Chapitre 5 - Les processus, les services et le shell

2.0

Système d'exploitation GNU/Linux

GORGOUMACK SAMBE

1 Avril 2022

Table des matières

Objectifs

A l'issue de ce chapitre, l'apprenant doit être capable de :

1. distinguer les **principes de gestion des processus et services** par le système Linux ;
2. **manipuler/gérer** les processus et services ;
3. **distinguer** les **fonctionnalités** offertes par le **shell** ;
4. **exploiter** les **fonctionnalités** offertes par le **shell**.

Introduction

GNU/Linux est un système **multitâche**, **plusieurs taches** ou **processus** sont **gérés** au même moment par le **système**. Un **processus** ou **tache** est une **instance d'exécution d'un programme.** Les processus s'exécutent tous sur le **processeur** et se partagent des ressources : exploitation de la **mémoire vive**, récupération et renvoie de données (**entrées/sorties)**.

Certains processus, parmi lesquels ceux lancés au **démarrage** et qui assurent la **bonne marche du système**, **s'exécutent de manière permanente** en attente de sollicitations, ils sont appelés des **services** ou **démons**.

Une fois le système démarré, l'**utilisateur** peut lancer des **processus** à travers les **logiciels** de l'**interface graphique** ou à travers des **commandes** soumises au **shell**. Le **shell** réalise **certaines tâches** avant de transmettre les commandes au noyau.

Ce chapitre est relatif à la **gestion des processus par le système d'exploitation**. Nous aborderons

1. les principes de gestion des processus par le système ;
2. la manipulation des processus par l'utilisateur
3. le traitement des commandes utilisateurs par le shell.

I.La gestion des processus et services par le système

1. Définitions

Définition : Processus linux

Un **processus** est une **instance d'exécution d'un programme**. Au lancement d'un programme, le système le **charge en mémoire** et demande au processeur de l'**exécuter**. Dans un système GNU/Linux, le processus est identifié de manière unique sur le système par un **identificateur de processus** (Process identifier - PID).

Exemple : Quelques exemples de processus

Donnons quelques exemples de processus :

1. lorsque vous **lancez un programme C** compilé, il s'exécute en temps que **processus** ;
2. lorsque vous **lancez une commande** sur le terminal, elle devient un **processus** ;
3. lorsque vous **démarrez un logiciel** sur l'interface graphique, il devient un **processus**.

Définition : Service Linux

Dans le monde **Linux** on appelle **service** ou **démon** (daemon - Disk And Execution MONitor), un **processus** :

1. qui **n'est associé à aucun terminal** et qui tourne en "tâche de fond" **indépendamment des utilisateurs** qui se connectent ;
2. s'**exécute de manière permanente** en attente de **requêtes** à traiter.

Le **fonctionnement du système** est **basé** sur des **services** qui sont **lancés au démarrage**.

Exemple : Quelques exemples de services

Donnons quelques exemples de services :

1. Le **serveur web** (apache par exemple) lorsqu'il est démarré est un service ;
2. l'**interface graphique** est un **ensemble de services ;**
3. le **blutooth**, autant que les autres types de connexion, est géré par un **service.**

2. Familles de systèmes

Systèmes monotâche/systèmes multitâches

Nous distinguerons deux systèmes dans la classification des systèmes par rapport à la gestion des processus :

1. les systèmes monotâches comme MS DOS qu'on ne rencontre presque plus ;
2. les systèmes multitâches comme MS Windows, UNIX, Linux, MACOS,...

Rappelons que les machines peuvent être équipés de **plusieurs processeurs** (**systèmes multiprocesseurs**) ou d'un seul (**systèmes monoprocesseur**).

Définition : Les systèmes monotâche

Un **système monotâche** est un système qui, du point de vue **l'utilisateur,** ne peut exécuter qu'**un processus à la fois**.

Rappelons que dans un **système monotâche**, lorsqu'un **programme** est **exécuté**, le système crée un **processus** qui est une **instance d'exécution du programme**. le processus est **chargé dans la mémoire vive** et est **exécuté** par le **processeur** jusqu'à ce qu'il **se termine** avant qu'on ne puisse **lancer un autre programme**.

Dans de tels systèmes, si le processus rentre **en attente d'une ressource**, le **processeur reste inactif** et le temps d'attente serait gaspillé.

