


# Chapitre 13 : Services

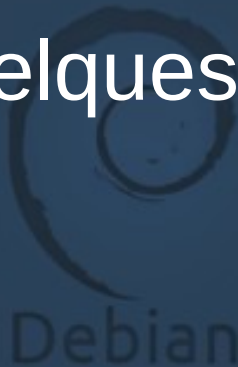
Gestion des services système

The Debian logo, which is a stylized white swirl or 'D' shape, is positioned behind the text 'Gestion des services système'.

Debian

# Plan de ce chapitre

- Explications sur les services en général
  - Pilotage des services
  - Démarrage et arrêt de la machine
- Présentation de quelques services utiles




# Présentation

- Le système Unix repose sur de nombreux « services ». Quelques exemples :
  - Service de connexion aux comptes
  - Services de tâches planifiées : à telle heure faire...
  - Services pour enregistrer les messages d'erreurs
  - Contrôle de l'alimentation (ACPI) : veille, reboot...
- Leur rôle : faire « vivre » le système. Sans eux, il n'y aurait que des commandes à taper, rien d'automatique et de réactif.

# 13.1 – Services

Un service = un démon + un lanceur

The Debian logo, which is a stylized spiral, is positioned behind the text. The word "Debian" is written in a light blue font below the spiral.

Debian

# Service Unix

- Un service est généralement composé de :
  - Un programme appelé « **démon** » (*daemon*) qui tourne en permanence et qui traite les demandes qui arrivent. Ce programme est généralement situé dans /usr/bin, ou parfois /sbin.
  - Un **script de contrôle** (démarrage, arrêt, état)
  - Des fichiers de configuration
  - Des fichiers d'état ou de données

# Le démon

- C'est un programme qui tourne en permanence et en arrière-plan
- Il surveille certaines sources d'informations :
  - Port réseau (*socket*)
  - Voie de communication interne (*bus logique*)
  - Horloge interne, Fichiers...
  - ...
- Son travail est généralement défini par un fichier de configuration dans **/etc**

# Le script de contrôle

- Les services sont activés ou arrêtés par des scripts situés dans `/etc/init.d`
- Ce sont des scripts bash, un par service, qu'on lance en fournissant un mot clé en paramètre :
  - start** => le service démarre : `demon &`
  - stop** => le service s'arrête : `kill $pid_demon`
  - status** => le service affiche son état
  - restart** => il s'arrête puis redémarre
  - reload** => il relit son fichier de configuration

# Exemples

- Pour démarrer le service d'impression :  
`sudo /etc/init.d/cups start`
  - Pour arrêter le partage de fichiers avec Windows :  
`sudo /etc/init.d/samba stop`
  - Pour redémarrer le gestionnaire de connexion :  
`sudo /etc/init.d/gdm3 restart`
- NB** : voir plus loin, la commande **service**



# Exemple de service minimal

- Voici un mini script de lancement d'un démon. On doit l'appeler en faisant :  
    /etc/init.d/contrôle **start** ou **stop**
- On voit qu'il lance /usr/bin/demon en arrière plan, ou au contraire qu'il tue ce processus :

```
#!/bin/bash
case $1 in
    start) /usr/bin/demon & ;;
    stop)  killall -9 demon ;;
esac
```

# Stockage du PID

- Souvent, les lanceurs de services mémorisent le PID du démon dans un fichier de `/var/run`  
Ex : `/var/run/demon.pid`
- Cela permet de cibler le démon à tuer, plutôt que faire un `killall` hasardeux.

