R202 — Administration système et fondamentaux de la virtualisation

Laure Petrucci

IUT R&T Villetaneuse

14 février 2023

Plan du cours

1. Environnement Unix
2. Scripts *shell*
3. Gestion des utilisateurs
4. Gestion des processus
5. Initiation à la virtualisation

1 Environnement Unix
- Commandes externes et internes
- Historique de commandes
- Environnement utilisateur

2 Scripts *shell*

3 Gestion des utilisateurs

4 Gestion des processus

5 Initiation à la virtualisation

Commandes externes et internes

Commande externe

- Stockée dans un fichier exécutable
- Son chemin d'accès doit être connu pour l'exécuter
- Possibilité de destruction du fichier

Commande interne (*builtin*)

- Intégrée à l'interpréteur de commande
- Différentes selon le *shell* utilisé
- Toujours exécutable
- Ne peut pas être détruite

- `ls` est une commande externe se trouvant dans `/bin/ls`
- `echo`, `help` et `pwd` sont des commandes internes

Comment les différencier ?

- Utilisation de la commande interne type

```
1  $ type ls
2  ls is /bin/ls
3  $ type alias
4  alias is a shell builtin
5  $ alias bonjour="echo␣bonjour"
6  $ bonjour
7  bonjour
8  $ type bonjour
9  bonjour is aliased to `echo bonjour'
10 $
```

Historique de commandes

Dernières commandes tapées

- Stockées dans le fichier `.bash_history` (pour le shell bash)
- `history` affiche l'historique

Utilisation de l'historique

- `!!` affiche et exécute à nouveau la dernière commande ;
- `!`*n* affiche et exécute la commande numéro *n* de l'historique ;
- `!-`*n* affiche et exécute la *n*-ième dernière commande ;
- `!`*chaîne* affiche et exécute la dernière commande commençant par *chaîne*.

Variables d'environnement

Variables

- nom d'une variable : chaîne de caractères contenant des lettres, des chiffres ou le caractère _ et commençant toujours par une lettre
- valeur d'une variable : chaîne de caractères

Variables d'environnement

- Valeurs par défaut ou valeurs personnalisées
- Définies :
  - à la connection
  - lors de la création d'un nouveau terminal ou d'un nouveau shell
  - peuvent être modifiées et transmises aux processus fils

Visualisation

- liste avec `printenv`
- `which` *commande* affiche le chemin d'accès à la *commande*

Exemple de variables d'environnement

```
1  $printenv
2  MANPATH=/usr/share/man:/usr/local/share/man:
3  /usr/local/man:/usr/X11/man
4  TERM=xterm-color
5  SHELL=/bin/bash
6  SVN_EDITOR=vi
7  USER=petrucci
8  PATH=/opt/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:
9  /usr/local/bin:/usr/X11/bin
10 PWD=/Users/petrucci
11 LANG=fr_FR
12 HOME=/Users/petrucci
13 LOGNAME=petrucci
14 $ which ls
15 /bin/ls
16 $
```

Principales variables d'environnement

Variable	Description
DISPLAY	adresse du terminal sur lequel se fait l'affichage
HISTSIZE	taille de l'historique (nombre de commandes stockées)
HOSTNAME	nom de la machine
HOME	référence absolue du répertoire privé de l'utilisateur
LANG	langue d'affichage des messages
LOGNAME	identité de l'utilisateur (*login name*)
MANPATH	répertoires contenant les pages du manuel en ligne `man`
PATH	répertoires pour rechercher les fichiers exécutables
PS1	première invite de commande (*prompt*)
PS2	seconde invite de commande
PWD	répertoire de travail
SHELL	référence absolue du *shell*
TERM	type de terminal utilisé pour l'affichage
UID	numéro d'identificateur de l'utilisateur (*user identifier*)

Création / modification de variables

Affectation

```
var=val
```

Valeur

```
$var
```

Si la variable `var` n'a pas été définie, son contenu est la chaîne de caractères vide.

