Introduction à UNIX et LINUX

Présentation :

UNIX est un système d'exploitation,
c'est-à-dire la couche logicielle qui se situe entre la machine matérielle et le logiciel d'application.
C'est un système d'exploitation "ouvert", qui fonctionne sur une multitude de machines.
Il est indépendant de tout constructeur, de toute marque, de tout processeur.

LINUX est une version libre de Unix, fonctionant sur PC, développée à l'origine par Linus Torvalds, un étudiant finlandais, et "amélioré" par une multitude de collaborateurs bénévoles par l'Internet.
Unix comporte

- un noyau (kernel) formant le coeur du système
- des programmes (utilitaires) grâce auxquels ont peut formatter une disquette, copier des fichiers, etc...
- des fichiers de configuration (fichiers système) qui permettent de personnaliser le système en fonction des utilisateurs

(comparaison avec DOS :
kernel = fichiers système cachés (IO.SYS, MSDOS.SYS, COMMAND.COM)
utilitaires = command.com et .com et .exe du DOS,
fichiers système = config.sys)

UNIX est multi-utilisateur et multi-tâche

- **multi-utilisateur :** plusieurs personnes peuvent travailler en même temps sur une machine Unix (en général, on y connecte plusieurs terminaux)
- **multi-tâche :** un utilisateur peut lancer un programme alors qu'un autre tourne déjà.

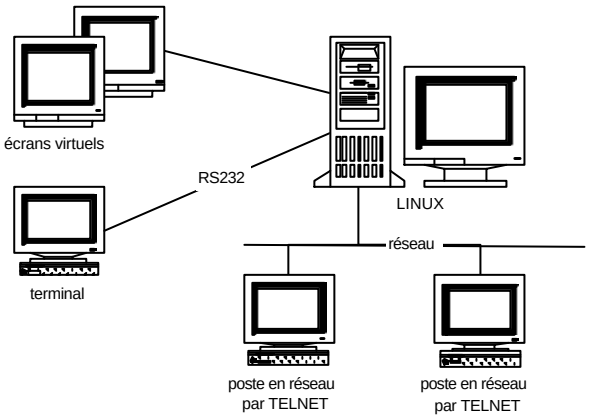
(à titre de comparaison, DOS est mono-tâche, mono-utilisateur,
Windows95 est multi-tâche, mono-utilisateur)

Pour cela, UNIX utilise

- **le "temps partagé" (time sharing) :** chaque programme en mémoire utilisera le processeur pendant un certain temps. Les tranches de temps sont très courtes et chacun croit être seul à travailler sur le système.

- **la préemption :** le système peut interrompre n'importe quel programme pour donner la main à un autre programme

- **la réentrance :** quand plusieurs personnes utilisent le même programme, celui-ci n'est chargé qu'une fois en mémoire. Chaque utilisateur dispose simplement en mémoire d'un environnement de travail spécifique.



Une machine UNIX accepte plusieurs sessions, par écran virtuel (taper F1, F2,…) ou par liaison série (COM1, COM2, …) ou par réseau (grâce à TELNET).

Introduction à UNIX et LINUX

Commandes UNIX (quelques commandes de base)

- Unix fait la différence entre majuscules et minuscules, il faut donc taper les commandes en minuscule !
- pour interrompre une commande en cours, taper SUPPR, ou Ctrl-C, ou Ctrl-D (c'est selon le système)

1. connexion :

login jim
password : ...

déconnexion : **logout** ou exit ou ctrl-D

si connexion en admin (root), alors prompt #, sinon prompt $
pour changer le mot de passe : **passwd**
pour bloquer le terminal pendant une absence : **lock** puis mot de passe temporaire
au retour, taper le mot de passe temporaire

2. environnement de travail :

logname	affiche le nom user
id	n° et nom user et groupe
finger jim	caractéristiques de user jim
	$ finger (sans paramètre) : caract. de tous les user connectés
whoami	nom, terminal et date de connexion
pwd	répertoire courant
tty	nom du fichier correspondant à la ligne du terminal
stty	paramètres de la ligne (ne pas modifier sans bien les connaître)
echo texte	affiche le texte
clear	efface l'écran
date	affiche date
cal	affiche le calendrier du mois courant
	$ cal mois année : calendrier du mois demandé
who	liste des user connectés
	-a donne tous les renseignements
	-b indique date du dernier démarrage système
	-H titre des colonnes
	-l liste des lignes sans nom de user
	-q nom user et nombre
	-r niveau d'exécution du système
	-T terminaux prêts à recevoir des messages
	-t date dernière modification du système
uname	nom de la version Unix
set	variables d'environnement

3. communications :

wall	permet à l'administrateur d'envoyer un message à tous les user
write jim	envoie un message à jim (dialogue entre terminaux) terminer par ctrl-D
mesg	affiche l'état y/n de la réception de messages
mesg -n	interdit la réception de messages
mesg -y	autorise...
mail jim	envoie un message dans la BAL de jim