Debian

# Commande service

- Au lieu de lancer le script de contrôle directement, il y a une commande appelée service :

```
sudo service samba start
```

```
sudo service network-manager stop
```

```
sudo service transmission-daemon status
```

- Elle vérifie que le service existe et appelle son script de contrôle, ex : /etc/init.d/samba

# Services et boot du système

Les services sont liés à la séquence de démarrage du système :

- 1) Chargement du noyau (décompression et exécution de `/boot/vmlinuz` ou `bzImage` et décompression de `/boot/initrd`)
  - `/boot/vmlinuz` ou `bzImage` = noyau
  - `/boot/initrd` = disque virtuel (ram disk) initial permettant de monter `/dev/hda1` ou `/dev/sda1` par la suite
- 2) Test du matériel
- 3) Démarrage des services, dont celui de connexion

# Démarrage des services

- Les gros ordinateurs ont plusieurs modes de fonctionnement :
  - Mode multi-utilisateur : les connexions de plusieurs utilisateurs sont possibles
  - Mode mono-utilisateur : seul l'administrateur peut se connecter
- Ces modes sont (mal) nommés **runlevels**

NB : cette notion de « runlevel » est héritée d'un passé lointain et commence à évoluer, voir plus loin.

# Runlevel

- Un runlevel est identifié par un chiffre ou une lettre. Voici ceux qui sont normalisés :
  - S** : il correspond au mode sans réseau et monoutilisateur (root est le seul connecté pour la maintenance), c'est aussi le mode du démarrage
  - 0** : ce runlevel fait éteindre la machine
  - 1** : mode mono-utilisateur local, pas de connexion par réseau ni d'invite graphique (gdm)
  - 2** : c'est le mode normal sur Debian, Ubuntu
  - 6** : ce runlevel fait redémarrer la machine

# Changement de runlevel

- Un runlevel définit une liste de services qui sont lancés ou arrêtés, exemple :
  - Dans le runlevel 1, sur Debian, la machine est en mode mono-utilisateur : on ne lance pas le service de connexion ailleurs que sur la console
  - Dans le runlevel 2, la machine est en mode multi-utilisateurs, il y a des services de login (ex : gdm et getty)
- Si on change de runlevel, certains services sont démarrés et d'autres arrêtés

# Commandes

- La commande `runlevel` affiche le runlevel précédent (N=aucun) et le runlevel actuel

Ex : elle affiche N 2

- La commande `telinit` fait changer de runlevel

Ex : `sudo telinit 0`

- Attention au runlevel qu'on demande (ex : si on est connecté par ssh et qu'on coupe le réseau)



# Lancement des services

- Il y a un mécanisme qui lance les services une fois le matériel initialisé, selon la version du système considéré :
  - Le bon vieux « init » hérité de Unix System V mais qui commence à disparaître
  - Un mécanisme appelé « upstart » mais qui est déjà dépassé,
  - Le mécanisme « systemd » qu'on trouve maintenant dans tous les Linux modernes
- Ils sont encore compatibles

# Mécanisme init (System V)

- La séquence de démarrage est définie par le fichier `/etc/inittab`
  - `label:runlevels:condition:action`
    - label = identifie la ligne, quelconque mais unique
    - runlevels = vide (tous) ou noms des RL concernés
    - condition = circonstance dans laquelle on lance l'action, ex : sysinit, ctrlaltdel, powerfailnow...
- Ce fichier est consulté par le processus init au moment du démarrage

# Exemples de directives inittab

- `id:2:initdefault:`  
fixe le runlevel par défaut
- `si::sysinit:/etc/init.d/rcS`  
indique un script de configuration initiale
- `l2:2:wait:/etc/init.d/rc 2`  
le runlevel 2 fait exécuter `/etc/init.d/rc 2`
- `1:2345:respawn:/sbin/getty tty1`  
propose une connexion en mode texte

# Script `/etc/init.d/rc`

- Ce script reçoit le runlevel dans `$1`, ex : 2 et fait ceci (simplification du cœur du script) :

```
for s in /etc/rc$1.d/K*
do
    $s stop
done
for s in /etc/rc$1.d/S*
do
    $s start
done
```

# Explications

- Le dossier `/etc/rc2.d` contient des liens logiques vers le dossier `/etc/init.d`
  - Ex : `S17cron` est un lien vers `/etc/init.d/cron`
- Ces liens sont nommés `KNNnom` s'il faut arrêter le service `nom` et `SNNnom` s'il faut démarrer ce service, `NN` est le rang (ordre)
- La boucle `for s in /etc/rc2.d/S*` va lancer le script `S17cron` avec le paramètre `start` => ce service va démarrer

