

2- Quel est le nom (`eth0` ou `eth1`) de l'interface connectée au réseau extérieur, et de celle connectée au réseau de la salle ?

3- Configurer l'interface reliée au réseau extérieur à l'aide du protocole DHCP.

1. Quelle adresse IP obtenez-vous ?
2. Quelle est l'adresse IP du serveur DHCP ?
3. Le serveur DHCP répond-il au ping ? Si oui, avec quel délai d'aller/retour ?
4. Quel serveur de nom (DNS) utilisez-vous ?
5. Quelle est l'adresse IP de `www.iutv.univ-paris13.fr` ? (utilisez ping)
6. Quel délais d'A/R vers `www.iutv.univ-paris13.fr` ?
7. Mêmes questions pour `www.google.com`.

4- Configurer l'interface reliée au réseau *intérieur* (celui de la salle) avec une adresse IP fixe de la forme `10.0.0.N`, où N est le numéro de votre machine (écrit sur la prise réseau).

1. La connexion vers l'extérieur est-elle toujours fonctionnelle (indiquez les tests effectués).

2. La machine de votre voisin répond-elle au ping ? Quelle est son IP et quel délai d'A/R mesurez-vous ?

5- Pour que la configuration réseau obtenue dans les questions précédentes soit automatiquement appliquée à chaque redémarrage, il faut modifier des fichiers de configuration.

L'état désiré de chaque interface est spécifié dans le fichier

```
/etc/sysconfig/network-scripts/ifcfg-ethX
```

où (`X` est le numero de l'interface).

Exemple pour une IP fixe sur eth4 :

```
DEVICE=eth4
BOOTPROTO=static
IPADDR=10.10.0.11
NETMASK=255.255.0.0
ONBOOT=yes
```

Chercher (sur le web) ce que doit contenir le fichier pour une interface en DHCP (votre système Linux est une version de Mandriva), puis créer les fichiers nécessaires.

Pour appliquer la configuration (comme si la machine démarrait), faire :

```
/etc/init.d/network restart
```

Tester l'état obtenu (avec `ifconfig`).

EXERCICE 2 - Utilisation de SSH

Le logiciel SSH (serveur et client) est normalement déjà installé sur votre machine.

Le processus serveur s'appelle "sshd". Pour savoir s'il est lancé, utiliser `ps aux | grep sshd`. Pour le lancer, utiliser

```
/etc/init.d/sshd start
```

Stopper le serveur avec `/etc/init.d/sshd stop` , et éditez le fichier de configuration `/etc/ssh/sshd_config`.

Les commandes `man ssh` et `man sshd_config` devraient vous fournir de la documentation.

1- L'utilisateur `root` peut-il se connecter sur votre machine via `ssh` ?

2- Peut-on se connecter via `ssh` sur votre machine en donnant un mot de passe (et non une clé) ?

3- Les clés cryptographiques sont stockées sous `~/.ssh`. L'identification par clés asymétriques met en place 3 fichiers :

– `~/.ssh/id_rsa` : la clé privée qui permet de décrypter l'information et de s'identifier (en fournissant la "passphrase")
– `~/.ssh/authorized_keys` : la liste des clés publiques autorisées à accéder au compte de l'utilisateur.
– `~/.ssh/id_rsa.pub` : la clé publique (qui sert à crypter). C'est le fichier à transmettre à quiconque nous autorise à utiliser son compte, le contenu étant ajouté dans son fichier personnel `authorized_keys`.

A l'aide de la commande

```
ssh-keygen -t rsa
```

créez (en tant qu'utilisateur "etudiant") une clé.

Fournissez la clé publique à votre voisin.

1. Quel moyen utilisez-vous pour la transmettre ?
2. Où doit-il la copier pour que vous puissiez vous connecter sans mot de passe sur sa machine ?
3. Qu'observez vous dans le fichier journal `/var/log/auth.log` lorsque la connexion s'établit ?

