Introduction au Système d'Exploitation Unix/Linux

Processus Unix

B. Jacob

IC2/LIUM

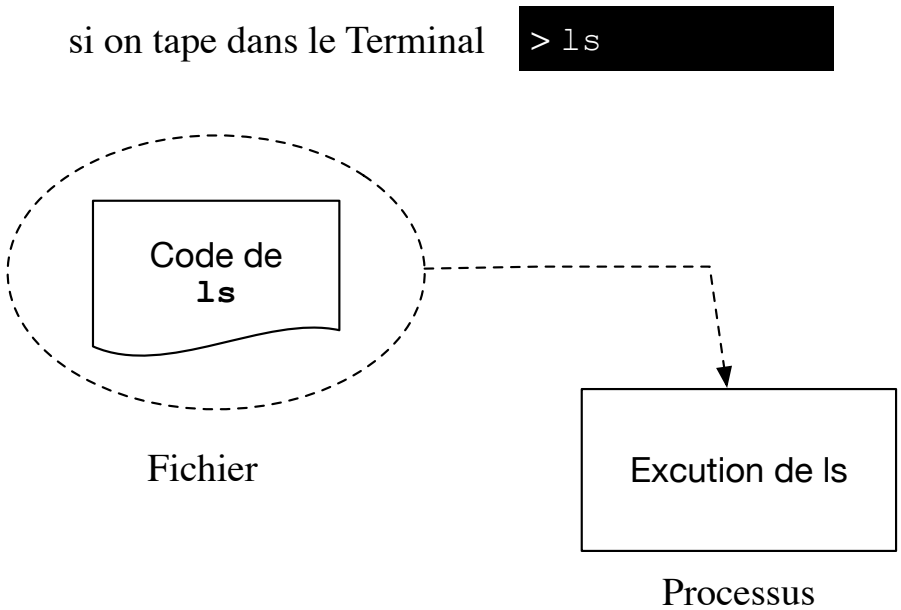
18 septembre 2017

Plan

Plan

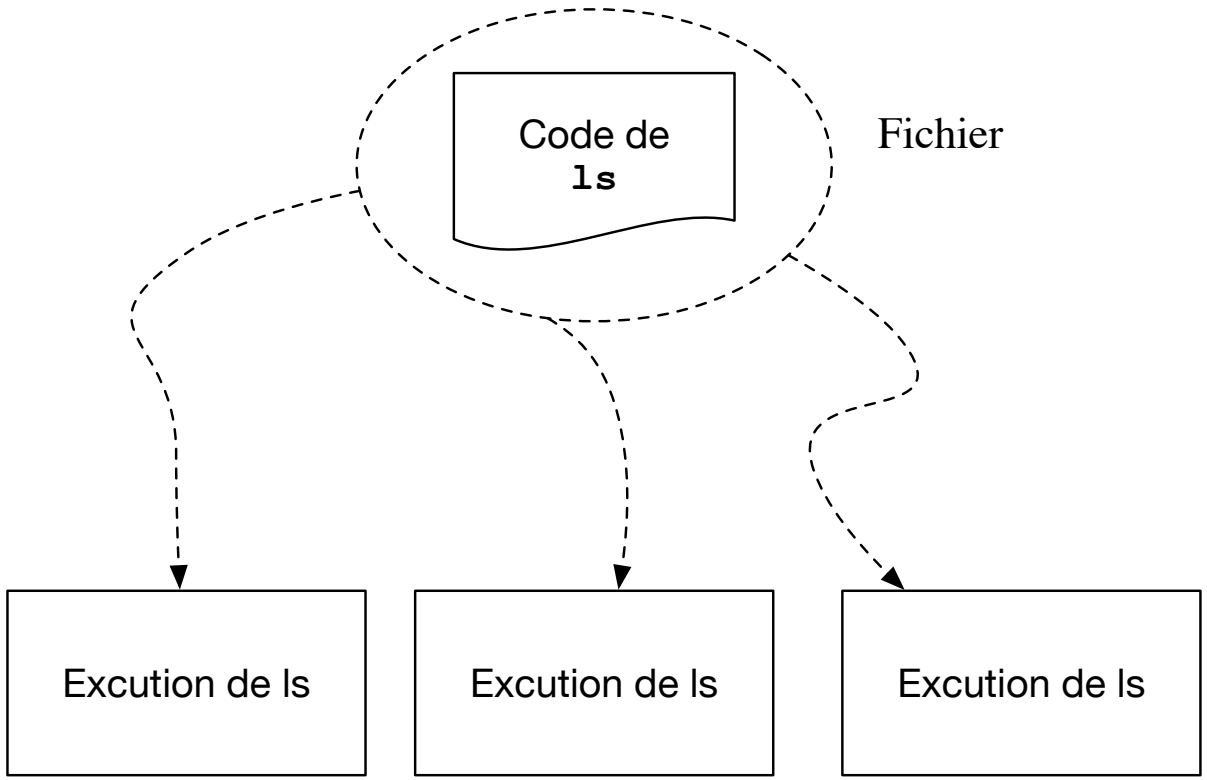
Processus/Programmes

- Programme exécutable = commande
  = lignes de codes
  = statique
- Processus
  = exécution d'un programme
  = dynamique



Processus/Programmes

Si on tape plusieurs fois la même commande
$\Rightarrow$ on exécute plusieurs fois la même commande
$\Rightarrow$ plusieurs processus exécutent la même commande



E/S standards

Par défaut, un processus

- à une entrée standard (n°0) : le clavier
- à une sortie (n°1) : l'écran
- à une sortie des erreurs (n°2) : l'écran

Plan

Les redirections

On peut rediriger les E/S d'un processus avec :

- > pour la sortie standard
- < pour l'entrée standard
- pour la sortie erreur
  - 2> sous `sh, bash`
  - >& sous `csh, tcsh`

Redirections écran

- redirection sortie standard de `echo` :

```
echo ‘‘coucou’’ > fich
```

```
% cat fich
coucou
```

- redirection sortie standard de `ls` :

```
% ls -l
total 6
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich2