# Gestion des services

- Si on crée un nouveau service, on doit le rajouter dans le runlevel courant

```
sudo update-rc.d service defaults
```

Ajoute ce service dans les runlevels

- Il y a des options pour affiner l'ajout : priorité, runlevels concernés...

- ```
sudo update-rc.d -f service remove
```

Enlève ce service de tous les runlevels

# Le mécanisme systemd

- Dans les Linux modernes (Debian 8, Fedora...), c'est **systemd** qui gère les services et l'état de la machine
- Ce dispositif examine l'ensemble des scripts de gestion des services et détermine l'ordre dans lequel les démarrer
  - Ex : démarrer le partage Windows seulement quand le réseau est actif
  - Ex : démarrer le réseau seulement quand le matériel est initialisé

# Entête LSB

- Il faut rajouter quelques lignes appelées entête LSB au début d'un script de gestion de service :

```
### BEGIN INIT INFO
# Provides:                mondemon
# Required-Start:          $remote_fs $syslog
# Required-Stop:           $remote_fs $syslog
# Default-Start:           2 3 4 5
# Default-Stop:            0 1 6
# Short-Description:       essai
# Description:              un service de test
### END INIT INFO
```




# Entête LSB

- Ces lignes disent :
  - Voici le nom de ce service
  - Avant de le lancer, il faut auparavant que tels et tels services soient lancés
  - Il faut l'arrêter avant d'arrêter tels et tels autres
  - Il démarre dans tels et tels runlevels
  - Il s'arrête quand on passe dans tels et tels runlevels
  - Voici sa description

# 13.2 – Création d'un service

Un service = un démon + un lanceur

The Debian logo, which consists of a stylized spiral or swirl shape, is positioned behind the text 'Un service = un démon + un lanceur'.

Debian

# Démarche


- Créer un programme ou script **mondemon**, le placer dans **/usr/bin**, ou /usr/sbin
- Éditer une copie (appelée mondemon aussi) du script **/etc/init.d/skeleton**, c'est le lanceur du démon
  - Modifier les descriptions et informations dans le LSB
  - Modifier les chemins et les noms
  - Modifier le lancement, les options...

# Mise en place

- Tester démarrage, arrêt et état :
  - `service mondemon start`
  - `service mondemon status`
  - `service mondemon stop`
- Utiliser `update-rc.d` pour mettre en place ce service dans les runlevels

# 13.3 – Étude de quelques services

Tâches planifiées et messages systèmes

The Debian logo, which is a stylized swirl or 'D' shape, is positioned behind the text 'Tâches planifiées et messages systèmes'.

Debian

# Cron

- Cron est un service qui exécute des commandes (ou scripts) à des dates et heures régulières, par exemple tous les jours, ou tous les samedi, ou chaque heure...
  - Le grain le plus fin est la minute
- On définit l'emploi du temps directement dans le fichier `/etc/crontab` ou avec la commande `crontab -e`

# /etc/crontab

- Chaque ligne = une tâche planifiée
  - Colonne 1 : minutes 0-59, mettre \* si toutes
  - Colonne 2 : heures 0-23, mettre \* si toutes
  - Colonne 3 : jour 1-31, mettre \* si tous
  - Colonne 4 : mois 1-12, mettre \* si tous
  - Colonne 5 : jour de semaine 0-7, dimanche=0
  - Colonnes suivantes : la commande à lancer avec ses paramètres
- Ex : **59 23 24 12 \* echo "Joyeux Noël"**

# Amélioration

- Dans certaines versions d'Unix (Debian récente), le fichier crontab est séparé en différents dossiers : /etc/cron.hourly, /etc/cron.daily... et il suffit d'y déposer un script (ou un lien vers) pour qu'il soit lancé automatiquement une fois par heure, par jour...
- NB : c'est assez facile à mettre en place (voir le TD7)