Définition : Les systèmes multitâches

Un système d'exploitation **multitâches** est un système dans lequel **plusieurs processus** peuvent s'exécuter en "**parallèle**", du point de vue de **l'utilisateur**. Avec un seul processeur, le système n'exécute en réalité qu'un processus à la fois. Le multitâche se base sur un principe : le système **alloue** un **espace mémoire** à chaque programme en cours d'exécution (processus) et **alloue** le **processeur** sur une **très courte durée de temps** (appelée **quantum**) à chaque processus, ce qui donne à l'utilisateur l'impression que plusieurs processus s'exécutent en même temps.

Chaque fois qu'un processus fini son **quantum** ou doit **entrer en attente** (par exemple d'une ressource), le système d'exploitation **retire le processeur à ce processus** pour **la donner à un autre processus "plus méritant"**.

Une des fonctions principales du système d'exploitation est la **gestion de ces processus** appartenant au **système** et à des **utilisateurs différents** avec des droits différents tout en prenant en compte la **confidentialité** et la **sécurité** du système et des données et tout en restant "**équitable**".

3. Du démarrage à l'utilisation du système : Les services...

A l'allumage d'un ordinateur, nous pouvons ressortir de manière grossière les étapes suivantes :

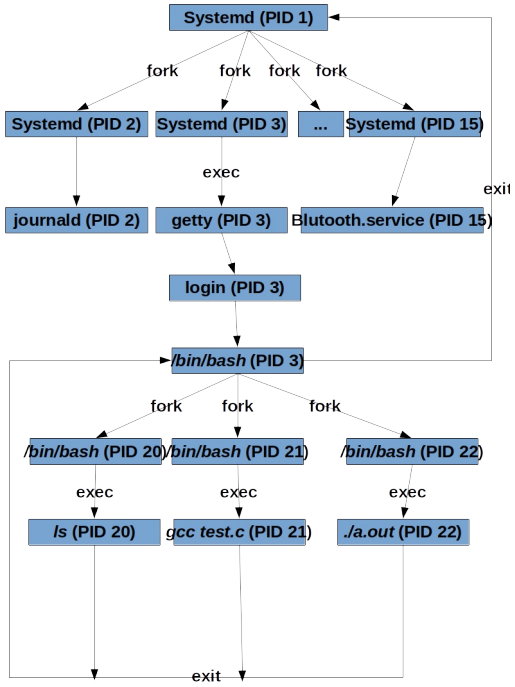
1. **exécution du BIOS**
   - **Lorsqu'un ordinateur démarre**, le **BIOS (aujourd'hui UEFI) initialise les différents composants** de l'ordinateur (processeur, mémoire vive, disque dur, carte graphique, etc.), effectue quelques vérifications basiques, détecte la partition d'amorçage (Master Boot Record), puis démarre le **chargeur d'amorçage** (GRUB par exemple);
2. **exécution du chargeur d'amorçage**
   - Le **chargeur d'amorçage** est un logiciel qui permet de lancer au choix un système parmi plusieurs installés sur un ordinateur. Il se charge du **démarrage du système GNU/Linux** ;
3. **exécution du noyau Linux**
   - Lorsque le système démarre, il fonctionne en **mode noyau** et il n'y a, en quelque sorte, qu'**un seul processus**, le **processus d'initialisation**. À la **fin de l'initialisation** du système, le **processus d'initialisation** démarre un **processus du noyau**, **systemd** (anciennement init).
4. **exécution de systemd**
   - **systemd** gére le **démarrage du système** à travers le **démarrage des différents services**. Il a pour **identifiant de processus 1** car il s'agit du **premier processus** réel du système. Il est aussi le **dernier processus à s'arrêter**, c'est lui qui contrôle tous les autres processus.
5. **lancement des services**
   - systemd est chargé de **lancer les services d'arrière-plan** (montage des périphériques de stockage, configuration du réseau, messagerie...) et l'**interface utilisateur**, qu'elle soit **graphique** ou en **mode console**. A la **différence du système de démarrage init** qui procédait à un **démarrage séquentiel des services** avec une forte utilisation des **scripts shell**, **systemd** permet le **chargement en parallèle des services** au démarrage et **réduit les appels aux scripts shell**. Il prend moins de temps pour démarrer le système.
6. **utilisation du système**

- Une fois le système bien démarré (systemd et services en exécution), il peut être utilisé : les différents utilisateurs peuvent se **connecter** (la connexion crée un processus) et **utiliser le système** (créer des processus).

