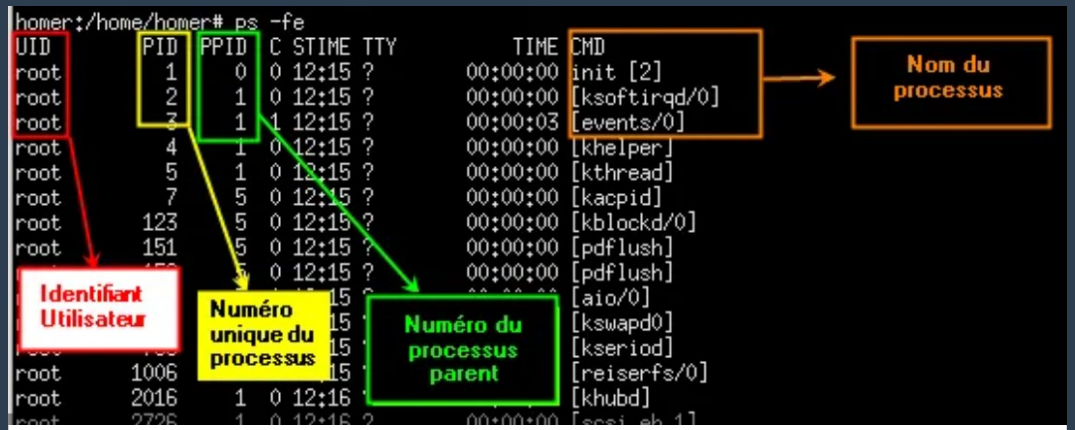


Processus

Plan du cours en 4 x 3h30

- **OS & UNIX**
- **Prise en main d'un système Unix (term)**
- **Système de fichiers Unix**
- **Un editeur de texte (vi)**
- **Expressions Régulières (regex)**
- **Processus (proc)**
- **Langages de commandes (shell)**
- **Le monde unix (X, net, ssh, ftp, http, mail,...)**

Processus (proc)

- Gérer les processus
- Entrées Sorties & Redirections des Processus
- Contrôle d’une séquence de processus

Gérer les processus

Un processus = un programme qui s’execute

Programme executable, avec ses données et sa pile d’execution, son compteur ordinal, son pointeur de pile et autres registres, ainsi que toutes les innformations necessaires.

Environnement d’exécution

Un processus comporte schématiquement une **zone de code exècutable** et une **zone de données**.

Un serveur Unix = Ensemble de processus

Exemple : processus d’accueil (cron)

un shell

Quand on tape une commande, on créer un processus

Gérer les processus

ps kill sleep nice at wait exec exit

$ ps [options] ex : ps -aux

liste les processus actif

$ top (liste dynamique)

$ kill <processus> ex : kill -9 5243

Détruit un processus

sleep -n

Suspend un processus pendant n secondes dans un script

$ nice -n <commande>

Execute avec une priorité n

$ at <heure> ex:at 11am ps -aux

Lance une commande à une date absolue.

$ wait <processus>

Attente de la fin du processus numero PID

$exec <commande>

La commande est executée sans créer un nouveau processus à la place du shell (delogue à la fin)

