

Cours 3 : Jobs d'été et TDs (travaux différés)

Christophe Gonzales

31015 — Principes et pratiques de l'administration des systèmes

1 Processus et jobs

Les pipelines

pipeline (définition du manuel du bash)

- **pipeline** = séquence de commandes simples séparées par |
- Dans un pipeline, commandes exécutées dans leur propre sous-shell
- Valeur de retour du pipeline = celle de la dernière commande
- format d'un pipeline :
`[time [-p]] [!] command1 [| command2 ...]`

Exemples :

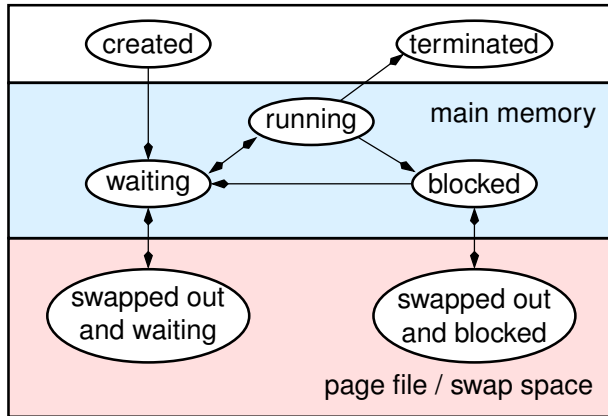
```
[gonzales@msLDAP /]$ ls -l | grep toto  
[gonzales@msLDAP /]$ ls -l
```

Les processus

Process

- **process** = instance d'un programme en exécution
- contient les ressources :
 - le code machine exécuté
 - une zone mémoire réservée
 - des descripteurs de ressources (fichiers, etc.)
 - des attributs de sécurité (permissions)
 - état (registre processeur, etc.)

Les états des processus (1/2)



- 1 création ⇒ chargement en mémoire ⇒ en attente
- 2 scheduler ⇒ exécution
- 3 attente de ressources ⇒ état bloqué
- 4 fin d'exécution : terminated

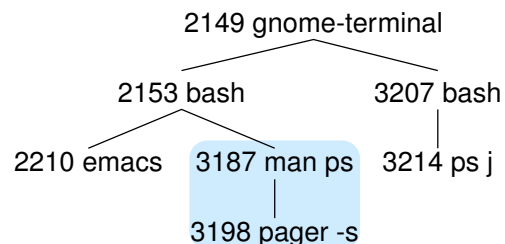
Les états des processus (2/2)

```
[gonzales@msLDAP /]$ ps u
USER      PID  %CPU %MEM  VSZ   RSS  TTY   STAT  START  TIME  COMMAND
gonzales  1910  0.0  0.0  108436 1848 pts/0  Ss   12 :59 0 :00  bash
gonzales  1932  1.0  0.0  108088 1080 pts/0  R+   13 :00 0 :00  ps u
```

sigle	signification
D	Uninterruptible sleep (usually IO)
R	Running or runnable (on run queue)
S	Interruptible sleep (waiting for an event to complete)
T	Stopped, either by a job control signal or because it is being traced
X	dead (should never be seen)
Z	Defunct ("zombie") process, terminated but not reaped by its parent

Arbres de processus

```
[gonzales@msLDAP /]$ ps j
PPID PID  PGID SID  TTY   TPGID  STAT  UID  TIME  COMMAND
2149 2153 2153 2153 pts/0 3187  Ss   500 0 :00  bash
2153 2210 2210 2153 pts/0 3187  S    500 0 :13  emacs cours5.tex
2153 3187 3187 2153 pts/0 3187  S+   500 0 :00  man ps
3187 3198 3187 2153 pts/0 3187  S+   500 0 :00  pager -s
2149 3207 3207 3207 pts/2 3214  Ss   500 0 :00  bash
3207 3214 3214 3207 pts/2 3214  R+   500 0 :00  ps j
```



Effective UID (1/3)

UIDs associés à un process

Process UNIX X ⇒ 3 UID :

- **Real UID** (RUID) = UID de l'utilisateur ou du process qui a créé X

Peut être modifié seulement si EUID = 0

- **Effective UID** (EUID) = l'UID utilisé pour évaluer les droits de X à exécuter certaines actions

$EUID \neq 0 \Rightarrow EUID$ peut être modifié en RUID ou SUID

$EUID = 0 \Rightarrow EUID$ peut être changé en n'importe quoi

- **Saved UID** (SUID) = $\begin{cases} \text{UID du propriétaire de X si bit SUID on} \\ \text{RUID sinon} \end{cases}$

