Sujet 103
Commandes GNU & Unix

Commandes GNU & Unix

- 103.1 : Travail en ligne de commande (Val. 4)
- 103.2 : Traitement de flux de type texte avec des filtres (Val. 3)
- 103.3 : Gestion élémentaire des fichiers (Val. 4)
- 103.4 : Utilisation des flux, des tubes et des redirections (Val. 4)
- 103.5 : Création, contrôle et interruption des processus (Val. 4)
- 103.6 : Modification des priorités des processus (Val. 2)
- 103.7 : Recherche dans des fichiers texte avec les expressions rationnelles (Val. 2)
- 103.8 : Édition de fichiers texte avec vi (Val. 3)

103.5
Création, contrôle et interruption des processus (Val. 4)

Création, contrôle et interruption des processus

- Description : Les candidats doivent être en mesure d'effectuer une gestion élémentaire des processus.
- Termes, fichiers et utilitaires utilisés pour cet objectif :
  - &
  - bg
  - fg
  - jobs
  - kill
  - nohup
  - ps
  - top
  - free
  - uptime
  - pgrep
  - pkill
  - killall
  - screen

Processus

- Un processus représente à la fois un programme en cours d'exécution et tout son environnement d'exécution (mémoire, état, identification, propriétaire, processus parent, etc.).
- L'exécution d'une commande externe crée un processus. Ce n'est pas le cas d'une commande interne.

Identification

- Un numéro de processus unique PID (Process ID).
- Un numéro de processus parent PPID (Parent Process ID).
- Un numéro d'utilisateur et un numéro de groupe : UID et GID de l'user qui a lancé le processus.
- Priorité.
- Répertoire de travail actif.
- Fichiers ouverts :
  - Table des descripteurs des fichiers ouverts. Par défaut au début seuls trois sont présents : 0, 1 et 2 (les canaux standards).
- État du processus.
- Etc.

États d'un processus

- Durant sa vie (temps entre le lancement et la sortie) un processus peut passer par plusieurs états :
  - Nouveau processus.
  - Prêt à l'exécution (runnable).
  - Exécution en mode utilisateur (user mode).
  - Exécution en mode noyau (kernel mode).
  - En attente E/S (waiting).
  - Endormi (sleeping).
  - Endormi dans le swap (mémoire virtuelle).
  - Fin de processus (zombie).

Lancement en avant plan

- Quand une commande externe est saisie, le Shell crée un nouveau processus pour l'exécuter, ce processus devient un processus enfant du processus Shell.
- Il faut attendre la fin de l'exécution d'une commande lancée en avant plan pour en saisir une autre.
- GNU/Linux est multi-tâche : rien n'empêche le Shell d'attendre la fin du processus pour rendre la main à l'utilisateur.

Lancement en arrière plan

- Le Shell peut autoriser la saisie d'une nouvelle commande sans attendre la fin de l'exécution de la commande précédente. Il suffit d'exécuter la commande en arrière plan.
- Pour exécuter une commande en arrière plan il faut ajouter après la commande le caractère : &.
- Dans ce cas, le Shell affiche directement l'invite de commandes et il est prêt à exécuter une nouvelle commande.

Exemples

```
etudiant@lpi:~$ sleep 60
```

- Après 60 secondes retour du prompt.

```
etudiant@lpi:~$
```

```
etudiant@lpi:~$ sleep 60 &
[1] 2142
etudiant@lpi:~$
```

- Le prompt est affiché directement.
- Après 60 secondes :

```
[1]+ Fini sleep 60
```