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich3
```

```
ls -l > listfich
```

```
% more listfich
total 6
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich2
-rw-r--r--   1 jacob parole 7 sep 22 16:00 fich3
-rw-r--r--   1 jacob parole 0 sep 22 16:03 listfich
```

Redirection clavier

Exemple avec la commande `sort`

```
% sort
zzz
aaa
kkk
^D
aaa
kkk
zzz

% more fich_clavier
zzz
aaa
kkk
```

% `sort < fich_clavier` donne la même chose :

```
zzz
aaa
kkk
```

Redirection sortie erreur

Exemple avec `ls` sur un fichier inexistant

```
% ls
clavier    fich       fich2      fich3      listfich
```

- sous `tcsh, csh`
  `% ls toto >& erreurs`
- sous `sh, bash`
  `% ls toto 2> erreurs`

```
% more erreurs
toto: Ce fichier ou ce répertoire n'existe pas
```

Concaténation des redirections

- >> pour la sortie standard
- pour la sortie des erreurs
  - sous `tcsh, csh`
    >>&
  - sous `sh, bash`
    2>>

Concaténation sortie standard

```
%  more fich
coucou
```

```
% echo salut >> fich
```

```
%  more fich
coucou
salut
```

Concaténation sortie erreur

```
% more erreurs
toto: Ce fichier ou ce répertoire n'existe pas
```

```
% cat titi >>& erreurs
```