4. Création des processus, sécurité et confidentialité

Création de processus

La **création de nouveaux processus** par le système se fait par un mécanisme de **clonage d'un processus existant** à travers un **appel système** (**fork** ou **clone**). À la fin de l'appel système, le nouveau processus créé sera en attente d'être exécuté par le système. Tous **les processus** du système **descendent du processus systemd** du noyau, il est ainsi **parent direct ou indirect** de **tous les processus du système**. La **plupart des services** ont pour **parent le processus de PID 1 (systemd)** qui ne meurt qu'à l'arrêt du système. Les processus forment donc une arborescence dont la racine est le processus systemd.



Tout processus Linux appartient à un **groupe de processus**, cela permet au système de pouvoir appliquer un même traitement à des processus liés tel que les processus lancés d'un même terminal.

Lorsqu'un **processus se termine proprement**, il effectue les opérations suivantes :

1. **Fermeture des fichiers** ouverts ;
2. **Libération de la mémoire** utilisée ;
3. **Envoi d'un signal** au **processus père** et aux **processus fils**.

Les **processus orphelins** sont adoptés par le processus **systemd**.

Sécurité des processus

Tout **processus** dans un système GNU/Linux a un **propriétaire**. Les **services** lancés au démarrage appartiennent soit à **root** soit à un **compte de service**. **tout processus lancé par un utilisateur appartient à l'utilisateur**. Le noyau conserve pour chaque processus l'**identité de l'utilisateur**, il **contrôle les droits** de l'utilisateur à chaque accès d'un processus à une **ressource**.

Le noyau assure une **indépendance totale des processus**, tant au niveau des **zones de mémoire** utilisées qu'au niveau de l'**accès aux ressources** de la machine. Un processus ne peut donc, par défaut, ni lire, ni écrire dans la mémoire d'un autre processus, et les ressources de calcul ou d'entrée/sortie sont partagées entre les différents processus par les mécanismes du multitâche.

5. Caractéristiques des processus

Un processus est identifié de manière unique dans le système. Du point de vue technique, un processus sur un système GNU/Linux est représenté par une structure qui contient entre autre les informations suivantes :

1. des **identifiants** : l'**identifiant du processus** (**process identifier - pid**), les **identifiants du propriétaire** qui a lancé le processus (uid,gid), les **identifiants effectifs** qui peuvent être différents des identifiants du propriétaire lorsqu'il y'a

endossement d'identité (**SUID,SGID**)...l'**identifiant de son processus père** (**process parent identifier - ppid**), l'identifiant de groupe du processus (**PGID**)...

2. l'**état du processus :**
   - **en exécution** : occupe le processeur
   - **prêt** : attend la disponibilité du processeur ;
   - **suspendu** : en attente d'une E/S (entrée/sortie) ;
   - **arrêté** : en attente d'un signal d'un autre processus ;
   - **zombie** : demande de destruction ;
3. des **pointeurs vers la mémoire virtuelle** utilisée par le processus. Cette mémoire virtuelle contiendra le code et les données du processus.
4. des **pointeurs vers des fichiers**...
5. des **informations relatives à la communication inter-processus** tel que les **signaux** et les **tubes**.
6. des **informations relatives à l'ordonnancement** telle que la **politique d'ordonnancement** et la **priorité**

Les fichiers référencés par un processus

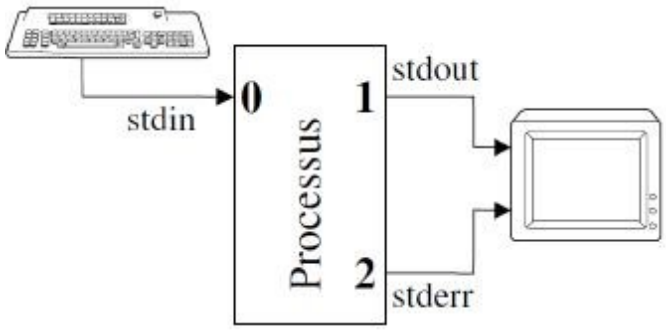
Tout **fichier** (inode) manipulé par un **processus** sur le système est **référencé** par ce dernier avec un **descripteur de fichier** (**file descriptor**). Un processus référence un certains nombre de répertoires (inode) sur le système par défaut :

1. **répertoire personnel** (home directory) ;
2. **répertoire de travail** (print working directory - pwd).

Les programmes utilisent trois ressources : ils lisent leurs entrées de l'**entrée standard** qui est par défaut le clavier, envoient leurs sorties sur la **sortie standard** qui est par défaut le terminal et envoient les messages d'erreur sur **la sortie d'erreur** qui est aussi par défaut le terminal. Ces entrées et sorties sont toutefois **traitées par le programme comme des fichiers**.