- process « usuels » : exécutés sous l'UID de l'utilisateur
- programmes SUID permettent des accès privilégiés

```
change_id.c
#include <sys/types.h>
#include <unistd.h>

int main () {
    seteuid(502);
    getchar();
    return 0;
}
```

```
[root@msLDAP /]# gcc -o change_uid change_uid.c
[root@msLDAP /]# chmod u+s change_uid
[root@msLDAP /]# exit
[gonzales@msLDAP /]$ ./change_uid
```

```
RUID EUID SUID COMMAND
500 500 500 bash
500 502 0 /change_uid
500 500 500 ps -eo ruid,euid,suid,command
```

job

- **Job** = ensemble de processus :
 - comportant un pipeline
 - tous les processus descendant de lui
 - qui appartiennent au même **groupe de process**
- job \Rightarrow numéro de job : JID (Job Identifier)
- la commande `jobs` permet de lister les jobs

contrôle d'un job

- contrôle d'un job \Rightarrow suspendre/continuer l'exécution de processus.
- Le système d'exploitation maintient la notion d'ID de groupe de processus du terminal courant.
- Les processus dont l'ID correspond à celui-ci reçoivent les signaux du clavier tels que SIGINT \Rightarrow **en foreground**.
- Les autres sont en **background** et ne reçoivent pas les signaux du clavier.
- Seuls les process en foreground peuvent lire et écrire sur le terminal.

Le contrôle des jobs (2/6)

Commandes de contrôle :

- **CTRL-Z** lorsqu'un process tourne :
⇒ stoppe le process et redonne la main au bash
- **CTRL-Y** lorsqu'un process tourne :
⇒ stoppe le process lorsque celui-ci essaye de lire à partir du terminal, et rend la main au bash
- **bg** : continuer le job en arrière plan
fg : continuer le job en avant-plan
kill : envoyer un signal à un process/un job
- Exécuter un process en terminant la commande par
<& > lance celui-ci en arrière-plan.

Le contrôle des jobs (3/6)

Exemple :

- commande `sleep` ⇒ processus sommeillant N secondes activée en mode interactif (avant-plan).
risque d'annuler l'effet d'interactivité pendant tout ce temps.
- `Ctrl-Z` suspend le processus associé.
Le contrôle est alors redonné au terminal.

```
[gonzales@msLDAP /]$ sleep 500
^Z
[1]+  stopped sleep 500
[gonzales@msLDAP /]$ ps
PID  TT  STAT  TIME  COMMAND
1020  co  TW    0:00  sleep 500
[gonzales@msLDAP /]$ jobs
[1]+  stopped sleep 500
```

Le contrôle des jobs (4/6)

Comment désigner un job donné : Le caractère « % »

- %n = le job numéro n.
- %% et %+ = job courant, c.-a.-d. le dernier job stoppé pendant qu'il tournait en avant-plan ou lancé en arrière-plan.
- %- = job précédent.
- Un job peut être désigné en utilisant un préfixe du nom utilisé pour le lancer (ex : %em pour désigner emacs), ou une sous-chaîne apparaissant dans la commande (%?ma pour emacs).
- Les commandes `fg` et `bg` peuvent prendre comme paramètre un job identifié par l'un des % ci-dessus.

Le contrôle des jobs (5/6)

```
[gonzales@msLDAP /]$ emacs
^Z
[1]+  Stopped      emacs
[gonzales@msLDAP /]$ sleep 500 &
[2] 20253
[gonzales@msLDAP /]$ jobs -l
[1]+ 20251 Stopped      emacs
[2]- 20253 Running    sleep 500 &
[gonzales@msLDAP /]$ bg %em
[1]+  emacs &
[gonzales@msLDAP /]$ jobs -l
[1]- 20251 Running    emacs &
[2]+ 20253 Running    sleep 500 &
[gonzales@msLDAP /]$ fg %2
[gonzales@msLDAP /]$ sleep 500
^C
[gonzales@msLDAP /]$ jobs -l
[1]+ 20251 Running    emacs &
[gonzales@msLDAP /]$ kill %1 ; jobs -l
[1]+  Terminated  emacs
```

Le contrôle des jobs (6/6)

Quelques caractères de contrôle (certains générateurs de signaux) :