Portée

La variable est seulement connue du processus *shell* dans laquelle elle a été affectée. Pour qu'elle soit transmise aux *sous-shells*, elle doit être exportée :

```
export var
```

Exemple

```
1  $ PS1="Bonjour>"
2  Bonjour>bash
3  $ exit
4  exit
5  Bonjour>export PS1
6  Bonjour>bash
7  Bonjour>
```

Personnalisation de l'environnement de travail

- `/etc/profile` : à l'ouverture d'une session
- `~/.bash_profile`, `~/.bash_login`, `~/.login` : aussi à l'ouverture d'une session
- `~/.bashrc` : lors de l'exécution d'un nouveau shell
- `~/.bash_logout` : à la fermeture d'une session

1. Environnement Unix

2. Scripts *shell*
   - Scripts shell
   - Délimiteurs de chaînes de caractères et variables numériques
   - Paramètres des scripts
   - Structures de contrôle

3. Gestion des utilisateurs

4. Gestion des processus

5. Initiation à la virtualisation

Scripts *shell*

Un *script shell* est un fichier contenant une suite de commandes *shell*.

Un *script shell* permet de rassembler plusieurs commandes, par exemple, si l'on doit exécuter plusieurs fois une suite de commandes relativement longue. C'est alors une commande écrite par l'utilisateur.

Début d'un script *shell*

On peut indiquer, dans la première ligne du fichier, l'interpréteur shell à utiliser :

```
#!/bin/bash
```

Des commentaires peuvent être insérés dans le fichier, et ne sont pas interprétés par le *shell*. Les commentaires sont des chaînes de caractères commençant par `#`.

Exécution d'un script shell

ïl faut d'abord renrde le script exécutable :

```
$ chmod +x nom_script
```

On peut ensuite l'exécuter :

```
$ ./nom_script
```

Délimiteurs

Les délimiteurs permettent d'effectuer des opérations à l'intérieur de chaînes de caractères.

`'chaîne'`

la `chaîne` de caractères entre apostrophes (*quotes*) est utilisée telle quelle. En particulier, s'il y a des appels à des variables, aucune substitution n'est effectuée.

`"chaîne"`

la substitution des variables contenues dans la `chaîne` de caractères entre guillemets est effectuée.

`` `chaîne` ``

la `chaîne` de caractères entre apostrophes inversées (*backquotes*) est considérée comme une commande *shell* et est exécutée.

Affichage

`echo chaîne`

affiche la `chaîne` de caractères, avec éventuelle substitution des variables, suivant les délimiteurs utilisés.

`echo -n chaîne`

affiche la `chaîne` de caractères, sans retour à la ligne.

Variables numériques

Une variable peut avoir une valeur numérique entière. Le *shell* peut alors évaluer des expressions arithmétiques.

```
1  $ var1=3
2  $ echo $var1
3  3
4  $ var2=$(($var1+4))
5  $ echo $var2
6  7
7  $ var3=$[$var1*5]
8  $ echo $var3
9  15
10 $
```

Paramètres des scripts

Un *script shell* peut accepter des paramètres :

```
./nom_script paramètre_1 paramètre_2 ... paramètre_n
```

Variable	Description
$1, $2, ..., $9	paramètres de la commande
$0	référence de la commande
$*	liste de tous les paramètres de la commande
$#	nombre de paramètres passés lors de l'appel
$$	numéro du processus *shell* correspondant à la commande
$?	code de retour de la dernière commande exécutée

Exemple (12) : un script affichant les paramètres

```
1 #!/bin/bash
2 echo "paramètre␣1=$1"
3 echo "paramètre␣2=$2"
4 echo "paramètre␣3=$3"
5 echo "commande=$0"
6 echo "tous␣les␣paramètres=$*"
7 echo "nombre␣de␣paramètres=$#"
8 echo "numéro␣de␣processus=$$"
9 echo "valeur␣de␣retour␣de␣la␣commande␣précédente=$?"
```

Exemple (12) : résultat d'exécution

```
1  $ ./exemple12.sh tata titi toto
2  paramètre 1=tata
3  paramètre 2=titi
4  paramètre 3=toto
5  commande=./exemple12.sh
6  tous les paramètres=tata titi toto
7  nombre de paramètres=3
8  numéro de processus=31068
9  valeur de retour de la commande précédente=0
10 $
```