TP N° 3 - Simulation avec Marionnet - Ethernet, IP

Ce TP s'effectuera *individuellement*, avec un PC sous Linux (**image CRIT standard**) et le logiciel Marionnet (`http://www.marionnet.org`). Chaque étudiant rédigera un compte-rendu lisible remis à l'enseignant en fin de séance (par email).

Objectifs

- Mise en réseau d'un réseau IP (simulation sous marionnet) ;
- routage statique ;

Ce TP utilise le logiciel **marionnet** développé à Paris 13, voir www.marionnet.org.

Sur les machines du CRIT, marionnet se trouve dans l'image Linux dans le menu `Applications / Education / Marionnet`.

TAPEZ les réponses aux questions dans un fichier texte que vous enverrez par mail à l'adresse que vous donnera votre enseignant. Vous utiliserez la commande `man` pour chercher les options des commandes qui vous seront nécessaires.

Truc utile : pour copier/coller depuis Marionnet vers l'extérieur, sélectionner le texte puis cliquer sur le bouton souris du milieu dans le logiciel hors marionnet.

EXERCICE 1 - *Introduction : création d'un nouveau projet*

Créez un nouveau projet et y ajouter deux machines virtuelles (m1 et m2 avec les valeurs par défaut) non reliées et lancez-les (bouton "tout démarrer"). Configurez les adresses IP de ces deux machines par la commande :

`ifconfig <interface> <adresseIP> [netmask <masque réseau>]`

L'interface est `eth0` ou `eth1` etc. Les paramètres entre crochets sont optionnels. Si le netmask n'est pas fourni, le système le calcule automatiquement.

Vous pouvez consulter les tables de routage par la commande `route`. Pour ajouter une entrée dans la table de routage :

`route add -net <destination> [netmask <masque réseau>] [gw <adresseIP>] <interface>`

Pour ajouter une route par défaut la syntaxe est simplifiée :

`route add default gw <adresseIP>`

Enfin, pour afficher une table de routage, utiliser :

`route -n`

(l'option `-n` permet de désactiver les appels DNS cherchant à convertir les adresses IP en noms ; elle existe aussi pour la commande `traceroute`).

EXERCICE 2 - *Relier les machines*

1- Essayez un câble droit et un câble croisé, comment vérifier que la liaison fonctionne ?

2- Consultez le cache ARP avec la commande `arp -an`. Que contient-il ?

Reliez maintenant ces machines par un *hub* (concentrateur) ; pensez à démarrer ce *hub* !

EXERCICE 3 - *Comparaison HUB et Switch, étude de ARP*

Ajoutez une 3ème machine que vous appellerez `Espion`. Démarrez-la et lancer dessus l'analyseur de trames `wireshark` (patientez son lancement peut être lent !). Une fois wireshark lancé, faites un ping entre m1 et m2.

1- Que constatez-vous ? Regardez en particulier les diodes du HUB.

2- Pourquoi voit-on des paquets ARP avant les paquets ICMP ?

Supprimez les entrées dans les caches ARP de m2 par `arp -d.` Changez l'entrée dynamique du cache ARP de m1 par une entrée statique équivalente (voir `arp -s`).

Testez à nouveau ping en lançant l'analyseur au préalable.

3- Que constatez-vous ?

4- Comment positionner une entrée dynamique dans le cache ARP ?

Supprimez les entrées du cache ARP et relancez le `ping` puis regardez à nouveau le cache ARP...

5- Qu'y a-t-il dans la partie donnée des paquets ICMP ?

Arrêtez le ping, relancez-le avec une option qui permette de changer le contenu des paquets ICMP par le motif (pattern) `0xBA` répété (utile pour détecter des erreurs de transmissions dépendantes des données).

6- Voyez-vous ces paquets dans l'analyseur ? Pourquoi ?

7- Consultez le cache ARP de chaque machine. Pouvez-vous y trouver l'adresse MAC de Espion ? Pourquoi ?

