Séance n° 5 : *Alternatives*, gestion des utilisateurs et des processus

1 *Alternatives* sous Debian

Il arrive fréquemment que plusieurs logiciels puissent remplir la même fonction de manière interchangeable (éditeur de texte, visionneur de texte ou *pager*, machine virtuelle Java, ...), ou qu'il existe plusieurs implémentations concurrentes du même logiciel (*gawk* vs *mawk*, *traceroute*, *netcat*, ...).

Sous Debian, les *alternatives* permettent de sélectionner un logiciel pour remplir un rôle de manière globale, pour tous les utilisateurs du système. Ainsi, si vous configurez *vim* comme éditeur par défaut, la plupart des logiciels qui ont besoin d'appeler un éditeur de texte vont désormais utiliser *vim*.

Q1. Vers où pointe le lien `/usr/bin/editor` ?

Q2. Positionnez vous dans `/etc/alternatives`, et listez le contenu de ce répertoire. Ce répertoire contient des liens symboliques définissant l'ensemble des choix d'alternatives pour le système.

Q3. Quel est actuellement votre éditeur par défaut ?

Q4. Installez les éditeurs `jed`, `nano`, `vim` et `emacs`.

Q5. Avec la commande `update-alternatives`, listez les différentes alternatives possibles pour *editor*.

Q6. Avec la commande `update-alternatives`, modifiez l'éditeur par défaut du système. Vérifiez que le choix est bien pris en compte, par exemple en lançant `visudo`, et en confirmant que l'éditeur choisi est bien utilisé.

2 Gestion des utilisateurs et des groupes

Sous Unix, un *UID (User ID)* numérique est attribué à chaque utilisateur.

Chaque utilisateur possède également un groupe primaire, et peut faire partie de groupes supplémentaires.

Chaque groupe est identifié par un GID (Group ID).

Q1. À l'aide de la commande `id`, déterminez votre UID, votre GID principal, et vos groupes supplémentaires.

Q2. Par défaut, les fichiers sont créés avec vos UID et GID courants.

(a) À quoi servent les commandes `newgrp` et `sg` ?

(b) Expérimentez les commandes `newgrp` et `sg` en créant des fichiers (commande `touch`).

Dans une infrastructure réseau complexe, la gestion des comptes utilisateurs est souvent centralisée (en utilisant *NIS* ou *LDAP*). Mais dans notre cas, les informations sur les comptes sont simplement stockées dans 3 fichiers :

— `/etc/passwd`, qui stocke la liste des utilisateurs du système avec leur caractéristiques : identifiant, mot de passe ou lettre « `x` », `uid` (identifiant utilisateur), `gid` (identifiant groupe principal), nom complet de l'utilisateur, répertoire de connexion et programme à exécuter après la connexion (généralement un *shell*, mais cela peut être `/bin/false` par exemple si on souhaite interdire la connexion d'un utilisateur).

— `/etc/shadow`, qui stocke des informations supplémentaires pour chaque utilisateur : identifiant, mot de passe crypté (si la lettre « `x` » a été précisée dans la seconde colonne du fichier `/etc/passwd` pour un utilisateur), ainsi que des champs permettant de gérer l'expiration du mot de passe.

— `/etc/group`, qui stocke la liste des groupes d'utilisateurs du système avec leurs caractéristiques : nom du groupe, mot de passe du groupe (habituellement vide), `gid` (identifiant du groupe), liste des membres du groupe séparés par des virgules.

Les comptes se créent à l'aide de la commande `adduser` qui s'utilise aussi bien en mode interactif (si aucun paramètre n'est passé en ligne de commande) qu'en mode *batch* si tous les paramètres nécessaires à la création d'un compte sont fournis.

Le fichier `/etc/adduser.conf` contient de nombreuses options relatives à la création des comptes. Il indique en particulier le squelette de répertoire qui sera copié dans le *home* de tout nouvel utilisateur créé (généralement, c'est le contenu du répertoire `/etc/skel`).