Structures de contrôle

si . . . alors . . . sinon . . . finsi

```
if liste_commandes1
then liste_commandes2
else liste_commandes3
fi
```

cas

```
case chaîne in
    motif1 ) liste_commandes1 ;;
    ...
    motifn ) liste_commandesn ;;
    * ) liste_commandes_autres_cas ;;
esac
```

Tests

Tests sur les chaînes de caractères

- `[ chaine1 = chaine2 ]`
  teste si les deux chaînes de caractères sont égales
- `[ chaine1 != chaine2 ]`
  teste si les deux chaînes de caractères sont différentes
- `[ -n chaine1 ]`
  teste si la chaîne de caractères est non vide
- `[ -z chaine1 ]`
  teste si la chaîne de caractères est vide

Tests

Tests sur les valeurs numériques

- `[ nb1 -eq nb2 ]` : égalité (*equal*)
- `[ nb1 -ne nb2 ]` : inégalité (*not equal*)
- `[ nb1 -gt nb2 ]` : plus grand (*greater than*)
- `[ nb1 -ge nb2 ]` : plus grand ou égal (*greater or equal*)
- `[ nb1 -lt nb2 ]` : plus petit (*lower than*)
- `[ nb1 -le nb2 ]` : plus petit ou égal (*lower or equal*)

Tests

Tests sur les fichiers

- `[ -d fichier ]` : teste si le fichier est un répertoire
- `[ -f fichier ]` : teste si *fichier* est un nom de fichier
- `[ -r fichier ]` : teste le droit de lecture sur le fichier
- `[ -w fichier ]` : teste le droit d'écriture sur le fichier
- `[ -x fichier ]` : teste le droit d'exécution sur le fichier

Exemple de test (13)

```
1 #!/bin/bash
2 if [ "$1" = 'fichier' ]
3 then if [ -f "$2" ]
4      then echo "$2␣est␣un␣fichier"
5      else echo "$2␣non␣trouvé"
6      fi
7 fi
```

Boucles

tant que ... faire ... fintq

```
while liste_commandes1
do
    liste_commandes2
done
```

pour ... faire ... finpour

```
for variable in liste_chaînes
do
    liste_commandes
done
```

Exemple de boucle (14)

```
1 #!/bin/bash
2 i=5
3 while [ $i -gt 0 ]
4 do
5     echo "i=$i"
6     i=$(($i-1))
7 done
```

Exemple de boucle (15)

```
#!/bin/bash
for i in 1 5 toto
do
    echo "$i"
done
```

Autres instructions

`set` *chaîne*

la *chaîne* de caractères devient la nouvelle liste de paramètres

`read` *liste_variables*

les variables prennent les valeurs fournies par l'entrée standard

`exit` *entier*

le script termine et renvoie l'*entier* comme code de retour

Exemple (16)

```
1  #!/bin/bash
2  set `ls`
3  for fich
4  do
5      if [ -d $fich ]
6      then echo "$fich␣est␣un␣répertoire"
7      else echo "Voulez-vous␣voir␣le␣contenu␣de␣$fich␣
8           read rep
9           case $rep in
10              o|O ) cat $fich ;;
11              n|N ) echo "Passage␣au␣fichier␣suivant"
12              * ) echo "Réponse␣incorrecte␣(traitée␣c
13          esac
14     fi
15 done
```

Exemple (17)

```
1 #!/bin/bash
2 chaine=$1
3 ps | grep $chaine | grep -v grep |
4   while read pid reste
5   do
6       kill -9 $pid
7   done
```

1 Environnement Unix

2 Scripts *shell*

3 Gestion des utilisateurs
- Création et suppression d'un utilisateur
- Groupes d'utilisateurs
- Droits des utilisateurs
- Quotas

4 Gestion des processus

5 Initiation à la virtualisation

Fichiers associés aux mots de passe

`/etc/passwd`

7 champs séparés par des `:`

1. `uname` : nom de connexion
2. `passwd` : mot de passe crypté
3. `UID` : numéro identifiant l'utilisateur
4. `GID` : numéro du groupe principal de l'utilisateur
5. `GECOS` : texte décrivant l'utilisateur
6. `homedir` : répertoire privé
7. `login shell` : commande exécutée à la connexion