8- Changez maintenant le HUB par un *Switch* (Commutateur niveau 2), que constatez-vous du côté des diodes ? Pourquoi ? Et au niveau de l'analyseur ?

9- En fait, si vous faites très attention, les diodes ne clignotent pas de la même façon au début et à la suite du ping... Pourquoi ?

Si vous n'avez pas pu voir le phénomène, il faut essayer de donner à Espion une adresse IP dans le même réseau et tester un ping à partir de lui...

EXERCICE 4 - Broadcast (diffusion)

Pour toute la suite il faut remplacer le *switch* par un *hub*.

Donnez à Espion une adresse IP sur le même réseau que M1 et M2. Sur M2 et Espion les broadcast sont refusés par défaut pour des raisons de sécurité. Il faut changer cela par :

```
sysctl -w net.ipv4.icmp_echo_ignore_broadcasts=0
```

(`sysctl` est utilisé pour modifier les paramètres du noyau en cours d'exécution). Depuis M1, lancer un ping en broadcast sur le réseau.

1- Quelle est la commande à saisir ?

EXERCICE 5 - *Fragmentation IP*

Les cartes réseaux Ethernet sont réglées par défaut pour respecter Ethernet, c'est à dire une MTU de 1500 octets.

1- Que veut dire MTU ? Vérifiez sa valeur actuelle sur M1 et M2.

Avec une option spéciale de ping on peut faire des paquets presque aussi gros que l'on veut.

2- Quelle est cette option ?

Utilisez-la pour faire un ping de 1500 octets entre M1 et M2.

3- Pourquoi il y a-t-il alors de la fragmentation ?

Avec l'analyseur consultez la taille des fragments en regardant la valeur offset (l'analyseur l'affiche en octets et non en nombre de mots de 8 octets).

4- Quel est la taille de chaque fragment fragment ? En déduire la taille de l'en-tête IP.

5- On peut changer le MTU de M1 et M2 avec la commande `ifconfig`. Quelle est la syntaxe à utiliser ?

On envoie maintenant des paquets ICMP de 500 octets. En jouant sur le réglage du MTU, déterminez expérimentalement la valeur minimale du MTU minimum pour qu'il n'y ait pas de fragmentation.

EXERCICE 6 - *Routage statique*

Nous allons ici utiliser un PC avec 2 cartes réseaux comme routeur.

Arrêtez la machine espion et supprimez là. Créez une nouvelle que vous appellerez « routeur », avec **2 cartes réseaux**. Démarrez cette machine et vérifiez que vous avec 2 cartes avec ifconfig :

1- Quelles sont les adresses MAC des cartes réseaux de la machine « routeur » ?

Reliez M1 à `eth0` de routeur et M2 à `eth1` de routeur.

Testez par ping les liens M1→ Routeur et M2→Routeur. Le routage n'est pas activé par défaut sur une machine normale (sur les vrais routeurs c'est différent). Nous allons l'activer par :

```
sysctl -w net.ipv4.ip_forward=1
```

(Pour désactiver le routage, mettre 0 à la place de 1 !)

2- Maintenant tester le ping de M1 à M2, que manque t-il encore ?

Ajoutez ce qu'il faut jusqu'à ce que le ping marche dans les deux sens. Note : il n'est pas nécessaire d'avoir une route par défaut sur le routeur...

Lancez wireshark sur le routeur, et observer les trames échangées.

EXERCICE 7 - *Routes et boucles*

Depuis M1 faites `traceroute -n` (ou `tracepath`) <adresseIP> où *adresseIP* est l'adresse de M2.

1- Donnez la route suivie par le paquet.

2- Si vous disposez de la commande tracepath, donner le MTU maximal sur le chemin du M1 à M2.

3- Sur le routeur ajoutez une route par défaut vers m1.

Testez par `traceroute` adresseIPbidon à partir de M2 avec wireshark lancé sur le routeur.