Les commandes `deluser`, `addgroup` et `delgroup` permettent d'effectuer les autres opérations de création et de suppression nécessaires pour la gestion des utilisateurs et groupes d'un système Linux.

Toutes ces commandes sont en faites des *wrappers* autour de commandes de plus bas niveau : `useradd`, `userdel`, `groupadd`, `groupdel`.

Les commandes suivantes peuvent être exécutées par des utilisateurs réguliers ou par l'administrateur du système. Certaines de leurs options ne sont accessibles qu'avec un des profils précités.

— `chfn` : permet de modifier les informations administratives associées à un utilisateur

— `chsh` : permet de changer le *shell* associé à un utilisateur

— `chage` : permet de changer les conditions d'expiration du mot de passe d'un utilisateur

— `passwd` : permet de changer le mot de passe d'un utilisateur, doté de nombreuses options
  — `-e` : oblige un utilisateur à changer son mot de passe lors de la prochaine connexion),
  — `-l` : désactive temporairement un compte
  — `-u` : réactive un compte précédemment désactivé

Enfin, d'autres commandes ne sont accessibles qu'à l'administrateur :

— `vipw` : permet d'éditer de manière sûre le fichier `/etc/passwd`, et de mettre ensuite à jour le fichier `/etc/shadow`.

— `vigr` : permet d'éditer de manière sûre le fichier `/etc/group`.

Q1. Consultez la documentation des différentes commandes mentionnées ci-dessus, et essayez-les.

Q2. Placez plusieurs fichiers dans `/etc/skel`, puis créez un utilisateur. Que constatez-vous ? À quoi sert le répertoire `/etc/skel` ? Sachant que *skel* est l'abréviation de *skeleton* (*squelette*), pourquoi ce nom ?

Questions subsidiaires : (à faire en fin de TP)

Q3. Écrivez un script permettant d'automatiser la création de 9 comptes nommés `etu1` à `etu9`, puis de les ajouter (chacun) à des groupes `gr1`, `gr2`, `gr3`. Créez un compte sans mot de passe, puis changez le mot de passe avec `chpasswd`. Le mot de passe sera généré automatiquement en utilisant `pwgen`. Si vous utilisez `adduser` pour créer les comptes, il faut renseigner toutes les options pour éviter qu'il pose des questions.

Q4. Installez le *cracker* de mots de passe `john`, et vérifiez que vous pouvez casser un mot de passe simple (4 à 5 caractères).

3 Processus

Un *processus* est un programme en cours d'exécution : il peut y avoir simultanément plusieurs processus du même programme en exécution en mémoire, par exemple si plusieurs utilisateurs du même système éditent des fichiers avec le même éditeur de textes.

Le premier processus d'un système, ancêtre de tous les autres, correspond au programme `init`. Il est lancé directement par le noyau après son chargement. Les autres processus sont créés par un processus parent et appartiennent à un utilisateur. Ainsi en règle générale, à chaque nouvelle commande, le système lance un nouveau processus. Chaque processus est identifié par un numéro unique, son PID, qu'il peut être important de connaître.

3.1 Liste des processus d'un système

La commande `ps` permet de dresser un état des processus d'exécutant sur un système à un moment donné. Sans argument, `ps` ne renvoie que les processus rattachés au terminal depuis lequel la commande a été saisie. Avec d'autres paramètres, `ps` renvoie d'autres informations, identifiées par les en-têtes de colonne suivants :

- `PID` : numéro d'identification (unique) d'un processus
- `USER` : utilisateur propriétaire d'un processus
- `TTY` : terminal auquel est rattaché un processus
- `%CPU` : pourcentage de processeur utilisé par un processus
- `%MEM` : pourcentage de mémoire vive utilisé par un processus
- `PPID` : numéro d'identification du père d'un processus
- `VSZ` : taille de la mémoire virtuelle occupée par un processus
- `RSS` : taille mémoire effective occupée par un processus
- `STAT` : état d'un processus, avec les significations principales suivantes
  - `R` : en cours d'exécution
  - `S` : en attente
  - `T` : processus stoppé
  - `Z` : processus terminé, mais non « récupéré » par son père

La page de manuel de cette commande comprend de nombreux exemples. Les exécutions du type `ps aux` ou `ps auxf` (permettant de visualiser la filiation des processus) sont fréquemment utilisées.