exit n

Termine une suite de commande avec un code de retour n

Gérer les processus

Transition d’état d’un processus

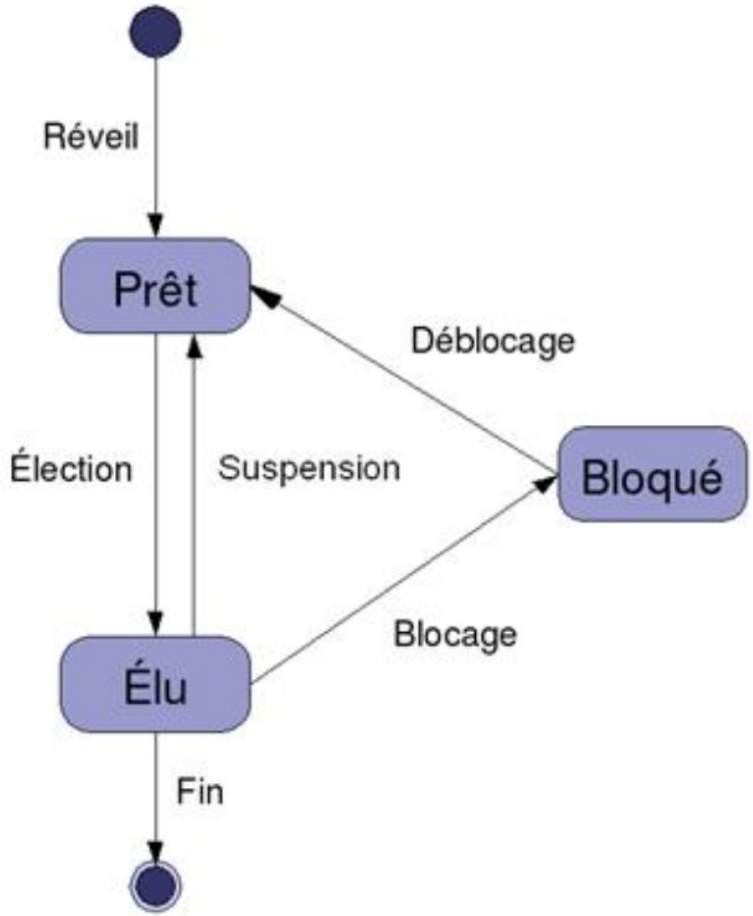
Vie des processus

Les proc sont manipulés par 4 opérations :

- Fork => créer des processus fils (code et data idem au père à la création puis peut évoluer)
- Exec => remplace code et data par un autre programme (passage d’args) execution dans le même contexte
- Exit fin normale d’un processus
- Wait attends l’exit d’un fils

Les jobs (cmd1;cmd2 ≡ job1;job2)

Un job j peut être arreté (wait j et repris %j). On peut jouer sur un job bg, fg, kill et %



Entrées Sorties (IO ES) des Processus

Les fichiers standards Entrée/Sortie

Toute ecécution de commande (ou processus) s’effectue avec assignation par défaut de 3 unités standard E/S

Par défaut, les 3 flux de données sont assignés au terminal de l’utilisateur devtty…., un flux associé à un élément :

- D’affichage (l’écran)
- De saisie (le clavier)
- D’affichage des messages d’erreurs.

E/S Redirections Simples : <>>>

Les redirections : < > >>

Pour la durée de l'execution d'un processus on peut changer les flux E/S par des directives

- > fichier sort sur un fichier comme sortie standard
  - Ex : $ ls > t envoie ls dans un fichier t
- < prend un fichier comme entrèe standard
  - Ex : $ wc < t
- > > fichier comme > mais ajoute à la fin si fichier existe
  - Ex : $ ls > > t

E/S Redirection de la sortie standard Erreur

Séparer la sortie standard de la sortie erreur

Permettre d'alimenter un fichier d'erreur pour alerter un utilisateur

Tout en gardant des traces dans un log principal

Exemple :

$ cc toto.c 2>t
Redirige les erreurs (sortie standard 2) de compilation vers le fichier t

$ echo ''toto'' >&2 (envoie toto en sortie standard 2 (erreur)

$ dosomething 2>&1 (renvoie vers la sortie standard1 'les erreurs'

Fonction	Bourne shell (sh, bash)	Z-shell (zsh)
Redirige la sortie d'erreur (2) et la sortie standard (1) sur l'entrée de la commande suivante	`2>&1 |`	`|&`
Redirige la sortie d'erreur et la sortie standard vers `fichier`	`>fichier 2>&1`	`>& fichier`
Redirige la sortie d'erreur et la sortie standard à la fin de `fichier`	`>>fichier 2>&1`	`>>& fichier`