- lisible par tous les utilisateurs
- pour des raisons de sécurité le `passwd` est remplacé par la lettre `x` sous Linux

Fichiers associés aux mots de passe

/etc/shadow

9 champs séparés par des :

1. uname : nom de connexion
2. passwd : mot de passe crypté
3. nombre de jours entre le 01/01/1970 et le dernier changement du mot de passe
4. nombre de jours minimum entre deux changements de mot de passe
5. nombre de jours pendant lesquels le mot de passe est valide
6. nombre de jours précédent l'expiration du mot de passe, utilisé pour prévenir l'utilisateur
7. nombre de jours de validité du compte après expiration du mot de passe
8. *date d'expiration du compte*, en nombre de jours depuis le 01/01/1970
9. *réservé* pour un usage futur.

Fichiers associés aux mots de passe

/etc/shadow

Lisible uniquement par l'administrateur

Valeurs particulières

2. - vide : aucun mot de passe n'est précisé, l'utilisateur n'en a pas
   - * : le compte est désactivé
4. 0 indique que l'utilisateur peut changer son mot de passe à n'importe quel moment
5. 99999 indique que le mot de passe ne doit pas nécessairement être changé

Création/suppression d'un utilisateur

- ajout dans `/etc/passwd`
- ajout dans `/etc/shadow`
- ajout dans la liste des groupes de l'utilisateur
- création du répertoire privé
- recopie éventuelle des fichiers d'environnement par défaut
- création de la boîte aux lettres dans `/var/spool/mail`

useradd et passwd

```
$ useradd -d /home/toto -s /bin/bash toto
$ passwd toto
```

userdel

```
$ userdel toto
```

Groupes d'utilisateurs

- droits d'accès aux fichiers similaires
- possibiilité d'appartenir à plusieurs groupes

/etc/group

4 champs séparés par des `:`

1. `group name` : nom du groupe
2. `password` : mot de passe (x ou rien)
3. `GID` : numéro identifiant le groupe
4. `users` : liste des noms d'utilisateurs séparés par des `,`

/etc/gshadow

4 champs séparés par des `:`

1. `group name` : nom du groupe
2. `password` : mot de passe crypté
3. `administrators` : liste des administrateurs séparés par des `,`
4. `users` : liste des noms d'utilisateurs séparés par des `,`

Création et suppression d'un groupe

groupadd et groupdel

```
$ groupadd -g 1001 rt1
$ groupdel rt1
```

Droits d'accès aux fichiers

Il y a 3 types d'utilisateurs de fichiers :

- propriétaire (user) : l'utilisateur propriétaire du fichier
- groupe (group) : les utilisateurs appartenant au même groupe que le fichier
- autres (other) : tous les autres utilisateurs

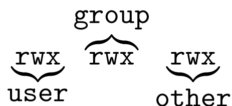
À chaque fichier sont associés différents droits :

- lecture (read) : possibilité de lire le fichier ou de regarder le contenu du répertoire
- écriture (write) : possibilité d'écrire le fichier ou d'écrire des fichiers dans le répertoire
- exécution (execute) : possibilité d'exécuter le fichier ou de traverser le répertoire

Droits d'accès aux fichiers

9 couples (type d'utilisateur, droit d'accès au fichier)

⇒ les droits d'accès sont codés sur 9 bits (indiqués par la commande `ls -l`).



Codage en octal :

`rwxr-x--- = 111101000 = 750`

Modification des droits d'accès

`chmod` (change mode)

`chmod droits liste_fichiers`

Les droits peuvent être caractérisés de 2 manières :

- code octal
- *utilisateur opération droits* où :

utilisateur : `u` (user), `g` (group), `o` (other)
opération : `+` (ajout), `-` (suppression), `=` (égale)
droits : `r` (read), `w` (write), `x` (execute)

Droits par défaut

Octal droits non attribués

Symbolique droits attribués

```
$ umask
0022
$ umask -S
u=rwx,g=rx,o=rx
$ umask 027
$ umask -S
u=rwx,g=rx,o=
$ umask u=rwx,g=r,o=
$ umask
0037
$ umask -S
u=rwx,g=r,o=
```