Enfin, la commande `top` affiche de façon interactive et synthétique les différents processus s'exécutant, en les classant suivant les ressources qu'ils occupent. Cette commande, très

utile pour surveiller l'état des ressources d'une machine, affiche également des informations de synthèse relative au système (nombre de processus présents, swap...)

3.2 Gestion des processus d'un terminal

Le lancement « normal » d'un programme se fait en saisissant son nom, ses paramètres et en validant par la touche « Entrée » ; le programme est lancé dans ce cas en *avant-plan*. Pour lancer un programme en *arrière-plan*, il suffit d'ajouter « `&` » à la fin de la ligne : le *shell* rend ainsi tout de suite la main à l'utilisateur. Dans ce cas, le shell affiche deux numéros : le numéro de *job* et le PID du processus créé.

Il ne faut pas confondre les numéros de job avec les numéros de processus. Tout d'abord, un numéro de job n'est unique que dans un shell donné et n'a aucune signification au niveau du système complet, alors que le numéro de processus est attribué par le système à chaque programme en cours d'exécution. Ensuite, une même commande du shell peut lancer plusieurs processus conjointement. Dans ce cas, il y a bien évidemment plusieurs numéros de processus, mais un seul et unique job. Ce genre de situation se produit par exemple lors de l'utilisation d'une redirection du flux de sortie standard d'un processus vers le flux d'entrée standard d'un autre processus.

Quelques raccourcis utilisables sur les processus en avant-plan :

— `Ctrl-C` : tue le processus en avant-plan

— `Ctrl-Z` : suspend le processus en avant-plan

La commande `jobs` dans un terminal permet d'avoir la liste des processus associés à ce terminal, ainsi que leur état. Exemples de commandes pour gérer les processus du terminal courant :

— `bg %1` : lance un arrière plan le premier job

— `kill -9 %3` : tue le troisième job

— `fg %2` : replace le deuxième job à l'avant-plan

3.3 Les signaux

Certains processus sont en « avant-plan », mode habituel de fonctionnement, soit en « tâche de fond » et il existe aussi des processus dits « détachés » qui n'ont été lancés dans aucune console. Ces processus détachés, qui sont généralement les exécutions de services du système, sont appelés *daemons*, francisés en *démons*. L'administrateur n'ayant pas directement la main sur un démon (où taper Crtl-C pour stopper le programme ?), il peut lui envoyer un signal pour le supprimer ou agir sur sa configuration. Suivant le type de signal envoyé, et le processus concerné, l'interprétation sera différente. La plupart des signaux sont envoyés par le système lui-même, mais il est également possible d'envoyer des signaux à partir d'un *shell*. La commande permettant d'envoyer des signaux aux processus est la commande `kill`. Les signaux peuvent être désignés par leur nom ou par leur numéro (liste non exhaustive) :

Nom	N°	Description
SIGHUP	1	signal émis à tous les processus d'un terminal quand ce terminal se déconnecte
SIGINT	2	signal d'interruption simple depuis le clavier (Ctrl-C)
SIGILL	4	signal émis en cas d'instruction illégale dans le programme
SIGKILL	**9**	**signal permettant de tuer un processus sans condition**
SIGBUS	10	signal envoyé en cas d'erreur sur le bus d'adressage
SIGSEGV	11	signal émis en cas de violation d'accès à la mémoire
SIGALRM	14	signal associé à une horloge
SIGTERM	15	signal demandant à un processus de se terminer