E/S Redirection entre processus : tubes, pipes, |

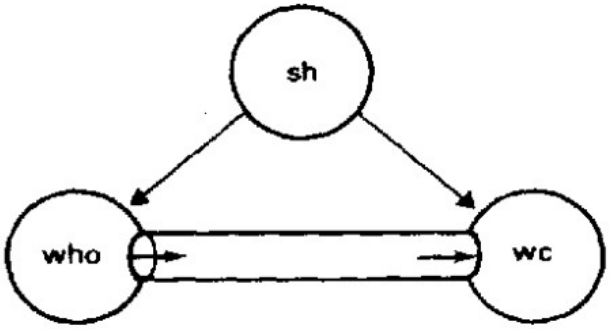
Connecte la sortie standard sur l'entrée d'un autre processus

$ <commande 1> | <command 2 >

Les processus s'executent en paralléle

- Exemple :

$ who > tmp
$ wc -l <tmp
≡
$ who | wc -l



Le noyau crée un fichier appelé tube de qq Ko dans /dev.

Un tube est un fichier en mémoire, il n'y a pas d'E/S disque

Exemple :

$ capture | filtre | sort | tail 10

E/S Redirection entre processus : tubes, pipes, |

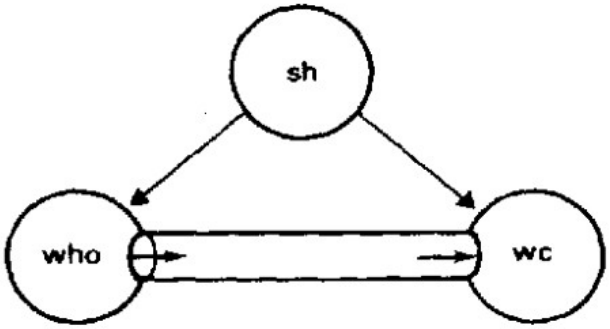
Connecte la sortie standard sur l'entrée d'un autre processus

$ <commande 1> | <command 2 >

Les processus s'executent en paralléle

- Exemple :

$ who > tmp
$ wc -l <tmp
≡
$ who | wc -l



Le noyau crée un fichier appelé tube de qq Ko dans /dev.

Un tube est un fichier en mémoire, il n'y a pas d'E/S disque

Exemple :

$ ls | wc -l

$ ps -elf | grep gnome

$ capture | filtre | sort | tail 10

E/S Redirection entre processus : en T (tee)

Sortir vers 2 sorties (tee)

Il est parfois utile de pouvoir sortir vers un fichier ET vers un autre processus

tee permet de recopier son entrée std vers sa sortie standard

- Exemple :