Changements de propriétaire et de groupe

chown (change owner)

```
chown utilisateur liste_fichiers

chown utilisateur.groupe liste_fichiers

chown -R utilisateur.groupe liste_fichiers
```

chgrp (change group)

```
chgrp groupe liste_fichiers
```

Autres permissions

3 bits spéciaux :

- **set-uid** permet d'exécuter un fichier avec les privilèges de son propriétaire et non pas ceux de l'utilisateur qui lance l'exécution.
- **set-gid** même chose avec le groupe.

bit de collage (sticky bit) assure le maintien de l'exécutable en mémoire même lorsqu'aucune exécution n'est en cours.

Quotas

Pourquoi des quotas ?

Limiter l'espace disque et le nombre de fichiers alloués aux utilisateurs

Limites

soft-limit : peut être dépassée temporairement

hard-limit : ne peut pas être dépassée

Comment ?

- monter la partition avec les options `usrquota` et `grpquota`
- présence à la racine de la partition des fichiers quota.user et quota.group avec les droits 600
- `quotaon` / `quotaoff` active / déactive les quotas
- `edquota` édite les quotas
- `repquota` liste les quotas

Limites du système

```
$ ulimit -a
-t: cpu time (seconds)              unlimited
-f: file size (blocks)              unlimited
-d: data seg size (kbytes)          unlimited
-s: stack size (kbytes)             8192
-c: core file size (blocks)         0
-v: address space (kbytes)          unlimited
-u: processes                       2784
-n: file descriptors                256
$ ulimit -n 512
$ ulimit -n
512
```

1 Environnement Unix

2 Scripts *shell*

3 Gestion des utilisateurs

4 Gestion des processus
- Généralités
- Mécanismes d'exécution
- Entrées/sorties, redirections
- Enchaînement de processus
- Signaux
- Commandes de gestion de processus

5 Initiation à la virtualisation

Généralités

Notion de processus

- Un processus est l'activité liée à l'exécution d'une commande
- Un utilisateur peut avoir plusieurs processus en cours à un instant donné
- Les différents processus existant à un instant donné sont indépendants et le processeur leur est attribué de façon imprévisible pour l'utilisateur

Types de processus

processus utilisateur : lancé par un utilisateur depuis un terminal

processus système : pas attaché à un terminal ; créé au démarrage ou à date fixes.

Mécanismes d'exécution

clonage : duplication avec `fork` du processus existant (père) pour créer un nouveau processus (fils), avec les mêmes instructions et une copie des données

substitution : remplace avec `exec` l'image mémoire du processus par une nouvelle image

suspension : un processus père attend avec `wait` la terminaison d'un ou plusieurs fils

Caractéristiques des processus

- **PID** (*process identifier*) : un numéro unique identifiant le processus
- **PPID** (*parent process identifier*) : identifiant du processus père
- **RUID** (*real user identifier*) : le numéro de l'utilisateur propriétaire réel du processus
- **EUID** (*effective user identifier*) : le numéro de l'utilisateur effectif du processus
- **RGID** (*real group identifier*) : le numéro du groupe propriétaire réel du processus
- **EGID** (*effective group identifier*) : le numéro du groupe effectif du processus
- **TERM** (*terminal*) : le terminal auquel le processus est rattaché

Mémoire allouée à un processus

4 zones

- segment de texte : les instructions à exécuter
- segment de données : gérée par le système pour contrôler le processus (état des *registres*, . . .)
- zone de données statiques : données connues au démarrage du processus et présentes durant toute l'exécution (*variables globales*)
- zone de données dynamique : objets non-permanents manipulés par le processus ; Zone divisée en 2 parties :
  - pile pour l'allocation des *variables locales*, des *paramètres de fonctions*
  - tas dans lequel se fait l'*allocation dynamique* de variables (*pointeurs*)

Programmes réentrants

Lorsque plusieurs processus exécutent le même programme, une seule copie du segment de texte est placée en mémoire et est utilisée par tous ces processus.