Processus en arrière plan

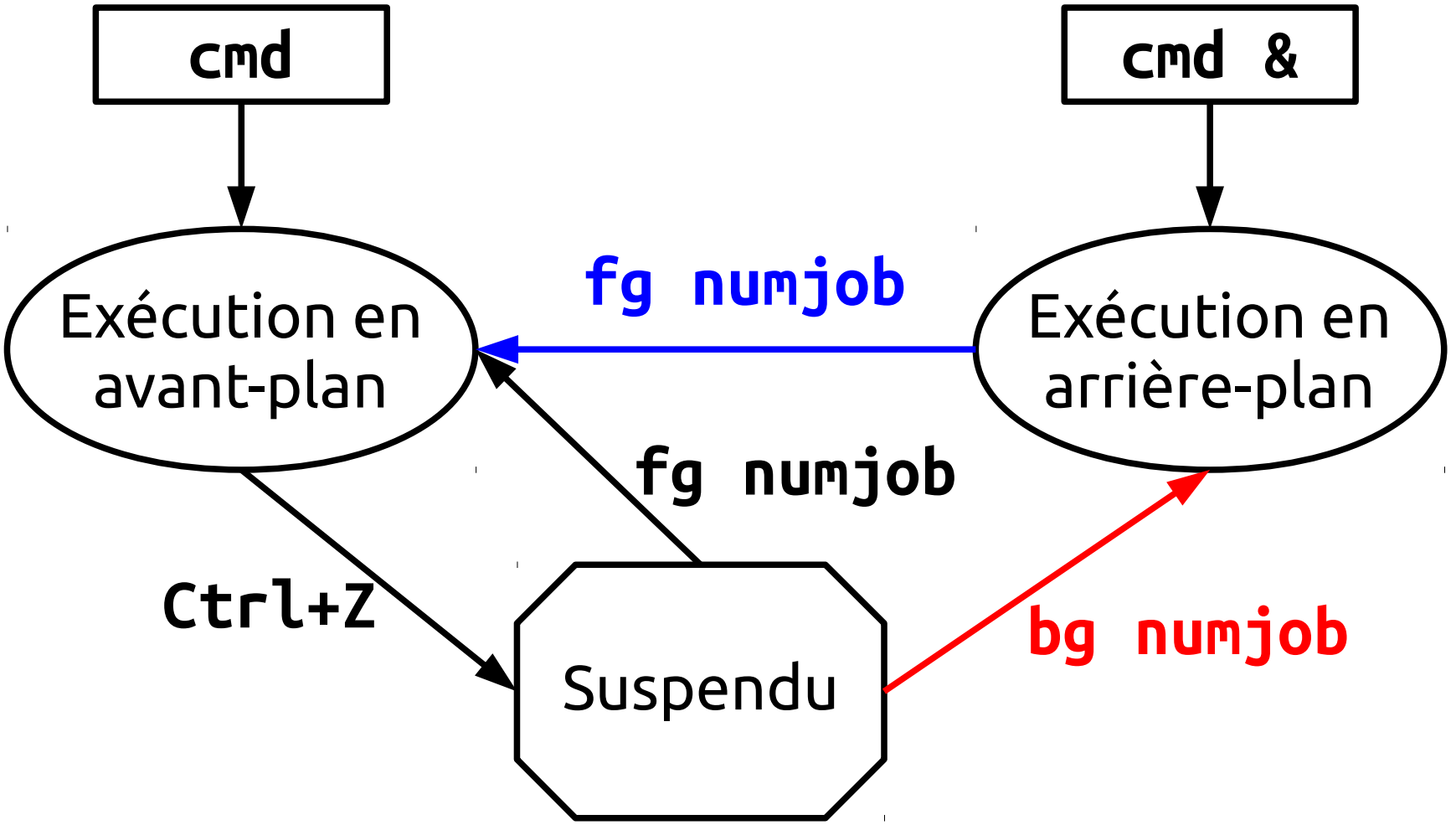
- Attention : Le processus lancé en arrière plan ne doit pas être interactif.
  - Il ne doit pas attendre de saisie au risque de confusion entre le processus et le Shell.
  - Il ne doit pas afficher de résultats sur l'écran au risque d'avoir des affichages en conflit avec celui du Shell.
- Quand on quitte le Shell, on quitte aussi tous ses processus fils. Il ne faut pas quitter le Shell pendant un traitement important.

Jobs

```
etudiant@lpi:~$ sleep 60 &
[1] 2142
[Numéro de job] PID
```

- **PID** : identifiant donné par le système.
- **Numéro de job** : identifiant donné par le Shell.
- En plus de la gestion des processus spécifique au système, le Shell introduit un niveau supplémentaire de contrôle de processus.
- Le Shell permet de stopper, reprendre, mettre en arrière plan, mettre en avant plan un processus, ce qui nécessite une identification supplémentaire = numéro de job.

Avant et Arrière plan



Dans le cas d'un seul job le **numjob** n'est pas obligatoire.

jobs – bg – fg

- Lister les jobs actifs et leurs état.

`jobs`

- Déplacer un job en avant plan.