La commande `killall` effectue également des envois de signaux, mais prend en paramètre le nom d'un processus plutôt que son PID. Selon les cas, cela peut être plus pratique, mais parfois aussi plus dangereux... Enfin, la commande `pidof` permet de retrouver le PID d'un processus à partir de son nom. Quelques exemples d'utilisation :

- `kill -15 PID` : demande normale d'arrêt au processus, il peut refuser (-15 peut être remplacé par SIGTERM)
- `killall -9 httpd` : suppression plus radicale, en cas de processus récalcitrant !
- `kill $(pidof ypserv)` : supprime le processus serveur NIS, dont le pid est obtenu le résultat de la commande `pidof`
- `killall -HUP httpd` : ordonne au processus de relire son fichier de configuration, ce qui évite de le relancer
- `kill -l` : pour connaître la liste des signaux qu'on peut passer à `kill`

3.4 Questions

Q1. Comment vérifier que le processus `init` est bien le tout premier lancé par le noyau ?

Q2. Comparez les PID fournis par la commande `ps aux` et les numéros conservés dans `/var/run`. Comparez les ensuite aux numéros qui sont dans `/proc`.

Q3. Expliquez ce que signifie la commande suivante `ps aux | grep bash` ?

Q4. Que retourne `pidof bash` ? Pourquoi ?

Q5. Expliquez ce que réalise la commande suivante (à lancer en tant que root) :
`tail -f /var/log/messages > /dev/tty1 &`
Notez le PID, observez le contenu de la console `tty1` (correspondant à la première console, accessible avec CTRL+ALT+F1), ensuite supprimez cette tache de fond par kill -9 PID et vérifiez.

Q6. Se connecter (sous un compte quelconque) à la console tty3. Saisissez la commande permettant de connaître tous les processus qui concernent la console tty3.

Q7. Combien de processus s'exécutent actuellement sur votre système ? Dont combien appartenant à l'utilisateur `root` ?

Q8. Placez-vous sous le terminal `tty1` et tuez le *shell* associé à ce terminal (testez plusieurs signaux pour voir s'il existe une différence). Que constatez-vous ?

Q9. Avec `top`, n'affichez que vos processus, en les triant par utilisation mémoire, et en raffraichissant l'affichage toutes les 0.5 secondes. Attention, certaines de ses fonctionalités

ne sont accessibles qu'une fois `top` en cours d'exécution : ce ne sont pas forcément des paramètres à passer au lancement de `top`.

Question subsidiaire : (à faire en fin de TP)

Q10. Une alternative à `top` assez couramment utilisée est `htop`. Installez `htop` et étudiez ses fonctionnalités.

4 Automatisation du lancement de tâches

Un administrateur Unix ne peut pas toujours être disponible, alors que les systèmes qu'il administre ont souvent besoin d'effectuer un certain nombre de tâches à des moment précis. Les systèmes Unix proposent un certain nombre de mécanismes standards pour effectuer ces tâches, suivant le type de besoin. Ces mécanismes sont également disponibles pour les utilisateurs non administrateurs, parfois avec quelques restrictions.

4.1 Programmation du lancement ponctuel de commandes

Il arrive parfois qu'on souhaite programmer le lancement ponctuel d'une commande donnée à un moment donné. Les systèmes UNIX proposent une commande, la commande `at`, permettant de programmer cette exécution. Il suffit alors de fournir à la commande `at` tous les paramètres nécessaires : horaire d'exécution (relatif ou absolu), ainsi que la commande à exécuter. Sur l'exemple suivant, on demande l'exécution de la commande `echo` dans 2 minutes (donc en utilisant un horaire relatif). Les lignes commençant par `at>` correspondent aux zones où il faut saisir les commandes à exécuter, sachant que `^D` (CTRL-D) permet de signifier à la commande `at` que la saisie des commandes est terminée (ce qui provoque l'affichage de `<EOT>`).