4- Notez sur votre compte-rendu la commande que vous avez entrée et le résultat, essayer d'expliquer ce comportement. Donnez la route suivie par le paquet. Pour un paquet ICMP émis par la machine m2, combien de paquets sont émis par m1 ?

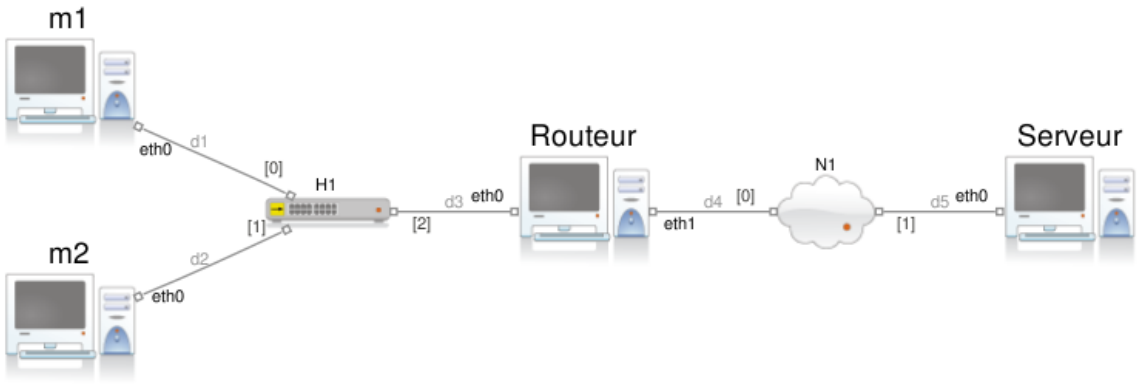
TP N° 4 - Routage, filtrage, NAT

Ce TP s'effectuera *individuellement*, avec un PC sous Linux (image CRIT standard) et le logiciel Marionnet (`http://www.marionnet.org`). Chaque étudiant rédigera un compte-rendu lisible remis à l'enseignant en fin de séance.

Objectifs

- Mise en réseau d'un réseau IP (simulation sous marionnet) ;
- routage statique ;
- filtrage et NAT avec iptables ;

EXERCICE 1 - *Mise en place du réseau*



1- Mettre en place un réseau selon la figure ci-dessus. La partie gauche simule un réseau local (2 machines), connecté via un routeur à Internet. Le routeur est ici un PC standard avec deux cartes réseau (eth0 et eth1).

2- Démarrer ce réseau (bouton "tout démarrer") et attribuer à chaque machine une adresse IP, selon le plan suivant :
- Réseau local (m1, m2, eth0 du routeur) : `192.168.0.0/24`
- Adresse externe du routeur : `10.0.0.1`
- Adresse du serveur : `10.0.0.100`

Vérifier (à l'aide de ping) que toutes les machines communiquent. Les machines du réseau local peuvent-elles communiquer avec le serveur ? Quel message observe-t-on ?

3- Là où cela est nécessaire, indiquer la route par défaut. Utiliser la commande

```
route add default gw <ip_de_la_passerelle>
```

Les machines du réseau local peuvent-elles communiquer avec le serveur ? Quel message observe-t-on ?

4- Sur le routeur, activer le routage des paquet à l'aide de la commande :

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

Les machines du réseau local peuvent-elles communiquer avec le serveur ? Quel message observe-t-on ?

5- Lancer l'analyseur wireshark sur la machine "Serveur". Quelle est l'adresse d'origine des paquets ICMP émis par le ping ?

6- À l'aide de la commande `traceroute -n 10.0.0.100`, donner la route suivie par les paquets émis de m1 vers Serveur.

EXERCICE 2 - *NAT simple*

Pour activer la traduction d'adresses (SNAT) sur le routeur, utiliser les commandes suivantes :

```
iptables --flush
iptables --table nat --flush
iptables -t nat -A POSTROUTING -o eth1 -j MASQUERADE
iptables -A FORWARD -i eth1 -o eth0 -m state --state RELATED,ESTABLISHED -j ACCEPT
iptables -A FORWARD -i eth0 -o eth1 -j ACCEPT
```

Placer ces commandes dans un fichier et copier/coller ou exécuter le script.