`fg <numéro_job>`

- Déplacer un job en arrière plan. Le job doit être déjà suspendu par la combinaison **`Ctrl+Z`**.

`bg <numéro_job>`

ps (I)

- Lister les processus en cours (*process status*).
- Options :
  - Par défaut lister les processus de la console actuelle.
  - **f** : Afficher plus d'informations.
  - **e** : Afficher tous les processus du système.
  - **u** : Filtrer par un ou plusieurs utilisateurs (liste séparée par virgule). **g** : Pour les groupes. **t** : Pour les terminaux. **p** Pour des PID précis.
  - **l** : Afficher encore plus de détails.

ps (II)

Colonne	Signification
UID	Identifiant de l'utilisateur.
PID	Identifiant du processus.
PPID	Identifiant du processus parent.
C	Facteur de priorité, plus la valeur est grande plus la priorité est élevée.
STIME	Heure du lancement du processus.
TTY	Nom du terminal du lancement du processus.
TIME	Durée de traitement du processus.
CMD	Commande exécutée.
S	État du processus. S *sleeping*, R *running*, Z *zombie*.
PRI	Priorité du processus.
NI	Facteur de modification de la priorité.

top

- Afficher et gérer les processus en temps réel.

`top`

- Commandes interactives dans **`top`** :
  - **`N`** : classer les processus par PID.
  - **`A`** : classer les processus dans l'ordre chronologique (les plus récents en premier).
  - **`P`** : classer les processus par rapport au pourcentage d'utilisation du processeur.
  - **`M`** : classer les processus par rapport au pourcentage d'utilisation de la mémoire.
  - **`k`** : Envoyer un signal à un processus (le PID sera demandé).
  - **`q`** : quitter **`top`**.

Signaux (I)

- Lorsqu'un processus tourne en arrière plan, il ne peux pas réagir aux combinaisons de touches (pour l'arrêter ou le terminer), sauf en utilisant les jobs avec fg et bg.
- Il peut être nécessaire d'envoyer des signaux à un processus auxquels il pourra éventuellement réagir.
- Un signal est aussi un des moyens de communication entre les processus.
- Lorsqu'on envoie un signal à un processus, il doit l'intercepter et réagir en fonction de celui-ci. Certains signaux peuvent être ignorés, d'autres non.
- Dans GNU/Linux on dispose de 64 signaux.

kill

- Envoyer un signal à un processus.

`kill -Signal PIDs`

- Arguments :
  - Il existe deux commandes **`kill`** : une interne et une externe.
  - La commande interne accepte le numéro de job en plus du **`PID`**.

`kill -Signal %jobs`

- Option :
  - Le signal peut être indiqué par son numéro ou son nom.
  - Manuel des signaux : **`man 7 signal`**
  - **`l`** :Obtenir la liste des signaux.

Signaux (II)

N°	Nom	Description
1	**SIGHUP**	*Hang Up*, envoyé par le parent à tous ses fils lorsqu'il se termine.
2	**SIGINT**	Interruption du processus depuis le clavier (**Ctrl+C**).
9	**SIGKILL**	Signal ne pouvant être ignoré. Force le processus à finir son exécution *brutalement*.
15	**SIGTERM**	Signal par défaut. Demande au processus de se terminer normalement.
17	**SIGCHLD**	Envoyé par le fils à son parent lorsqu'il se termine.
18	**SIGCONT**	Demande de continuer après arrêt.
19	**SIGSTOP**	Demande de suspension/arrêt. Signal ne pouvant être ignoré.
20	**SIGTSTP**	Demande de suspension/arrêt depuis le clavier (**Ctrl+Z**).

nohup

- Demander au processus d'ignorer le signal **1**, **SIGHUP**.

```
nohup cmd &
```

- Quand le Shell est quitté le signal **SIGHUP** est envoyé a tous les processus fils pour qu'ils se terminent.
- Lorsqu'un traitement long est lancé et que l'utilisateur quitte le Shell, ce traitement sera alors arrêté et il faudra tout recommencer.
- Par défaut les canaux de sortie et d'erreur sont redirigés vers un fichier **nohup.out**, sauf si la redirection est explicitement précisée.