La plupart des processus référencent et ouvrent donc trois descripteurs de fichiers :

1. l'**entrée standard** qui a pour descripteur **0 ;**
2. la **sortie standard** qui a pour descripteur **1 ;**
3. la **sortie d'erreur** qui a pour descripteur **2**.



Descripteurs standards

Communication inter-processus : signal et tubes

Le noyau et les processus et les processus entre eux communiquent à travers des mécanismes tels que les signaux et les tubes :

1. Les **signaux** sont l'une des plus anciennes méthodes de communication interprocessus utilisées par les systèmes UNIX. Ils sont utilisés pour signaler des **événements asynchrones** à un ou plusieurs processus. Un signal peut être **généré** par une **interruption du clavier** ou une **condition d'erreur** telle qu'un processus qui tente d'accéder à un emplacement inexistant dans sa mémoire virtuelle. Le **noyau** et l'**administrateur** peuvent **envoyer un signal à tous les processus**, les **processus réguliers** ne peuvent envoyer des signaux qu'au **processus de même uid et gid** ou appartenant au **même groupe de processus**.
   Il existe un ensemble de **signaux définis** que le **noyau** peut générer ou qui peuvent être générés par d'autres **processus** dans le système, à condition qu'ils aient les

privilèges appropriés. La **page de manuel de signal(2)** vous donne une documentation assez conséquente et la liste des signaux de votre système. A l'exception du signal **SIGSTOP** et **SIGKILL**, tout signal peut être **bloqué par un processus**.

2. Le **tube (pipe)** est un **flux** d'octets **unidirectionnels** qui **relie la sortie standard d'un processus à l'entrée standard d'un autre processus**. C'est l'**interpréteur de commandes** (**shell**) qui met en place ces tuyaux temporaires entre les processus. Linux supporte également les **tubes nommés** qui ont une existence physique.

Politique d'ordonnancement et priorité

Linux est un noyau **préemptif**. Un système d'exploitation multitâche est préemptif lorsque celui-ci peut **arrêter** (réquisition) **à tout moment n'importe quelle application** pour passer la main à la suivante. Le noyau garde toujours le contrôle (qui fait quoi, quand et comment), et se réserve le droit de fermer les applications qui monopolisent les ressources du système. Ainsi les blocages du système sont inexistants.

L'**ordonnanceur** est le composant du noyau qui décide **quel processus sera exécuté** par le processeur. Linux utilise par défaut l'ordonnanceur **CFS**, the "**Completely Fair Scheduler**" depuis la version 2.6.23. **A Chaque processus** on associe une **politique d'ordonnancement** et une **priorité statique**,sched_priority..

Il existe deux types de processus Linux :

1. Les **processus en ordonnancement en temps réel**, ont une priorité plus élevée que tous les autres processus entre 1 et 99. Si un processus en temps réel est prêt à être exécuté, il sera toujours exécuté en premier. Les processus en temps réel peuvent avoir deux types de politique, **round robin**(**SCHED_RR**) et first in first out(**SCHED_FIFO**). Dans l'ordonnancement **round robin**, chaque processus en temps réel exécutable est **exécuté à tour de rôle**. Dans l'ordonnancement **first in, first out**, chaque processus exécutable est **exécuté dans l'ordre** où il se trouve dans la file d'attente d'exécution et cet ordre n'est jamais modifié. Seul le **root** peut définir la **police d'un processus** dans cette **catégorie**.
   Techniquement, l'ordonnanceur maintient une liste de processus exécutables pour chaque valeur possible de sched_priority. Afin de déterminer quel processus s'exécute ensuite, l'ordonnanceur recherche la liste non vide avec la plus haute priorité statique et sélectionne le processus en tête de cette liste.
2. Les processus avec une police d'**ordonnancement normal** (**SCHED_OTHER**) pour lesquels l'ordonnanceur applique la **politique du temps partagé** ont une priorité de 0. Le processus à exécuter est choisi dans la liste statique de priorité 0 sur la base d'une **priorité dynamique** qui n'est déterminée qu'à l'intérieur de cette liste. La **priorité dynamique** est basée sur la valeur d'un attribut, **nice** (comprise entre -20 et 19 sur ubuntu 20.04 et qui vaut initialement 0), et est augmentée pour chaque quantum de temps où le processus est prêt à s'exécuter, mais refusé par l'ordonnanceur. Ceci assure une progression équitable entre tous les processus **SCHED_OTHER**...Seul le **root** peut **augmenter la priorité dynamique** d'un processus (**en diminuant nice**, -20 est la meilleurs priorité), un utilisateur simple ne peut que le réduire. Cette limitation peut être levée.