Entrées/sorties

- Entrées : données fournies à une commande
- Sorties : ce qui est écrit par la commande

Les entrées et sorties se font a priori sur des canaux spécifiques :

entrée standard associée au clavier

sortie standard associée à l'écran

sortie erreur standard également associée à l'écran

Redirections : pourquoi ?

On peut vouloir modifier les entrées/sorties, parce que, par exemple :

- les entrées sont contenues dans un fichier
- les sorties sont trop longues pour être lues à l'écran, donc on veut les mettre dans un fichier

$\Rightarrow$ on redirige le canal associé.

Redirection des entrées

Redirection de l'entrée standard

```
commande < nom_fichier
```

La commande `commande` prend ses entrées dans le fichier référencé par `nom_fichier`.

Redirection des sorties

Redirection de la sortie standard

```
commande > nom_fichier
```

redirige les sorties de la commande `commande` sur le fichier référencé par `nom_fichier`. Ce fichier est créé s'il n'existe pas ou écrasé s'il existe déjà.

```
commande >> nom_fichier
```

redirige les sorties de la commande `commande` sur le fichier référencé par `nom_fichier`. Ce fichier est créé s'il n'existe pas ou les sorties sont écrites à la fin du fichier s'il existe déjà.

Redirection de la sortie erreur standard

```
commande 2> nom_fichier
```

redirige les erreurs générées lors de l'exécution de la commande `commande`.

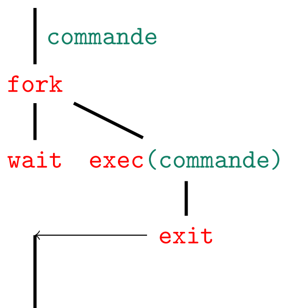
Exécution de commandes

Deux cas possibles :

commande interne : l'action est exécutée par l'interpréteur de commandes lui-même ;

commande externe : le nom de l'action est le nom d'un fichier contenant un programme exécutable. Le processus *shell* est dupliqué, et sa copie est remplacée par l'exécutable de la commande. Le processus d'origine attend la fin de l'exécution de la commande.

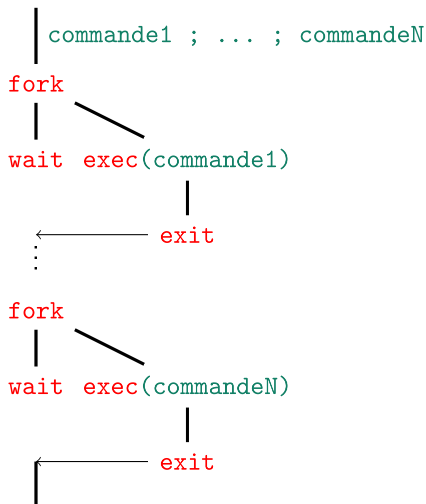
Exécution de commande externe



Enchaînements de processus

- séquentiel : `commande1 ; commande2`
- parallèle : `commande1 | commande2`
- tâche de fond : `commande &`

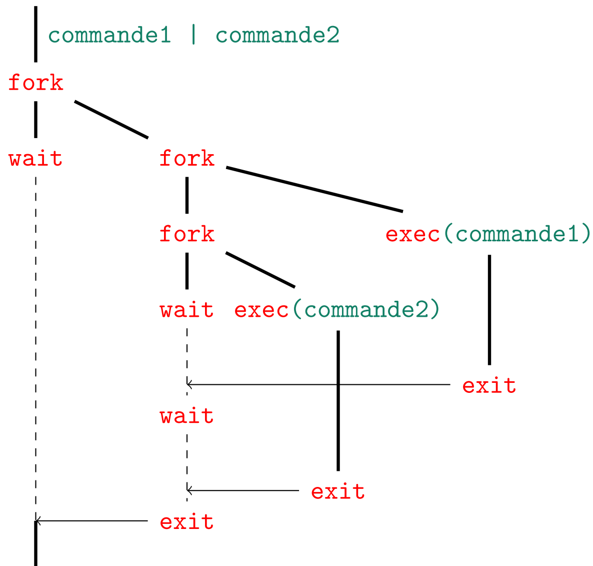
Enchaînement séquentiel



Enchaînement parallèle — tubes

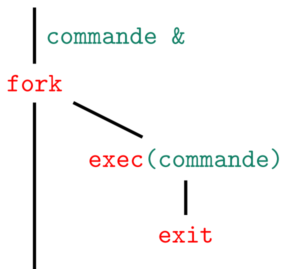
Dans `commande1 | commande2`, `|` représente un tube (*pipe*) : les sorties de `commande1` sont les entrées de `commande2`.