Exemple d'exécution de la commande `at`

```
1  phi phil 54 : at now + 2 minutes
2  warning: commands will be executed using /bin/sh
3  at> echo yop
4  at> <EOT>
5  job 1 at 2007-10-22 10:24
```

La commande demandée est alors réalisée 2 minutes plus tard, et les résultats de la sortie standard sont envoyés par mail à l'utilisateur qui en a fait la demande. Les commandes `atq` et `atrm` permettent respectivement d'examiner la liste des commandes en attente d'exécution et de supprimer une commande en attente d'exécution.

4.2 Programmation du lancement régulier de commandes

Souvent, les commandes que souhaitent exécuter un administrateur UNIX ne sont pas ponctuelles, mais régulières. Il s'agit par exemple de surveiller une ressource particulière du système ou d'effectuer des tâches de maintenance : place restant disponible, fichiers temporaires trop anciens, sauvegardes... Dans ces situations, la commande `at`, destinée à effectuer des tâches ponctuelles, n'est alors pas satisfaisante. Les systèmes UNIX proposent alors un autre mécanisme, basé sur le démon `cron`. Cela permet alors de programmer des tâches à intervalle

régulier, comme tous les ans, tous les mois, toutes les semaines, toutes les heures ou toutes les heures. Il existe plusieurs manières de programmer l'exécution de tâches régulières par l'intermédiaire de `cron` :

— placer un script dans un des répertoires `cron.monthly`, `cron.weekly`, `cron.daily` ou `cron.hourly`, qui sera automatiquement exécuté à chaque échéance mentionnée dans le nom du répertoire,
— utiliser la commande `crontab` permettant d'éditer le fichier `/etc/crontab` où sont placées des commandes à exécuter et leur planification d'exécution ; les possibilités de réglage des horaires d'exécution sont alors beaucoup plus riches.

Pour éditer le fichier `/etc/crontab`, il faut lancer la commande `crontab -e` qui va alors appeler un éditeur pour cela (celui par défaut — `vi` *a priori* — ou celui spécifié par la variable d'environnement `VISUAL` ou `EDITOR`). Le fichier `/etc/crontab` contient alors une ligne par commande programmée, sur 7 colonnes permettant d'exprimer :

— la condition portant sur les minutes (nombre de 0 à 59 ou `*` pour représenter toutes les valeurs possibles),
— la condition portant sur les heures (nombre de 0 à 23 ou `*` pour représenter toutes les valeurs possibles),
— la condition portant sur le jour du mois (nombre de 1 à 31 ou `*` pour représenter toutes les valeurs possibles),
— la condition portant sur le mois (nombre de 1 à 12 ou `*` pour représenter toutes les valeurs possibles),
— la condition portant sur le jour de la semaine (nombre de 0 à 7, le 7 correspondant au dimanche, ou `*` pour représenter toutes les valeurs possibles),
— l'utilisateur exécutant la commande (`root` pour /etc/crontab, les tâches planifiées des autres utilisateurs étant stockées dans `/var/spool/cron/crontabs` généralement),
— la commande à exécuter avec ses paramètres d'exécution.

Pour les valeurs numériques, il est possible d'exprimer des énumérations de valeurs séparées par des virgules, ou des intervalles spécifiés par des tirets sachant que `a-b` spécifie toutes les valeurs de `a` à `b` et que `a-b/c` spécifie toutes les valeurs entre `a` et `b` avec un incrément de `c` (par exemple `1-7/2` représente les valeurs `1`, `3`, `5` et `7` et `*/3` pour les heures représente une commande lancée toutes les 3 heures).

La commande `crontab -l` peut être utilisée pour afficher le contenu du fichier de configuration de l'utilisateur qui l'appelle. En tant qu'administrateur du système, il est possible de modifier les paramètres pour n'importe quel utilisateur. Pour cela, il faut préciser le login de l'utilisateur avec l'option `-u`.

Q1. Installez et configurez le paquet `exim4-daemon-light`, et vérifiez que vous pouvez recevoir des mails localement (vous pouvez les consulter avec `mutt`, par exemple).