Nous vous renvoyons à la page de manuel de sched pour plus d'information. vous avez une bonne introduction sur le sujet *ici*[1]

6. Pour aller plus loin

Complément : le répertoire /proc

les informations de gestion des processus sont conservées dans le répertoire **/proc**. C'est un Pseudosystème de fichiers d'**informations sur les processus**. reportez vous a sa page de manuel pour plus d'informations.

1 - https://cours.polymtl.ca/inf2610/documentation/notes/chap8.pdf

Complément : systemd

systemd est un **gestionnaire** de **systèmes** et de **services** pour les systèmes d'exploitation Linux. ***systemd***[2] est un projet composé entre autre de :

1. un processus d'initialisation, **systemd**, qui s'occupe de gérer le démarrage, du lancement du noyau Linux à l'interface graphique, et de la surveillance des processus. Lorsqu'il est exécuté en tant que premier processus au démarrage (en tant que PID 1), il agit comme un système init qui met en place et maintient les services de l'espace utilisateur. (tiré des manpages).
2. un ensemble d'outils qui contrôlent le processus systemd, notamment **systemctl**, et qui permettent, entre autres, de suivre, redémarrer et arrêter les différents services d'une machine ;

Reportez vous aux pages de manuel de systemd pour plus d'informations...

2 - https://systemd.io/

II.Manipulation des processus et services par l'utilisateur

1. Affichage des processus

Les commandes de visualisation des processus

ps est la commande de référence pour l'affichage des processus, il présente un **cliché instantané des processus en cours.**

sans paramètres, il affiche les processus lancés depuis le terminal.

Sur certaines versions de ubuntu, la commande ps accepte **plusieurs options** :

1. les **options UNIX** qui peuvent être regroupées et qui doivent être précédées d'un tiret ;
2. les **options BSD** qui peuvent être regroupées et qui ne doivent pas être utilisées avec un tiret ;
3. les **options étendues GNU** qui doivent être précédées de deux tirets.

Reportez vous à sa page de manuel pour plus d'information. Quelques options de la syntaxe UNIX :

- **-e : tous les processus**(ou -A) ; **-f : infos supplémentaires** ; **-u login : processus d'un utilisateur**, **-T : processus du terminal**.

ps affiche son résultat sous forme de **colonnes**, selon les options :

- **UID** (Utilisateur propriétaire) , **PID** (Identifiant du processus), **PPID** (Identifiant du processus parent), **C** (Priorité du processus), **STIME** (Date et heure d'exécution), **TTY** (Terminal d'exécution), **TIME** (Durée de traitement), **CMD** (commande exécutée).

Vous avez des **options de filtrage** :

-u utilisateur, **-g** group, **-t** terminal, **-C** commande, **-o** format de sortie (on peut afficher seulement quelques champs), **--sort=%mem** (pour trier sur la consommation mémoire, cpu,...)

D'autres commandes permettent également d'afficher les processus :

1. **top** : Affiche un **cliché dynamique** temps réel des processus en cours ;
2. **pstree** : affiche un **arbre des processus ;**
3. **pgrep** : **Rechercher des processus** en fonction de leur nom et d'autres propriétés

Ces commandes sont très riches, reportez vous à leurs pages de manuel pour plus d'options.

Exemple : Exemples sur les commandes d'affichage de processus

$ **ps -e -f** # affiche tous les processus avec des infos supplémentaires.

$ **top** #affiche un cliché dynamique, pour quitter taper q.

$ **pstree** # affiche l'arborescence des processus

$ **pgrep firefox** # affiche les pid qui correspondent à une recherche sur le mot clé firfox

2. Envoi de signal à un processus

les commandes kill et pkill

les commandes **kill** et **pkill** envoient un signal à un processus . Les signaux principaux sont

- **2 - SIGINT** :**Interruption** du processus ;
- **9 - SIGKILL** : **Terminaison immédiate** du processus ;
- **15 - SIGTERM** :**Terminaison propre** du processus ;
- **18 - SIGCONT** : **Reprise** du processus ;

- **19 - SIGSTOP** : **Suspension** du processus.

Vous pouvez voir la liste de tous les signaux en faisant la commande **man 7 signal** ou **kill -l**. Lorsqu'un processus est bloquant, on lui envoie un signal **9-SIGKILL**.