Enchaînement parallèle



Tâches de fond

`commande &` lance la `commande` en tâche de fond : l'interpréteur n'attend pas la fin de l'exécution de la commande et permet de relancer immédiatement une autre commande. Le système affiche le numéro du processus créé.



Signaux

Les signaux permettent d'avertir un processus qu'un événement important s'est produit. Le processus peut alors réagir à cet événement.

Principaux signaux

- `SIGINT` (2) (*interrupt*) est émis lorsque l'on tape `<CTRL>-c`
- `SIGQUIT` (3) (*quit*) est émis lorsque l'on tape `<CTRL>-\`
- `SIGKILL` (9) (*kill*) tue un processus, quel que soit son état
- `SIGALRM` (13) (*alarm*) est associé à une horloge
- `SIGTERM` (15) (*terminate*) est émis lorsqu'un processus termine normalement
- `SIGTSTP` (20) (*terminal stop*) est émis lorsque l'on tape `<CTRL>-z`

Suspension et reprise de processus

Suspension d'un processus qui s'exécute en avant-plan

`<CTRL>-z`

Reprise d'une tâche suspendue

- `fg` (*foreground*) relance la tâche dans le terminal
- `bg` (*background*) relance la tâche en tâche de fond

Arrêt complet d'un processus

commande `kill`

Attention : lorsque l'on a appuyé sur `<CTRL>-z`, on a l'impression que le processus est *mort*. Ce n'est pas le cas, il est seulement *suspendu*.

Gestion des tâches

L'interpréteur de commandes (*shell*) maintient une table des processus en cours d'exécution, qu'ils soient alloués ou non au processeur et suspendus ou non.

Lancement d'un processus en tâche de fond

```
[1] 25647
```

Cette ligne indique que c'est la tâche numéro 1, et que l'identificateur du processus est 25647.

Terminaison d'une tâche de fond

```
[2]+  Done        gedit
```

Cette ligne indique que la tâche numéro 2 s'est terminée, et que c'était `gedit`.

Contrôle des tâches

`jobs` affiche la table des tâches.

`%n` ou `fg %n` met la tâche numéro *n* en avant-plan.

`%n &` ou `bg %n` met la tâche numéro *n* en arrière-plan.

`ps` (*process status*) affiche les informations sur les processus.

`top` donne ces informations de manière interactive.

Arborescence des processus

```
$ ls | more | pstree
init-+-apache2-+-5*[apache2---12*[{apache2}]]
     |          `-10*[apache2]
     |-atd
     |-2*[automount]
     |-cron---cron---sh---run-parts---minieddie---superv
     |-dbus-daemon
     |-dhclient3
     |-ntpd
     |-portmap
     |-rpc.statd
     |-sshd-+-3*[sshd---sshd---nc]
     |      |-sshd---sshd---bash
     |      |-sshd---sshd---sftp-server
     |      `-sshd---sshd---bash-+-bash
     |                          |-more
     |                          `-pstree
```

Commandes associées aux signaux

Associer une action à un signal

```
trap commande numsignal
```

associe la `commande` au signal de numéro `numsignal`

```
trap numsignal
```

associe de nouveau la commande par défaut au signal de numéro `numsignal`

Envoyer un signal à un processus

```
kill -numsignal pid
```

envoie le signal de numéro `numsignal` au processus numéro `pid`

Exemple

```
$ trap 'echo coucou' 2
$ ps | grep bash
  296 ttys000    0:00.02 -bash
  312 ttys000    0:00.02 bash
  324 ttys000    0:00.00 grep bash
$ kill -2 312
coucou
$ coucou
$ trap 2
$ kill -2 312
$
```