1- Si m1 envoie un ping à Serveur, que se passe-t-il ? Donner l'adresse destination et l'adresse source du paquet reçu par Serveur.

2- Même question dans l'autre sens. Si Serveur envoie un ping à m1, que reçoit m1 ? Pourquoi ?

3- Lancer un serveur web sur le Serveur :

```
/etc/init.d/apache2 start
```

Ce serveur est préconfiguré avec une page minimale à la racine qui affiche "It works !".

Le fichier journal (log) est `/var/log/apache2/access.log`.

Sur un client (m1), on utilise la commande `lynx` comme client web :

```
lynx http://10.0.0.100/
```

1. Qu'observez vous dans le journal du serveur web lors de l'accès avec lynx ?
2. De quelle adresse IP semble provenir la requête ?
3. À l'aide de wireshark lancé sur le routeur et/ou sur les machines d'extrémité, donner (pour un accès web via lynx depuis un client) :

– l'IP source du paquet émis par le client :
– le port TCP source du paquet émis par le client :
– l'IP destination du paquet émis par le client :
– le port TCP destination du paquet émis par le client :
puis
– l'IP source du paquet reçu par le serveur :
– le port TCP source du paquet reçu par le serveur :
– l'IP destination du paquet reçu par le serveur :
– le port TCP destination du paquet reçu par le serveur :

EXERCICE 3 - *NAT et filtrage*

Le script ci-dessous est un exemple de configuration iptables. Attention : cette configuration est extrêmement simple et devrait être complétée avant une utilisation sérieuse !

Le script présenté ci-dessous configure iptables pour :
– laisser sortir tous les paquets émis localement ;
– faire suivre (NAT) les paquets du réseau interne vers le réseau public ;
– refuser tous les paquets arrivant de l'extérieur.

```
#!/bin/sh
# -*- Mode: sh -*-
# Configuration IPTABLES pour NAT/firewall SIMPLE(iste)
# E. Viennet, Fev 2009 pour GEII

## Interface exterieure  (reseau "public" ou "untrusted")
EXTIF="eth1"

## Interface interne (reseau de confiance)
INTIF="eth0"

# Reinitialise iptable:
iptables -F
iptables -F -t mangle
iptables -F -t nat
iptables -X
iptables -X -t mangle
iptables -X -t nat

# Defauts:
iptables -P INPUT DROP
iptables -P OUTPUT ACCEPT
iptables -P FORWARD DROP

# Active kernel IP forwarding
echo 1 > /proc/sys/net/ipv4/ip_forward

# --- REGLES

# Forwarde *tous* les paquets de l'interne vers l'exterieur
iptables -A FORWARD -i $INTIF -o $EXTIF -j ACCEPT

# Forwarde les paquets faisant partie de connexions existantes de ext -> int
iptables -A FORWARD -i $EXTIF -o $INTIF -m state --state ESTABLISHED,RELATED -j ACCEPT

# Active NAT (MASQUERADE) sur EXTIF
iptables -A POSTROUTING -t nat -o $EXTIF -j MASQUERADE
```

1- Copier ce script dans un fichier “firewall.sh” l'exécuter. Toutes les machines sont-elles toujours en contact (via ping) ?

Expliquer pourquoi le Serveur ne peux plus “pinguer” les machines m1 et m2 (indiquer la règle iptable en cause).

2- Ajouter une règle (juste après le commentaire “REGLES” dans le script) :

```
iptables -A FORWARD -i $INTIF -s IP_DE_m2 -p tcp --destination-port 80 -j DROP
```

(en remplaçant `IP_DE_m2` par l'adresse IP de m2).

Que fait cette règle ?