Exemple : Exemples avec kill et pkill

1. Lancez le logiciel de traitement de texte Libre office writer
2. rechercher son identifiant de processus : [pid_office]
3. exécutez la commande
   $ **kill -19 [pid_office]** #suspend le processus
   $ **kill -18 [pid_office]** #reprend le processus
   $ **kill -SIGSTOP [pid_office]** #suspend le processus
   $ **kill -15 [pid_office]** #termine proprement le processus

Les raccourcis clavier

Les **raccourcis clavier** peuvent être utilisés pour envoyer un signal à un **processus synchrone** :

- **[CTRL] + [Z]** : suspension du processus
- **[CTRL] + [C]** : terminaison du processus
- **[CTRL] + [D]** : fermeture de l'entrée standard

La commande trap

La commande trap permet de "piéger" les signaux. Elle permet ainsi de faire certaines tâches telle que le nettoyage des fichiers temporaires, la destruction des processus enfants ...à la réaction au signal.

3. Tâches (job) et processus indépendants d'une connexion

modes de fonctionnement d'une commande

Les processus peuvent fonctionner de deux manières :

1. **synchrone** : l'utilisateur perd l'accès au shell durant l'exécution de la commande. L'invite de commande réapparaît à la fin de l'exécution du processus.
2. **asynchrone** : le traitement du processus se fait en **arrière-plan**, l'invite de commande est ré-affichée immédiatement. l'ajout de "**&**" à la fin d'une commande exécute la **commande en mode asynchrone**. Le processus est alors appelé **job** (tâche), le **numéro de job** est obtenu lors de la mise en tâche de fond et est **affiché entre crochets**, suivi du numéro de PID.

Manipulations des tâches

Les commandes suivantes permettent de manipuler des tâches (jobs) :

1. **jobs** : affiche la liste des processus tournant en tâche de fond et précise leur numéro de job ;
2. **fg** : place un processus au premier plan ;
3. **bg** :place un processus en arrière plan

Exemple : Exemples avec les taches

$ **xeyes -center red &** # lance une fenêtre xeyes a centre rouge en tache de fond

$ **jobs** # affcihera les taches de fond

$ **xeyes -center blue** # lance une fenêtre xeyes a centre bleu, on perd la main, faisons

[CTRL]+[Z] #suspend la tache avec pour numéro de job [xeyes_bleu]

$ **bg [xeyes_bleu]** # relance le processus en tache de fond.

processus indépendant d'une connexion

nohup permet d'exécuter une commande en la rendant insensible aux déconnexions, avec une sortie hors terminal.

4. Politique d'ordonnancement et Priorité

Priorité d'un processus

Les commandes suivantes permettent de jouer sur les priorités :

1. **nice** : Exécuter un programme avec une politesse modifiée
2. **renice** : Modifier la priorité des processus en cours d'exécution

Exemple : Exemples avec nice et renice

$ **nice -n+15 find / -name "fichier"** #lance une recherche avec une priorité dynamique de 15

$ **renice +15 -p 65577** # 65577 représente le pid du processus

5. Gestion des services

Affichage des services

Voici quelques commandes qui affichent les services :

- $ **systemctl list-unit-files --type=service**
  liste triée de tous les services accompagnés de leur état
- $ **systemctl list-units --type=service**
  liste des services actifs avec leur description
- $ **systemctl list-units --type=service --state=running**
  liste des services actifs, encours d'exécution, avec leur description :

manipulation des services

la commande systemctl Controle le processus systemd et gère les services. Sa syntaxe de base est :

$ systemctl ACTION <Nom_du_service>.service

ACTION sera la commande que l'on souhaite appliquer à la dite unité:

1. **start** : démarrer le service
2. **stop** : arrêter le service
3. **restart** : relancer le service
4. **reload** : recharger le service
5. **status** : connaître l'état du service
6. **enable** : activer un service
7. **disable** : désactiver un service

<Nom_du_service> est le nom du service visé.

Exemple : Exemples avec systemctl

$ **systemctl status ssh.service**

$ **systemctl restart automount.service**

Complément : target/runlevel

Au démarrage du système, dans le système **init**, il est possible de fixer un **niveau de démarrage** appelé **runlevel** et indiqué par un **chiffre** : **tous les services ne seront pas démarrés mais seulement une partie** de manière a ce que le système reste cohérente. Par exemple, on peut décider de démarrer un système sans interface graphique (c'est très fréquent avec les serveurs), il correspond au niveau 3.