4. répertoires :

ls liste des fichiers et répertoires
- -a tous les fichiers
- -C en colonnes
- -F les rép. avec / et les fichiers exe avec *
- -R affichage récursif des sous-rép
- -l détail des fichiers
- -r ordre de tri inverse
- -s taille en blocs (de 512 octets)

ls | more : affiche par page
ls | lp : imprime la liste

exemple d'affichage : **`d rwx rw- r-- 2 microjim group 128 Juil 01 19:28`**
d indique un répertoire
ensuite sont affichés les droits sur cet objet de l'utilisateur (user), du groupe (group), des non-membres du groupe (other) : r=read, w=write, x=execute
- rwx indique droits read, write et execute pour l'utilisateur
- rw- indique les droits read et write pour les membres du groupe
- r-- indique le droit read pour les non-membres du groupe (les autres)

"microjim" est le nom du propriétaire du répertoire
"group" est le nom du groupe
le répertoire a été créé le 1er juillet à 19h28

cd change de rép
cat > essai entrée au clavier dans le fichier essai
mkdir crée un rép
rmdir supprime un rép
mv déplace ou renomme un rép

5. fichiers

cat affiche un fichier texte
more contrôle de l'affichage
pg idem
lp imprime un fichier
mv déplace ou renomme un fichier
cp duplique un fichier
rm efface un fichier
find recherche un fichier

protections :
- **ls -l** affiche les droits
- **chmod** modifie les droits
- **umask** droits par défaut
- **chown** change de propriétaire
- **chgrp** change de groupe

ln crée un lien entre 2 fichiers
file nature du fichier
sort trie un fichier texte

6. disquettes

les disquettes sont gérées comme des fichiers spéciaux (dans rép /dev)

mount monte une disquette dans le système de fichiers
tar utilise la disquette comme une bande pour l'archivage

7. processus :

ps	renseigne sur les processus en cours
	-e affiche tous
	-f listing détaillé
	-g appartenant à un groupe spécifié
	-l listing long
	-p correspondant aux PID spécifiés
	-t associés aux terminaux spécifiés
	-u appartenant aux user spécifiés
kill	tue un processus en cours
nice	diminue la priorité d'un processus
at	permet de lancer un processus à une date donnée

8. utilisateurs :

adduser jim	ajoute un utilisateur "jim"
passwd jim	pour changer le mot de passe de "jim"

9. Création de script

un fichier exécutable sous UNIX peut être :
un programme en C (faut être programmeur)
ou un script (faut être utilisateur averti)

un script est un enchainement de commandes (semblable à un fichier .BAT sous DOS)

création d'un script :

utiliser un **éditeur** de texte (ex: vi) ou la commande d'édition "cat" :

$ cat > prog [enter]	lance la commande
...	on tape le contenu
^D	fin de commande

pour rendre le script **exécutable :**
$ chmod +x prog change l'attribut de "prog" (r=read, w=write, x=execute)

pour exécuter le script, tape son nom (avec des paramètres éventuellement) :
$ prog

le **chemin d'accès** au script doit être précisé :
ex: on a un script nommé "essai" dans le répertoire /usr/mrbt/bin
le répertoire courant est /usr/mrbt
on tape
$ essai
normalement ça ne marche pas
il faudrait taper
$ /usr/mrbt/bin/essai
ou bien
$ /bin/essai

on peut définir un chemin d'accès par défaut (comme PATH en MS-DOS)
les "chemins implicites" d'un utilisateur sont définis à la connexion via le script local **.profile**
ces chemins sont indiqués dans la variable PATH :
$ echo $PATH [enter]
/bin:/usr/bin:/usr/mrbt/bin:.
ici, toute commande placée dans le répertoire /bin , /usr/bin , /usr/mrbt/bin ou dans le répertoire courant (.) peut être lancée sans qu'il soit nécessaire d'indiquer son chemin.

attention à ne pas oublier (.) à la fin du PATH, sinon il faudrait indiquer

explicitement le répertoire courant dans une commande

exemples :

a) script donnant l'heure :

```
$ cat > heure [enter]
date +"%H:%M:%S [enter]
^D
$ chmod +x heure
```

b) script de recherche :

pour rechercher un fichier, on utilise fréquemment une commande du genre

```
$ find / -name toto -print 2>/dev/null
```

on veut la remplacer par la commande

```
$ cherche toto
```

on utilise donc un paramètre variable

```
$ cat > cherche
find / -name "$1" -print 2>/dev/null
^D
$ chmod +x cherche
```

variables dans UNIX :

$0	nom du programme en cours	$$	PID de la commande (Processus identifier)
$*	liste des arguments de la ligne de commande	$!	PID de la dernière commande asynchrone
$#	nombre d'arguments	$1	1er argument
$?	code de retour de la dernière commande synchrone	$9	9e argument

métacaractères :

!	NON logique	?	joker mono caractère
&&	ET logique	{ }	groupage de commandes, substitution
*	joker multi caractères	|	tube
;	séparateur de commandes	||	OU logique
<	redirection en entrée		
>	redirection en sortie		
>>	redirection en sortie avec ajout		

exemple :

```
$ jour='date +"%d/%m/%y"'
$ echo jour
14/05/97
```