Ctrl	nom	action
~C	interrupt	envoie le signal SIGINT au process en cours
~Z	suspend	envoie le signal SIGTSTP au process en cours
~\	quit	envoie le signal SIGQUIT au process en cours
~S	stop	arrête le défilement de l'écran.
~Q	start	reprend le défilement de l'écran.

```
[gonzales@msLDAP /]$ emacs & sleep 500 &
[1] 20856
[2] 20858
[gonzales@msLDAP /]$ jobs
[1]- Running      emacs &
[2]+ Running      sleep 500 &
[gonzales@msLDAP /]$ kill -INT %2
[2]+ Interrupt    sleep 500
[gonzales@msLDAP /]$ jobs [1]+  Running      emacs &
```

Priorité des processus/jobs

possibilité de donner des priorités aux process

- nice -n priorité commande
 - priorité $\in \{-19, \dots, 19\}$
 - -19 = le plus prioritaire
- renice -n priorité -p pid

SIGHUP

- SIGHUP = signal HangUP
- terminaison de bash \Rightarrow envoi SIGHUP à tous les jobs
 - \Rightarrow logout \Rightarrow envoi SIGHUP
- comportement normal : terminaison du job

nohup et screen

- nohup commande
 - ⚠ entrées/sorties standard/d'erreur !
- screen : multiplexeur de terminaux en mode texte
 - persistant aux déconnexions/reconnexions
 - partage de sessions
 - \Rightarrow session utilisable *simultanément* par plusieurs ordinateurs simultanément
 - partage de terminaux
 - \Rightarrow plusieurs terminaux dans un même screen
 - entrées/sorties standards

2 Travaux différés

Les travaux différés

3 manières d'exécuter des travaux en différé :

- **at** (arbitrary time) : exécute une commande à une date précise
- **batch** : exécution quand le niveau d'utilisation du système le permet (< 80%)
- **cron** (chronograph) : exécution à intervalles réguliers

- at et batch dépendent du démon atd (/usr/sbin/atd).
- cron dépend du démon crond (/usr/sbin/crond).

La commande at

invocation de la commande at

at -f script TIME où TIME vaut :

- HH:MM : aujourd'hui si heure est venue, sinon le lendemain
- midnight, noon, teatime (4 heures de l'après midi) ;
- une heure suivie de AM ou PM pour le matin ou l'après midi ;
- une date MMDDYY ou MM/DD/YY ou MM.DD.YY.
La date doit être écrite après l'heure ;
- now + N unités de temps, où unité de temps = minutes, hours, days, weeks ;
- heure suffixée de today ou de tomorrow.

Exemple :

```
at -f job 4pm + 3 days
at -f job 10:00am July 31
at -f job 10:00am 07/31/10
at -f job 01:00 tomorrow
```

at, queues, atq et atrm (1/2)

- **at -f script TIME** exécute commandes du fichier script.
at TIME lit les commandes sur l'entrée standard.
- Commandes placées dans une queue.
Le démon atd vérifie régulièrement l'état des queues et exécute les commandes qui doivent être exécutées.
- Possibilité d'utiliser plusieurs queues at -q queue TIME avec des priorités différentes.
 - queue désignée par une lettre de a à z, ou bien A à Z.
 - La queue a est celle par défaut de at, et la b celle de batch.
 - La priorité d'exécution baisse avec la lettre.
 - Queue spéciale < = > pour les jobs en cours d'exécution.
- possibilité de visualiser les queues avec la commande atq.
- possibilité de supprimer un travail en queue avec atrm.

```
[gonzales@msLDAP /]$ atq
[jonzales@msLDAP /]$ echo 'echo toto' > toto && at -f toto -q
job 6 at 2007-03-21 19:14
[jonzales@msLDAP /]$ echo 'echo toto' | at now + 22 minutes
job 7 at 2007-03-21 19:25
[jonzales@msLDAP /]$ atq
6      2007-03-21 19:14 c gonzales
7      2007-03-21 19:25 a gonzales
[jonzales@msLDAP /]$ ls -l /var/spool/at
total 16
-rwx----- 1 gonzales users 2557 Mar 21 19:03 a00007012ab0f1
-rwx----- 1 gonzales users 2557 Mar 21 19:03 c00006012ab0e6
drwx----- 2 daemon  daemon 4096 Mar 21 18:44 spool
```