Systemd introduit la notion de **target** au sein de ses **unités**. Une **target** permet de **regrouper plusieurs unités** dans un seul paquet et de retrouver la notion de **runlevel**.

Runlevel	Systemd Target	Remarques
0	runlevel0.target, poweroff.target	Arrête le système
1	runlevel1.target, rescue.target	Mode single user
3	runlevel3.target, multi-user.target	Mode multi-utilisateur, non graphique
2,4	runlevel2.target, runlevel4.target, multi-user.target	Mode défini par l'utilisateur, identique au 3 par défaut.
5	runlevel5.target, graphical.target	Mode graphique multi-utilisateur
6	runlevel6.target, reboot.target	Redémarre
emergency	emergency.target	Shell d'urgence

$ **sudo systemctl list-units --type target --all**

affiche toutes les unités de type target

Reportez vous à la page de manuel de **systemd.target**

Complément : Fichiers journaux

La commande **journalctl** permet de faire des requêtes sur le **journal de log de systemd.**

III.Le shell et les commandes

1. Fonctionnement du shell

Le shell est l'**interface** entre l'utilisateur et le noyau. Il **interprété** les commandes passées par l'utilisateur pour effectuer la tâche requise. Nous rappelons que sur un système GNU/Linux, **plusieurs shell** sont installés (cat /etc/shells). **bash** est le shell par défaut de Linux, il est compatible sh. Le shell effectue **certaines tâches** avant de soumettre les commandes au noyau :

1. la gestion des **métacaractères** ;
2. la **substitution des noms de fichiers et répertoires** ;
3. la **gestion des variables** ;
4. la **redirection des entrées et des sorties** ;
5. la **gestion des tubes(pipe)** ;

c'est lui également qui exécute les scripts.

2. Métacaractères,substitution de commande et substitution de noms de fichiers

Définition et liste des métacaractères

Un métacaractère du shell désigne un caractère que le shell transforme avant de le traiter. La liste des métacaractères et leurs sémantique est donnée dans le tableau suivant :

métacaractère	signification
$	Contenu d'une variable
;	Séparateur de commande (permet d'en placer plusieurs sur une ligne)
( )	Groupement de commandes exécutées dans un sous-shell
`` (c'est l'anti-cote, on peut aussi utiliser $())	substitution de commande
<	Redirection en entrée à partir d'un fichier
<<	Redirection en entrée à partir des lignes suivantes
>	Redirection en sortie vers un fichier
>>	Redirection en sortie vers un fichier en mode « ajout »
|	Redirection vers une commande (tube ou pipe)
~	Chemin absolu du répertoire de travail de l'utilisateur
&	Exécution en arrière plan d'une commande

métacaractère	signification
*? []['^]	joker de substitution des noms de fichier.
\	Neutralisation de tout métacaractère placé après
" "	chaîne de caractères. neutralisation des métacaractères placés à l'intérieur à l'exception de $,‘ (anticote) et !.
' '	chaîne de caractères. neutralisation de tous les métacaractères placé à l'intérieur.

substitution de noms de fichiers

Les métacaractères utilisés pour la substitution de noms de fichiers sont :

métacaractère	signification
*	chaîne de caractères quelconque
?	un caractère quelconque
[abc]	un caractère parmi ceux spécifiés
['^]	un caractère différent de ceux spécifiés

Exemple : Exemples avec la substitution de noms de fichiers

$ **ls /dev/sd*** #liste tous les pèriphèriques SATA et leurs partitions

$ **ls /dev/tty*** #liste tous les tty (terminaux)

$ **ls ab*** # liste tous les fichiers dont le nom commencent par ab

$ **ls [ab]cd*** # liste tous les fichiers dont le nom commence par a ou b suivi de cd et se terminant par n'importe quelle suite de caractères

$ **ls ['f]cd??** #tous les fichiers dont le nom commence par autre chose que f

suivi de cd et de deux caractères quelconques.

Substitution de commandes

La substitution de commande permet d'exploiter le résultat d'une commande dans le paramètre d'une autre commande. Elle se fait en passant la commande entre **anti-cotes (`)** ou entre les parenthèses de **$()**. Par exemple pour afficher le message "Nous somme le date_actuelle" ou date_actuelle est la date et l'heure au moment de l'affichage du message :

$ **echo Nous somme le `date`**

ou

$ **echo Nous somme le $(date)**

3. Variables

Variables spéciales

Reportez vous au chapitre sur les scripts shell pour les généralités sur les variables : création, accès,..