3- Modifier le firewall pour ne laisser passer que le trafic vers le serveur web (tcp/80) et rien d'autre (même pas les pings).

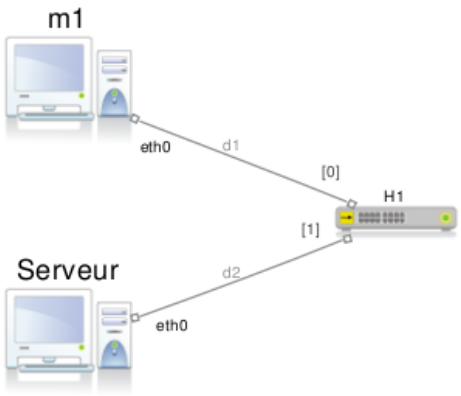
TP N° 5 - Configuration serveur web

Ce TP s'effectuera *individuellement*, avec un PC sous Linux (image CRIT standard) et le logiciel Marionnet (`http://www.marionnet.org`). Chaque étudiant rédigera un compte-rendu lisible remis à l'enseignant en fin de séance.

Objectifs

– Configuration de base d'un serveur web Apache.

EXERCICE 1 - *Installation et configuration de base*



1- Mettre en place un réseau Marionnet selon la figure ci-dessus, selon le plan d'adressage privé de votre choix (indiquer le dans votre compte-rendu).

2- Editer les fichiers `/etc/hosts` de chaque machine pour y indiquer les noms et adresses IP des deux machines.

Vérifier que `ping Serveur` fonctionne depuis `m1` et `ping m1` depuis `Serveur`.

3- Le fichier de configuration principal du serveur est `/etc/apache2/apache2.conf`

Éditer ce fichier et répondre aux questions suivantes, qui *se réfèrent toutes à la configuration par défaut* :

1. Il y a-t-il d'autres fichiers de configuration chargés (voir directives `Include`) depuis `apache2.conf` ? Où sont-ils placés ? Donner leur liste.
2. Quel directive spécifie le ou les ports TCP sur lesquels écouter ? Sur quels ports écoute ce serveur ?
3. Quelle est la directive chargeant un module ?

4. Sous quelle identité unix (utilisateur et groupe) le serveur va-t-il s'exécuter ? Donner les UID et GID correspondants.
5. Quel est le répertoire racine pour les documents (pages) servis ?
6. Quelle page Apache renvoie-t-il lorsque l'URL demandée correspond à un répertoire ?
7. Comment sont traitées les URL de la forme `http://serveur/cgi-bin/toto` ?
8. Quels sont les fichiers de logs générés ? Où sont-ils placés ? Quel est leur format et comment est-il contrôlé ?

4- Créer une page HTML minimale et la placer à la racine de l'arborescence servie. Lancer le serveur web (commande `/etc/init.d/apache2 start`). Dans quel fichier de log peut-on constater le démarrage ?

EXERCICE 2 - Protection des accès

1- Créez un répertoire `secret` dans l'arborescence web et placez y une page HTML. Configurez Apache afin que vous ne puissiez accéder au répertoire `secret` que depuis le serveur lui même.

Quel est le code renvoyé par Apache lorsqu'on tente d'accéder à ce répertoire depuis une autre machine ? Qu'observe-t-on dans les logs ?

2- (extraite de http://ww2.ac-creteil.fr/reseaux/systemes/linux/lamp2/TP-apache2-configuration.html)

Soit à protéger l'accès au sous-site privé d'un établissement, supposons qu'il s'agit du sous-répertoire `/var/www/html/prive`.

Il ne devra être accessible qu'à un ensemble limité de comptes Apache (et non Linux) à créer.

Une requête s'adressant à ce répertoire protégé provoquera l'affichage d'une boite de dialogue par laquelle l'utilisateur devra s'authentifier (nom et mot de passe).