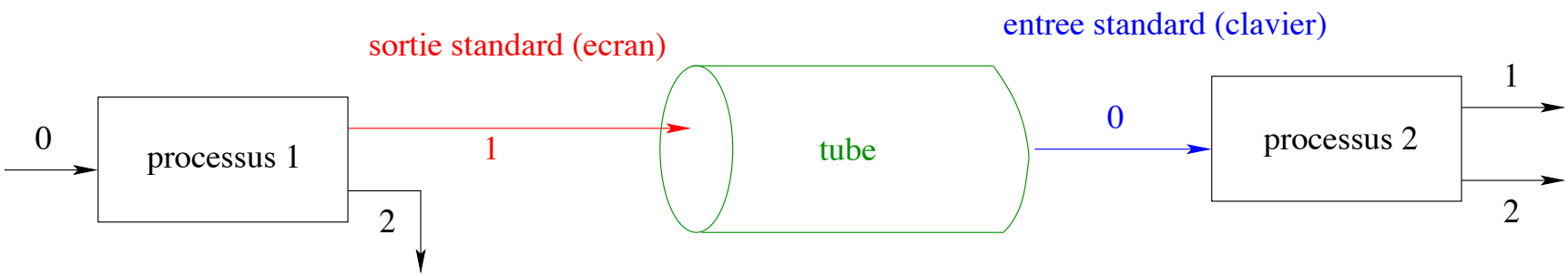
```
% more erreurs
toto: Ce fichier ou ce répertoire n'existe pas
cat : impossible d'ouvrir titi
```

Tubes

Un tube est une redirection entre 2 processus/commandes

```
% processus1 | processus2
```

- la sortie standard (n°1) du processus1 est redirigée vers l'entrée standard (n°0) du processus2
- le résultat du processus1 est la donnée du processus2



Exemple tube 1

Exemple avec les commandes

- `wc` : sans argument `wc` attend que l'on tape les lignes au clavier pour les compter
- `cat` : affiche à l'écran les lignes d'un fichier

```
%  wc -l
lig1
lig2
^D
       2

% cat fich
coucou
salut
hello
bonjour
```

```
% cat fich | wc -l
       4
```

Exemple tube 2

Exemple avec les commandes

- `wc` : sans argument `wc` attend que l'on tape les lignes au clavier pour les compter
- `ps` : affiche à l'écran les processus en cours d'exécution

```
%  wc -l
lig1
lig2
^D
       2

%  ps
  PID TTY           TIME CMD
  268 ttys000    0:00.17 -bash
```

```
% ps | wc -l

       3
```

→ Pourquoi ?

Plan

Notion de Filtres en Unix

Ce sont des commandes qui utilisent les 3 E/S standards

- Filtres unix $\Leftrightarrow$ Filtre en traitement du signal
  - filtre signal : prend un signal en entrée, le modifie, l'envoie sur sa sortie pour être éventuellement repris par un autre filtre
  - filtre unix : lit des données sur son entrée standard, les modifie, les écrit sur sa sortie standard.
    Ils peuvent donc être enchaînés avec des tubes
    $\Rightarrow$ construction de commandes complexes
- possibilité d'écrire des filtres en C

Filtres les plus connus sous Unix

- tr
- cut
- sort
- paste
- uniq

Tous ces filtres

- lisent donc leur données sur l'entrée standard et
- écrivent leurs résultats sur la sortie standard

Filtre `tr`

Syntaxe = `tr [options]` *`chaine1 chaine2`*

- tr = translate characters
- substitution ou suppression de caractères sélectionnés
- Un caractère $\in$ *chaine1* est remplacé par le caractère de même position dans *chaine2*
- Options principales :
  - `-d` : suppression des caractères sélectionnés
  - `-s` : "aaaaa" dans *chaine1* $\rightarrow$ "a" dans *chaine2*
- Abréviations :
  - `[a-z]` = segment de 26 car. allant de 'a' à 'z'
  - `[a*n]` = a...a (n fois)
  - `\xyz` = désigne le car. de code octal `xyz`

Exemple

```
$echo "coucou" | tr [a-z] [A-Z]
COUCOU

$ echo "aaabbbaaa" | tr -s [a-z] [A-Z]
ABA

$tr -d"\150" < fich_MS_DOS.c > fich_Unix.c
# le CR est elimine : cas transfert fichier MS DOS vers Unix
```

Filtre cut

- cut = sélectionne des caractères selon leur position dans la ligne
- Position par rapport :
  - au n°d'octet (`cut -b` *`no octet`*)
  - au rang du car. (`cut -c` *`rang`*)
  - au n°de champ (`cut -f` *`champs`* `-d` *`delimiteur`*)
- Option
  - `-s` (avec `-f`) : supprime les lignes vides