Le Shell possède ds variables spéciales :

- **$$** : Identifiant de processus(PID) du **shell courant** ;
- **$!** : Identifiant(PID) de la **dernière tâche (job)** lancée en arrière plan ;
- **$?** : **Code retour de la dernière commande** : Il vaut **0** en cas de **réussite** et un **nombre strictement positif** en cas d'**échec**.

Variables d'environnement

L'**environnement** désigne le **contexte** d'exécution d'un programme, c'est un ensemble de **paramètres** (variables)

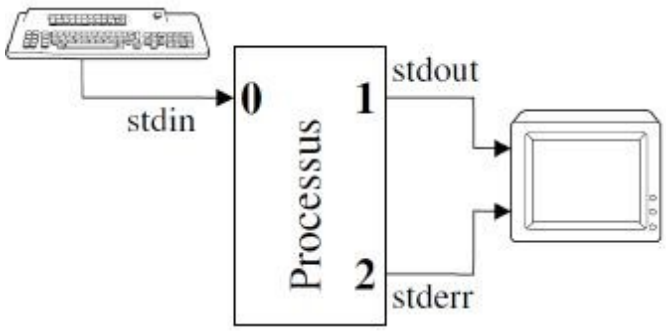
contenant des valeurs de type chaîne de caractères.

La commande **env** sans paramètres affiche les variables d'environnement. La commande env permet d'exécuter un programme dans un environnement modifié.

Vous avez dans le tableau suivant quelques variables d'environnement et leur signification.

variable	sémantique
USER,LOGNAME	Nom de login de l'utilisateur.
HOME	Chemin du répertoire personnel de l'utilisateur.
TERM	Type de terminal utilisé.
SHELL	Chemin sur le fichier de programme du shell actuellement utilisé.
PATH	Liste des répertoires dans lesquels les programmes à exécuter seront recherchés.
LANG	Nom de la locale (langue) à utiliser par défaut pour les paramètres d'internationalisation des applications.

4. Redirection



Descripteurs standards

Rappelons que toute commande exécutée (processus) ouvre trois descripteurs :

1. 0 qui désigne l'entrée standard, le clavier par défaut ;

2. 1 qui désigne la sortie standard, le terminal ;
3. 2 qui désigne la sortie d'erreur, le terminal.

La commande attend ses entrées du clavier et affiche ses résultats et ses messages d'erreur sur le terminal.

Redirection de la sortie standard

La redirection de la sortie standard consiste à modifier la sortie standard en demandant à la commande d'envoyer son résultat vers un fichier :

1. la **redirection simple** se fait avec **>** : le fichier de redirection est créé s'il n'existe pas, il est écrasé puis recréé s'il existait.
2. la **redirection double** se fait avec **>>** : le fichier est créé s'il n'existe pas, s'il existait le résultat de la commande est ajouté en fin de fichier.

Par exemple la commande

$ **who > quiestconnecte.txt**

met dans le fichier quiestconnecte.txt le résultat de la commande who.

La suite de commandes

$ **date > ousuisje.txt**

$ **pwd >> ousuisje.txt**

met dans le ficher ousuisje.txt la date et le print working directory.

Redirection de la sortie d'erreur

La redirection de la sortie d'erreur consiste à modifier la sortie d'erreur en demandant à la commande d'envoyer les messages d'erreur vers un fichier :

1. la **redirection d'erreur simple** se fait avec **2>** : le fichier de redirection est créé s'il n'existe pas, il est écrasé puis recréé s'il existait.
2. la **redirection d'erreur double** se fait avec **2>>** : le fichier est créé s'il n'existe pas, s'il existait le résultat de la commande est ajouté en fin de fichier.

Redirection de l'entrée standard

La redirection de l'entrée standard consiste à modifier l'entrée standard en demandant à la commande de prendre son entrée d'un fichier. Cela se fait avec **<**. Elle est moins utilisée, beaucoup de commande peuvent prendre en entrée un flux ou un nom de fichier.

la commande :

$ **sort < monfichier.txt**

Trie le contenu du fichier monfichier.txt et l'affiche à l'écran

5. Tubes

Le tube permet de rediriger la sortie d'une commande vers l'entrée d'une autre commande. Cela se fait avec **|.**

Par exemple la commande :

$ **du | sort -rn**

affichera la taille des fichiers et répertoires triés du plus grand au plus petit

Conclusion

La gestion des processus est un mécanisme très complexe. Vous pouvez vous reporter à la page de manuel de de systemd...