Principe
La clause `AccessFileName .htaccess` fixe globalement le nom des fichiers de paramètres locaux.
Un fichier de ce nom, présent dans un répertoire, peut contrôler complètement les accès à ce répertoire, pourvu que la permission soit accordée par la directive `AllowOverride AuthConfig` ou `AllowOverride All`
Alors, les directives contenues dans ce fichier seront systématiquement respectées avant toute autorisation. Voici les directives usuelles et leur signification :
`AuthType basic`, type d'authentification communément adopté, fait hélas circuler les mots de passe en clair ;
`AuthName texte`, affichera le texte comme invite dans la boite de dialogue ;
`AuthUserFile chemin/fichier`, précise le fichier qui contient les comptes et mots de passe des utilisateurs ayant droit d'accès ;
`Require valid-user | liste-noms` tous, ou seulement les comptes énumérés dans la liste, auront accès au répertoire.

1. Créer le répertoire `/var/www/html/prive`, y placer quelques pages HTML.

Tester leur accessibilité pour tous. Sinon penser à modifier les permissions Linux sur ces fichiers.

2. Créer dans ce répertoire à protéger le fichier `.htaccess`. En voici une écriture standard :

```
AuthUserFile /etc/httpd/users
AuthGroupFile /dev/null
AuthName "Accès privé"
AuthType Basic
# autres clauses
# AuthGroupFile /etc/httpd/conf/groups

<limit GET>
# ATTENTION : GET en majuscules !
require valid-user
# require user toto dupond
# require group profs
</limit>
```

3. Dans ces conditions où se trouvera le fichier d'authentification ?
4. Créer un premier compte Apache avec la commande `htpasswd` .

```
cd /etc/httpd/
htpasswd -c users admin
  ---> mot de passe demandé (admin), puis confirmé.
```

5. Examiner le fichier `/etc/httpd/users`

   L'utilitaire `htpasswd` a créé (option `-c`) le fichier `users` dans le répertoire courant, ici `/etc/httpd`, et y a enregistré `admin` avec son mot de passe crypté.
6. Ajouter un second compte, `toto`, puis d'autres

```
htpasswd  users toto
 ---> mot de passe demandé, puis confirmé
```

7. Tester l'accès au répertoire `http://serveur/prive`. Pourquoi la protection ne semble-t-elle pas fonctionner ?

   (remarque : `service httpd reload` permet de prendre en compte les changements de configuration)
8. Rechercher dans le fichier de configuration la section `<Directory  /var/www/html>` qui fixe des directives par défaut pour le site principal. Par sécurité mettre si nécessaire la clause AllowOverride None
9. Ajouter une directive concernant le répertoire privé

```
<Directory  /var/www/html/prive>
  AllowOverride ...
  Options -Indexes
.........
<Directory>
```

10. Retester normalement avec succès ! N'oubliez pas de relancer le *navigateur* quand on change de compte.
11. Observez le trafic HTTP avec `ethereal` lors d'un accès au répertoire privé depuis une machine distante. Où se trouve le mot de passe ? Est-il lisible ?

EXERCICE 3 - Serveur “virtuel”

Lorsqu'une machine doit héberger plusieurs sites webs différents et même si elle ne dispose que d'une adresse IP, on utilise la technique des “serveurs virtuels” (*Virtual Hosts*).

1- Donner plusieurs noms à votre machine (soit via `/etc/hosts` soit en modifiant le DNS de la salle s'il y en a un).