# Messages du noyau

- La commande `dmesg` affiche tous les messages du noyau (et des pilotes)
  - Initialisation et tests de la machine
  - Lancement des pilotes
  - Branchement de périphériques à chaud (clé USB)
- Le consulter si un périphérique semble ne pas fonctionner

# Messages du système

- Comment le système peut-il gérer les messages des logiciels et de ses composants ?
  - sources = services, logiciels...
  - messages = erreurs, plantages, informations, avertissements, trace de mise au point...
  - destination = écran, fichier(s), autre machine sur le réseau
- Les messages importants (ex : erreurs dans les fichiers de configuration) doivent pouvoir être récupérés : écran = pas bon, fichier = ok

# Solution Unix : syslog

- Voici comment ça marche :
  - Un logiciel (ou partie du système) veut écrire un message. Il appelle une fonction C ou une commande bash qui transmet ce message à syslog
    - Ex : `logger "démarrage du démon"`
  - Syslog est un service qui sait quoi faire avec tous les messages, exemple : enregistrer le message dans un fichier, ou l'envoyer par mail à une autre machine.
    - Ex : ce message est mis dans `/var/log/messages`

# Fichiers de log

- Quand un message est enregistré, il l'est généralement dans un fichier de /var/log :
  - /var/log/messages
  - /var/log/syslog
  - /var/log/auth.log
  - /var/log/daemon.log
  - Etc...
- C'est décidé par le fichier de configuration de syslog selon la **catégorie du message**

# Catégories des messages

- Les messages sont étiquetés par un couple :
  - **Service** (*facility*) : identifie la source du message
  - **Niveau** (*level*) : du plus grave au plus léger
- On indique ce couple à la commande logger :

```
logger -p daemon.info "arrêt du démon"  
logger -p auth.warning "login impossible"
```

# Services émetteurs

- Il y a 24 sources (*facility*) dont :
  - kern (noyau) (mais, au début du démarrage, seul dmesg est actif, syslog ne l'est pas encore)
  - user
  - mail
  - daemon (services)
  - auth (connexion)
  - et aussi local0..local7 (perso)



# Gravité du message

- La gravité a été normalisée sur 8 niveaux :
  - (0) `emerg` = le CPU fonctionne très bizarrement : en panne
  - (1) `alert` = réagir immédiatement sinon ça va devenir `emerg`
  - (2) `crit` = erreur grave, une partie du système est bloquée
  - (3) `err` = erreur logicielle, le processus a été tué
  - (4) `warning` = problème à régler
  - (5) `notice` = à voir
  - (6) `info` = message ordinaire
  - (7) `debug` = trace de mise au point

# Configuration /etc/syslog.conf

- Il est composé de règles :

*source.level destination*

- Ex :

```
daemon.* - /var/log/daemon.log
*.emerg /dev/pts/0
```

- Destination = nom complet absolu de fichier
  - S'il est précédé de « – » alors il n'y a pas d'écriture disque immédiate (plus rapide, mais message perdu si panne)



# Logrotate

- Le problème des logs = les quelques fichiers concernés deviennent énormes au bout de quelques mois :
  - Nettoyage régulier pour ne garder que les messages récents
  - Yaka : tous les lundi supprimer les fichiers... PB : on perd les messages du dimanche dès le lendemain
- Logrotate fait mieux : tapis roulant de fichiers
- On peut employer cron pour le lancer

# Algo de logrotate

- Soit un fichier à faire « tourner », ex : auth.log, sur 4 exemplaires
- On a auth.log et on garde 3 précédentes versions

```
rm auth.log.3  
mv auth.log.2 auth.log.3  
mv auth.log.1 auth.log.2  
mv auth.log auth.log.1  
touch auth.log
```

# Configuration de logrotate

- Le fichier `/etc/logrotate.conf` contient des directives

- globales :

- `weekly`, `monthly` : périodicité de rotation
- `rotate 4` : nombre de versions à garder
- `create` : recréer le fichier après rotation

- par fichier :

```
Nomdufichier {  
    Directives  
}
```