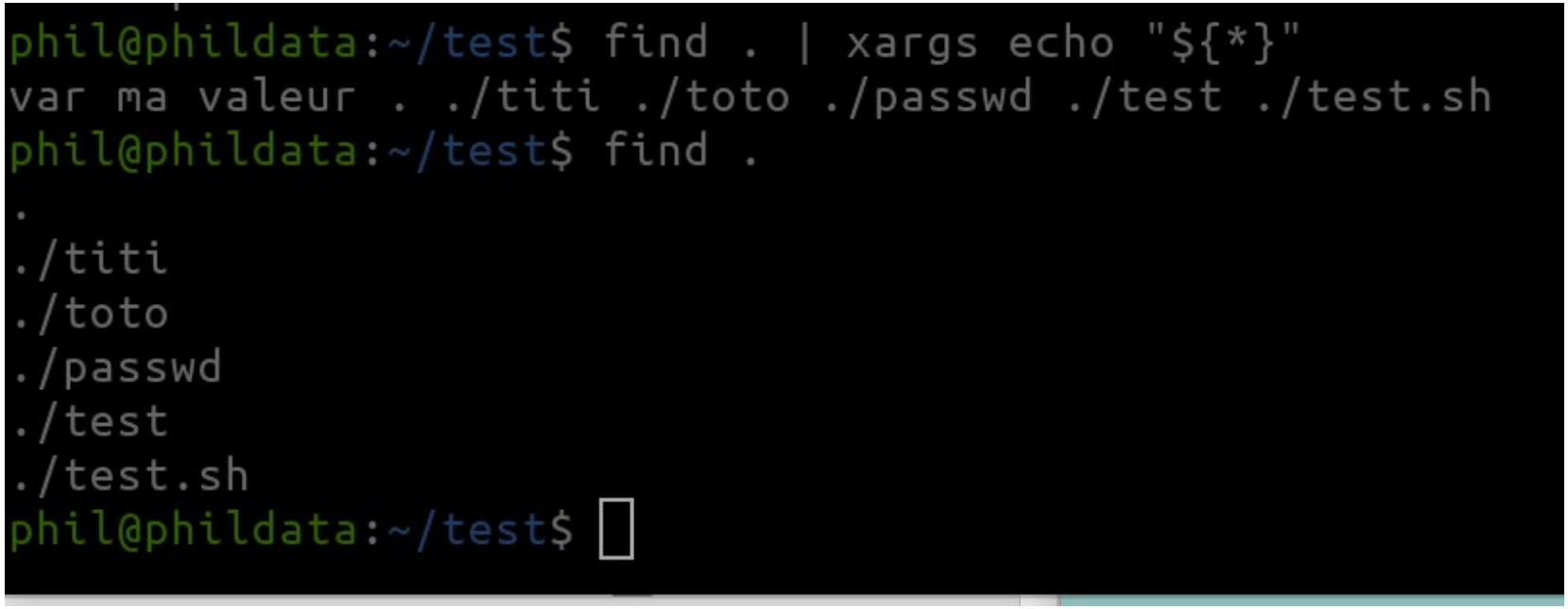
```
$  who  | tee liste.txt | wc -l > nombre.txt
≡
$ who > liste ; cat liste.txt | wc -l > nombre.txt
```

E/S Redirection entre processus : passage de d'arguments (xargs)

Communiquer une entrée standard vers les paramètres d'un autre processus

Xargs permet de recopier son entrée std vers sa sortie standard

- Exemple :



E/S Redirection entre processus : outil sur les flux : sed

Modification d'un flux de texte sans passer par un editeur (vim)

Utilise les expressions régulières pour modifier le flux des contenu

```
sed [OPTION]... {script-only-if-no-other-script} [input-file]...

# Commandes :
sed -e --> commande uniligne
sed -f fichier --> passage de commande par un script
sed -i --> modifie directement le fichier source au lieu de rediriger vers
```

Quelques exemples :

$ sed '/^#/d' test.txt # supprime les lignes commencant par #
$ sed '1d;4d' test.txt # supprime ligne 1 et 4
$ sed -n '/^#/p' test.txt # inverse on garde uniquement les lignes # et on les redirige vers l'imprimante
$ sed -r 's/toto\.txt/tutu.txt/g' text.txt

$ man sed

E/S Redirection entre processus : Code retour & conditions logiques

proc

En condition standard sans erreur, un script se termine par un 0

Une sortie exit <n> retourne une valeur différente de 0

$ cmd1 && cmd2 <CR>

- Si retour cmd1 <> 0 alors cmd2 n’est pas activée
- Si retour cmd1 =0 alors cmd2 est executée et son code de retour est celui de la ligne

$ cmd1 || cmd2 <CR>

- Si retour cmd1 <> 0 alors cmd2 est executée
- Si retour cmd1 = 0 alors cmd2 n’est pas executée

Conclusion :

Le fonctionnement d’un unix est un ensemble de processus qui communiquent entre eux

Gèrer des processus c’est :

- Lister les processus en cours, en créer, en détruire, en stopper
- Identifier l’état d’un processus en cours
- C’est de les connecter entre eux