Parent et Fils

- Quand un fils se termine, il envoie le signal **SIGCHLD** à son parent. Le parent doit obtenir autant de **SIGCHLD** qu'il a eu de fils ou émis de **SIGHUP**.
- Si le parent ne traite pas le signal **SIGCHLD** de ses fils ceux-ci deviennent des **zombis**.
- Un processus zombi ne consomme aucune ressource mais continue à occuper une entrée dans la table des processus.
- C'est le processus **init** qui récupère les processus zombis. Ces processus finiront par disparaître.

killall

- Envoyer un signal à tous les processus relatif à une commande (nom).

`killall options nom`

- Options :
  - **`l`** : lister les signaux.
  - **`s`** : le signal à envoyer. Par défaut **`SIGTERM`**.

free

- Afficher la quantité de mémoire libre et utilisée du système

`free`

- Options :
  - **`b`**, **`k`**, **`m`**, **`g`** : Afficher les champs en, respectivement, octet, kilo-octet, méga-octet, giga-octet.
  - **`h`** : Afficher les champs automatiquement à l'échelle la plus appropriée.

time

- Mesurer la durée d'exécution d'une commande.

`time commande`

- Résultat :
  - **`real`** : durée totale d'exécution de la commande.
  - **`user`** : durée du temps processeur nécessaire pour exécuter la commande.
  - **`sys`** : durée du temps processeur nécessaire pour exécuter les appels système au sein de la commande.
- Exemple :
  - **`time ls -lR /home > /dev/null`**

uptime

- Afficher la durée depuis quand le système a été démarré.

`uptime`

- Option :
  - **`p`** : Affichage plus élégant (*pretty*).

pgrep

- Rechercher des processus selon des critères.

`pgrep options motif`

- Options :
  - **`u`** : filtrer par utilisateur (nom ou UID).
  - **`G`** : filtrer par groupe (nom ou GID).
  - **`t`** : filtrer par terminal.
  - **`l`** : afficher le nom du processus et son PID.
- Exemples :
  - **`pgrep -u root`**
  - **`pgrep -lu 1000`**
  - **`pgrep -lt pts/1`**

pkill

- Envoyer un signal à des processus selon des critères.

`pkill options motif`

- Options :
  - **`signal`** : le signal à envoyer. Par défaut **`SIGTERM`**.
  - **`u`** : filtrer par utilisateur (nom ou UID).
  - **`G`** : filtrer par groupe (nom ou GID).
  - **`t`** : filtrer par terminal.
- Exemples :
  - **`pkill -u 1010`**
  - **`pkill -t pts/2`**
  - **`pkill -9 -t pts/2`**

screen (I)

- Multiplexeur de terminaux :
  - Ouvrir plusieurs terminaux dans une même console ;
  - Attacher et détacher une session à une console ;
  - Partager un terminal avec d'autres utilisateurs.
- Dispose d'un mode de commande interne : **Ctrl+a**
- Créer un screen :

```
screen -S [nom]
```

- Lister les screen existants :

```
screen -ls
```

screen (II)

- Rattacher un screen existant :

`screen -r [nom]`

- Avoir plusieurs attachements simultanés :

`screen -x [nom]`

- Commandes screen **`Ctrl+a`** :
  - **`c`** : Créer un nouveau terminal ;
  - **`d`** : Se détacher du screen ;
  - **`n`**, **`p`** : Aller au terminal suivant « ***n****ext* » ou précédent « ***p****revious* » ;
  - **`0...9`** : Aller au terminal numéro ;
  - **`"`** : Lister les terminaux avec la possibilité de choisir un.
  - **`A`** : Nommer les terminaux.

screen – Exemple (I)

- Dans un terminal
  - **`screen -S lpi1`**
  - **`Exécution d'un programme`**
  - **`Ctrl+a d`**
  - **`screen -ls`**
  - **`screen -r lpi1`**
  - **`Ctrl+d ou exit`**

screen – Exemple (II)

- Terminal 1
  - **screen -S lpi2**
  - **Exécution d'un programme**
  - **Ctrl+a d**
  - **screen -ls**
  - **screen -x lpi2**
  - **...**
- Terminal 2
  - **screen -ls**
  - **screen -x lpi2**
  - **...**
  - **Ctrl+d**

screen – Exemple (III)

- Terminal 1 – user1
  - **screen -S lpi3**
  - **Ctrl+a : multiuser on**
  - **Ctrl+a : acladd user2**
  - **Exécution d'un programme**
  - **...**

- Terminal 2 – user2
  - **screen -x user1/lpi3**
  - **....**
  - **Ctrl+d**