Q2. À quoi sert la commande `at` ?

(a) Avec `at`, programmez une tâche simple dans 2 minutes.

(b) Vérifiez qu'elle a bien été programmée avec `atq`.

(c) Après son exécution, consultez son résultat envoyé par mail.

Q3. À quoi sert `cron` ?

Q4. Avec votre compte utilisateur (non-administrateur), programmez l'exécution périodique d'une commande simple. Vérifiez que vous recevez son résultat par mail.

Question subsidiaire : (à faire en fin de TP)

Q5. Imaginons que votre serveur souffre de "pics de charge" difficiles à comprendre.

(a) Écrivez un script qui va consulter `/proc/loadavg`. Si le premier nombre est supérieur à 1, exécutez quelques commandes vous permettant d'analyser la situation (`ps aufx`, par exemple).

(b) Programmez l'exécution périodique de votre script.

(c) Testez. Pour faire monter la charge de votre machine, vous pouvez utiliser la commande `stress`, ou simplement exécuter `sh -c "while true; do echo -n ' '; done"` (= boucle infinie en *shell*)

5 Démarrage et arrêt du système

Au démarrage ou à l'arrêt d'un système UNIX, de nombreuses tâches doivent être réalisées. Suivant l'usage d'une machine, ces tâches peuvent correspondre aux fonctionnalités de base du système (détection des périphériques présents, montage des partitions, ...), ainsi que d'autres plus spécifiques (chargement d'un module particulier dans le noyau, lancement/arrêt de serveurs...)

Au démarrage d'un système UNIX, le premier processus à être lancé correspond au programme `init`. Ce processus a pour charge d'exécuter toutes les tâches nécessaires au démarrage du système, d'effectuer un certain nombre de tâches de maintenance durant la période où le système est en cours d'exécution, et finalement d'effectuer d'autres tâches au moment où le système est arrêté.

Au démarrage comme à l'arrêt d'un système, `init` effectue un certain nombre de tâches relatives aux fonctionnalités de base du système, qui ne sont généralement pas ou peu personnalisées.

Il est également chargé de démarrer tous les services (serveur web, mail, etc.) tournant sur la machine.

Historiquement, le système d'init utilisé sous Linux était `sysvinit`. La plupart des distributions ont maintenant migré vers `systemd`, auquel une séance entière sera consacrée plus tard dans l'année.

Quel que soit le système d'init utilisé, il est possible de démarrer, d'arrêter ou de connaître l'état d'un service avec la commande `service`.

Q1. Avec `service ssh status`, observez le statut du service `ssh`

Q2. Essayez les paramètres `stop`, `start`, `restart` du service `ssh`.

Pour arrêter le système, plusieurs commandes peuvent être utilisées :

— `shutdown` : permet d'arrêter, de redémarrer le système. Cette commande offre en outre la possibilité de programmer une de ces opérations et d'en avertir les utilisateurs.

— `halt` : synchronise les disques avec les *buffers* et arrête le système.

— `poweroff` : comme `halt`, mais coupera également l'alimentation électrique de la machine.

— `reboot` : comme la commande `halt`, mais en redémarrant le système à a fin.

Questions subsidiaires : (à faire en fin de TP)

Q1. Il est possible de choisir un autre programme pour réaliser la fonction d'*init* lors du démarrage du système, en précisant `init=`*`programme`* dans la ligne de commande du noyau. Une astuce courante est d'utiliser cette possibilité pour se connecter sur un système sur lequel vous avez perdu le mot de passe root, en lançant un shell au lieu de lancer init. Rebootez votre machine, et dans Grub, éditez la ligne de commande du noyau pour rajouter `init=/bin/bash`, puis démarrez votre système.

Q2. La configuration d'*exim4* que vous avez réalisée ne vous permet pas d'échanger des mails avec vos voisins. Configurez *exim4* pour pouvoir échanger des mails avec d'autres étudiants.