Exemple

```
#!/usr/bin/sh
# Selection sur le no d'octet
cut -b 5,7-10  fich_cut.txt

# Selection sur le rang du caractere
cut -c 1-5,10  fich_cut.txt

# Selection sur les champs delimites par un ":"
cut -d: -f1,3  fich_cut.txt
```

⇝ *exécution*

Filtre `sort`

- sort = trie, fusionne un ou plusieurs fichier(s)
- tri lexicographique
- Options :
  - `-o` *`fichier`* le résultat est mis dans *fichier*
  - `-k` *`n1,n2`* : clé = champ *n1* à *n2*
  - `-t` *`delimiteur`* : car. de séparation des champs (avec `-k`)
  - `-u` : (unique) efface toutes les lignes sauf une qui ont la même clé

Exemple

```
#!/usr/bin/sh

# Exemple de tri
sort fich_sort1.txt

# Exemple de fusion
sort fich_sort1.txt fich_sort2.txt

# Exemple avec delimiteur
sort -t: -k2 fich_sort1.txt fich_sort2.txt

# Exemple avec cle unique
sort -t: -k2  -u fich_sort1.txt fich_sort2.txt
```

⇝exécution

Filtre paste

- usage : `paste` *`fichier1 fichier2...`*
- concatène les lignes de même n°dans *fichier1* et *fichier2*
- option
  - `-d` *caractère* : concatène avec *caractère* au lieu de la tabulation par défaut
- Exemple

```
#!/usr/bin/sh
paste fich_paste1.txt fich_paste2.txt
```

⇝*exécution*

Filtre `uniq`

- `uniq` = donne un seul exemplaire des lignes
- Options :
  - `-u` seules les lignes en 1 exemplaire sont affichées
  - `-c` donne le nombre d'exemplaires de chaque ligne

Exemple

```
#!/usr/bin/sh

# Affichage des lignes en 1 seul exemplaire
uniq fich_uniq.txt

# Affichage des lignes qui sont en 1 seul exemplaire
uniq -u fich_uniq.txt

# Affichage des lignes avec leur nombre d'occurences
uniq -c fich_uniq.txt
```

⇝exécution

Plan

Enchainement des commandes

- ; séquencement (synchronisme)
  les commandes s'excutent séquentiellement
- & lancement en *background* (asynchronisme) les commandes s'exécutent en parallèle

Exemples

- `xterm ; firefox ; emacs`
  → `xterm` puis `netscape` puis `xemacs`
- `xterm & firefox & emacs &`
  → `xterm` en même temps que `firefox` et que `emacs`

Arrêt des processus

Cela revient à envoyer un signal d'interruption au processus

- processus synchrone : appuyer sur `Ctrl-C`
  → revient à envoyer le signal `SIGINT` au processus
  → interrompt le processus

- processus asynchrone / en background :
  envoyer explicitement un signal d'arrêt au processus
  → commande `kill` [n^o *signal*] *PID processus*

Exemples

- Liste des processus actif par `ps`

```
% ps
  PID TTY          TIME CMD
  268 ttys000    0:00.25 -bash
 6471 ttys000    0:08.50 ./JeNePeuxPlusMArreter
 6474 ttys001    0:00.01 -bash
```

- Arrêt du 2^{ieme} processus

  `% kill 6471` ou

  `% kill -2 6471` ou `% kill -INT 6471`

- Vérification

```
% ps
  PID TTY          TIME CMD
  268 ttys000    0:00.25 -bash
 6474 ttys001    0:00.01 -bash
```

Exemples

Il arrive que des processus ignorent le signal `SIGINT`

Envoyez alors un autre signal

- `SIGQUIT` par `% kill -3 ...`
- `SIGKILL` par `% kill -9 ...`
  (celui-ci ne peut être ignoré)

Plan

ssh

On peut faire exécuter un processus sur une machine distante avec la commande `ssh` = Secure Shell

1. Connexions sur la machine distante
   - `% ssh machinedist`
   - ou `% ssh machinedist -l login`
2. Eventuellement identification : par votre password
3. Lancement du processus : idem que sur votre propre machine
4. Déconnexion : `logout`