- le *working directory*, l'environnement (à part TERM, DISPLAY et _), et le umask utilisés par les travaux exécutés sont ceux au moment de l'invocation de at.
- les affichages sur stdout et stderr sont envoyés par mail à l'utilisateur invoquant at en utilisant la commande /usr/sbin/sendmail.
- Sécurité :
 - root peut invoquer les commandes at et batch
 - Si le fichier /etc/at.allow existe, seuls les utilisateurs mentionnés dans ce fichier peuvent exécuter at et/ou batch
Syntaxe du fichier : un seul nom par ligne, pas d'espace
 - Si /etc/at.allow n'existe pas, tout utilisateur n'appartenant pas à /etc/at.deny peut exécuter at et/ou batch.
- lire man 1 at et man 8 atd

Cron

cron (chronograph) : permet d'exécuter des commandes à intervalles réguliers

Principe d'utilisation :

- la crontab contient les commandes à exécuter
- /usr/bin/crontab -e ⇒ éditer pour ajouter, supprimer, modifier des commandes à exécuter
- /usr/bin/crontab -l ⇒ lister la crontab de l'utilisateur
- le démon crond (/usr/sbin/crond) vérifie régulièrement s'il doit exécuter des process et, le cas échéant, les exécute

- Démon crond lancé au démarrage de linux
- crond recherche dans /var/spool/cron les fichiers de crontab des utilisateurs (nommés d'après leur login)
- il examine le fichier /etc/crontab et ceux du répertoire /etc/cron.d
- crond se réveille toutes les minutes et vérifie si, d'après ces tables, certains processus doivent être exécutés.
Les affichages de ceux-ci sont envoyés par mail au propriétaire de la crontab
- crond vérifie les changements dans les différentes crontabs toutes les minutes

la syntaxe de crontab (1/2)

- chaque ligne de job a le format :
minute heure jour-du-mois mois jour-de-la-semaine commande
- valeurs des champs :

champ	valeurs possibles
minute	0-59
heure	0-23
jour-du-mois	1-31
mois	1-12 (ou les 3 premières lettres du mois (jan, feb))
jour-de-la-semaine	0-7 (0 ou 7 = dimanche)

- des listes de valeurs pour chaque champ, séparées par des « , »
- la notation « - » \implies intervalles. Exemple : 8-11 \iff 8,9,10,11.
- la notation « /nombre » \implies pas des intervalles.
Exemple : 0-11/3 \iff 0,3,6,9.
- une astérisque = toutes les valeurs possibles.
« */nombre » pour spécifier un pas.

la syntaxe de crontab (2/2)

- Les lignes vides ou commençant par une espace ou une tabulation sont ignorées
- Les lignes commençant par un « # » sont ignorées
- Une ligne active est soit une ligne de commande, soit une définition d'environnement de la forme : nom = valeur
- crond définit automatiquement les variables LOGNAME et HOME à partir de l'/etc/passwd.
HOME et SHELL peuvent être redéfinis.
- si la variable MAILTO est définie, tous les affichages provoqués par les exécutions de processus seront envoyés par mail.

Exemple de crontab

```
# utiliser /bin/tcsh au lieu de /bin/sh pour exécuter les commandes
SHELL=/bin/tcsh

# envoyer les affichages a toto
MAILTO=toto@titi.fr

# exécuter une commande tous les jours,
# 5 minutes après minuit
5 0 * * *      $HOME/bin/daily.job >> $HOME/tmp/out 2>&1

# exécuter une commande à 14h15 le 1er jour de chaque mois
15 14 1 * *    $HOME/bin/monthly

# exécuter une commande à 22h00 tous les jours de la
# semaine excepté le week-end
0 22 * * 1-5   mail -s "It's 10pm" joe%Joe,%%coucou%

# exécution toutes les 2 heures, tous les jours
23 0-23/2 * * * echo "run 23 minutes after midn, 2am, 4am..."

# exécution tous les dimanches à 4h05
5 4 * * sun    echo "run at 5 after 4 every sunday"
```

Le démon crond (fin)

- Syntaxe des fichiers cron de /etc légèrement différente de celle des crontabs des utilisateurs.
- crond utilise /var/spool/cron pour les crontabs des utilisateurs
- possibilité d'interdire des soumissions de travaux avec les fichiers /etc/cron.allow et /etc/cron.deny
- lire man 5 crontab, man 1 crontab, man 8 cron
- utiliser anacron pour les jobs à exécuter périodiquement (avec des fréquences en termes de jours) sur des machines qui ne sont pas allumées 24h/24.