No Hangup

Les processus utilisateurs sont tués lorsque l'on quitte leur terminal (ils reçoivent le signal SIGHUP).

```
nohup commande
```

lance la commande en inhibant le signal 1. Le résultat est écrit dans un fiochier nohup.out

Gestion des priorités

Niveaux de priorités

- priorités de 0 à 20
- plus le nombre est faible, plus le processus est prioritaire
- le système allour le processeur d'abord aux processus les plus prioritaires

Attribution d'une priorité à un processus

```
nice -n num_priorité commande
```

lance la `commande` avec le numéro de priorité `num_priorité`.

- seul l'administrateur peut augmenter une priorité
- sert (surtout) à déclarer qu'un processus est peu prioritaire
- `renice` permet de modifier la priorité d'un processus en cours d'exécution

Gestion du temps

Exécution à une date précise

```
at date
```

attend une commande sur l'entrée standard et planifie son exécution à la date indiquée

Exécution répétitive

```
crontab -e
```

édite un fichier de tâches répétitives devant être exécutées automatiquement. Le processus système `cron` en gère l'exécution.

Utilisateurs autorisés/interdits pour `at` et `cron`

listés dans les fichiers `/etc/at.allow`, `/etc/at.deny`, `/etc/cron.allow` et `/etc/cron.deny`

Fichier des tâches répétitives

Chaque ligne a la structure suivante :

1. minutes auxquelles l'exécution a lieu, de 0 à 59
2. heures auxquelles l'exécution a lieu, de 0 à 23
3. jours du mois auxquels l'exécution a lieu, de 1 à 31
4. mois auxquels l'exécution a lieu, de 1 à 12
5. jours de la semaine auxquels l'exécution a lieu, de 0 (dimanche) à 6 (samedi)
6. commande à exécuter

On peut spécifier une plage avec un - ou toutes les possibilités avec $*$

Exemple

$$\underbrace{0,30}_{(1)}\ \underbrace{8-20}_{(2)}\ \underbrace{*}_{(3)}\ \underbrace{*}_{(4)}\ \underbrace{1,3,5}_{(5)}\ \underbrace{/\texttt{usr}/\texttt{local}/\texttt{bin}/\texttt{sauvegarde}}_{(6)}$$

spécifie une exécution :

1. à l'heure exacte et à la demie ;
2. toutes les heures de 8 heures à 20 heures ;
3. tous les quantièmes ;
4. tous les mois ;
5. seulement les lundi, mercredi et vendredi ;
6. de la commande `/usr/local/bin/sauvegarde`.

1. Environnement Unix

2. Scripts *shell*

3. Gestion des utilisateurs

4. Gestion des processus

5. Initiation à la virtualisation
   - Système d'exploitation et noyau
   - Virtualisation

Système d'exploitation et noyau

Rôle du système d'exploitation

- interface entre les applications et le matériel
- gérer les accès au matériel
- ordonnancement des processus

parce que :

- programmation trop complexe dans un programme
- garantir un minimum de sécurité d'accès

Noyau

- Programme central du système d'exploitation
- Premier programme à s'exécuter, au démarrage de la machine
- s'exécute dans un « espace-noyau » avec tous les privilèges d'accès
- Premier programme exécuté par la machine

Objectifs de la virtualisation

- mieux exploiter les nombreuses ressources de la machine
- améliorer le système d'exploitation (sécurité, stabilité, facilité d'administration) par un cloisonnement des serveurs
- exécuter plusieurs systèmes d'exploitation sur uyne même machine physique

Sytème hôte

orchestre l'utilisation des ressources matérielles de la machine

Sytème invité

s'exécute en espace utilisateur

demande au système hôte d'accéder aux ressources matérielles

Hyperviseurs

Qu'est-ce qu'un hyperviseur ?

Noyau hôte allégé dont le seul rôle est d'exécuter les systèmes invités

Hyperviseurs de type I

entre le matériel de la machine et les systèmes invités

Exemple : `kvm`

Hyperviseurs de type II

entre le système d'exploitation hôte et les systèmes invités

Exemple : `VirtualBox`