2- Configurez Apache, en vous inspirant de ces extraits (à adapter !) :

```
# Protection maximale de la racine de l'hote du serveur
<Directory />
    Options FollowSymLinks
    AllowOverride None
</Directory>

# Paramétrage du site web usuel accessible par http://serveur/
################################################################
<Directory "/var/www/html">
# Options possibles :"All", ou une combinaison de Indexes,Includes,FollowSymLinks,ExecCGI,M
    Options -Indexes FollowSymLinks

# Pour empecher l'action "outrepassante" des fichiers .htaccess dans les répertoires
# les paramètres possibles sont All, ou une combinaison qcq de Options,FileInfo,AuthConfig,

    AllowOverride None

# Pour controler les permissions d'accès au serveur
# pour des droits restrictifs interdire D'ABORD de partout, puis ENSUITE
# accorder à des machines particulières
    order deny,allow
    deny from all
    allow from localhost 10.0.0.0/255.255.255.0
</Directory>

# Paramétrage d'un site web virtuel accessible par http://toto.gtr.org/
#########################################################################
<Directory "/home/totoweb">
    Options Indexes FollowSymLinks
    order deny,allow
    deny from all
    allow from localhost 10.0.0.0/255.255.255.0
</Directory>
```

Site principal et sites virtuels :

```
#######################
# Hotes virtuels nommés
#######################
# le numéro ip de la machine
NameVirtualHost 10.0.0.5
```

```
# Le premier paragraphe décrit le site principal
################################################
<VirtualHost 10.0.0.5>
    DocumentRoot /var/www/html
    ServerName poste01.perp77.fr
</VirtualHost>

# serveur virtuel pointant dans une autre partition
#####################################################
<VirtualHost 10.0.0.5>
    DocumentRoot /home/totoweb
    ServerName toto.gtr.org
</VirtualHost>
```

3- Tester votre configuration.

4- Comment séparer les logs d'accès de chaque serveur virtuel ?

TP N° 6 - Serveur web : script CGI en bash et en C

Ce TP s'effectuera *individuellement*, avec un PC sous Linux (image CRIT standard) et le logiciel Marionnet (`http://www.marionnet.org`). Chaque étudiant rédigera un compte-rendu lisible remis à l'enseignant en fin de séance.

Objectifs

– Écriture de scripts CGI

Mettre en place sous *Marionet* une machine "client" et une machine "serveur" avec le serveur web Apache comme dans le TP précédent.

EXERCICE 1 - Script CGI simple en BASH

1- Écrire en shell BASH un script CGI nommé `date` qui renvoie la date et l'heure dans une page HTML.

Où placez-vous le script ? Faut-il configurer quelque chose ?

2- Modifier la configuration pour que toutes les pages suffixées par `.sh` soient considérées comme des CGI et exécutées.

(Note : cette façon de faire n'est pas recommandée sur un vrai serveur car l'exécution de CGI peut poser des problèmes de sécurité, et il vaut donc mieux les regrouper dans un répertoire bien surveillé).

3- Écrire un CGI qui permette d'afficher la liste des processus appartenant à un utilisateur donné.

`http://serveur/listeprocs.sh?user=toto` afficherait dans une page HTML la liste des processus de toto s'exécutant sur le serveur.

4- Écrire une page HTML avec un formulaire simple permettant la saisie du nom de l'utilisateur.

EXERCICE 2 - *CGI en langage C*

Sous linux, le répertoire `/proc` contient des "pseudo-fichiers" permettant d'obtenir (ou de modifier) des paramètres système.

Par exemple, le "fichier" `/proc/cpuinfo` donne des informations sur le (ou les) processeurs installés sur l'ordinateur, et le fichier `/proc/meminfo` des informations sur la mémoire centrale :

```
$ cat /proc/meminfo
MemTotal:     33014340 kB
MemFree:      18980920 kB
Buffers:        524556 kB
Cached:       11081752 kB
```

Ces fichiers ont tous la même structure : chaque ligne comporte un nom de paramètre, suivi d'un deux points, et d'une valeur.

Écrire un script CGI en langage C qui présente le contenu du fichier `/proc/meminfo` sous forme d'une table.

On rappelle à la structure d'une page XHTML contenant une table :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Exemple de page xhtml</title>
</head>

<body>
<table>
  <tr>
    <th>Titre colonne 1</th>
    <th>Titre colonne 2</th>
  </tr>
  <tr>
    <td>case 1</td>
    <td>case 2</td>
  </tr>
</table>
</body></html>
```