M3206
Automatisation des tâches d'administration
Rappels sur UNIX / Linux

E. Viennet

Département R&T
IUT de Villetaneuse

9/2020

Plan du cours

Qu'est-ce qu'un système d'exploitation ?

Le système d'exploitation (*Operating System, ou OS*) d'un ordinateur est un programme qui assure la **gestion du matériel** et défini des mécanismes et API pour le fonctionnement des **applications** :

- accès aux ressources (processeur, mémoire, fichiers, périphériques) ;
- gestion des utilisateurs ;
- interfaces homme-machine (parfois).

API : *Application Programmer Interface*

Terminologie :

- système multi-tâches
- système multi-utilisateurs

Systèmes répandus :

Sur les ordinateurs (bureaux, serveurs) :

- Microsoft Windows (XP, 7, 8.1, 10, ...)
- Macintosh OS X (Unix)
- GNU/Linux (Unix)

Mais aussi sur les appareils embarqués :

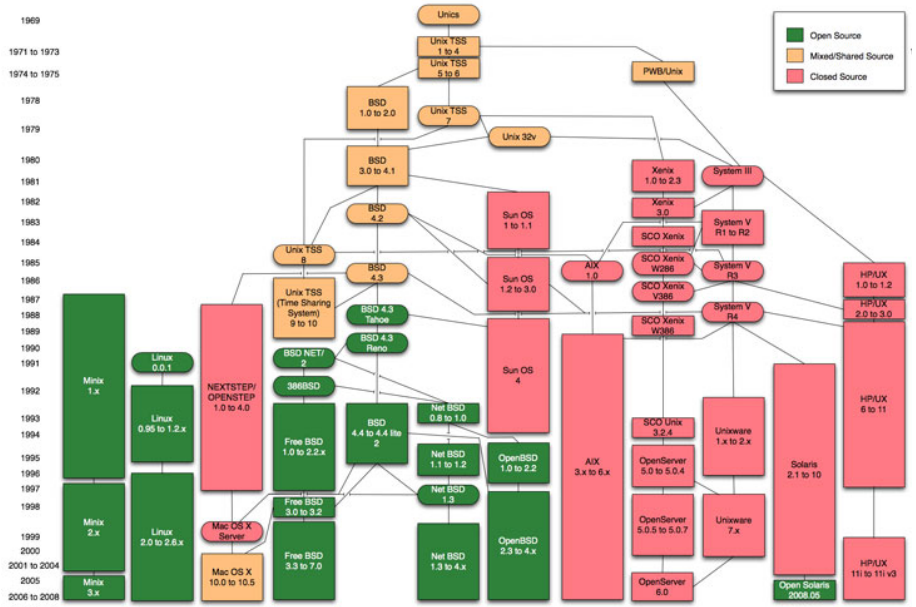
- Cisco IOS (routeurs)
- Smartphones et tablettes :
  Apple iOS, Google Android, basés sur Unix

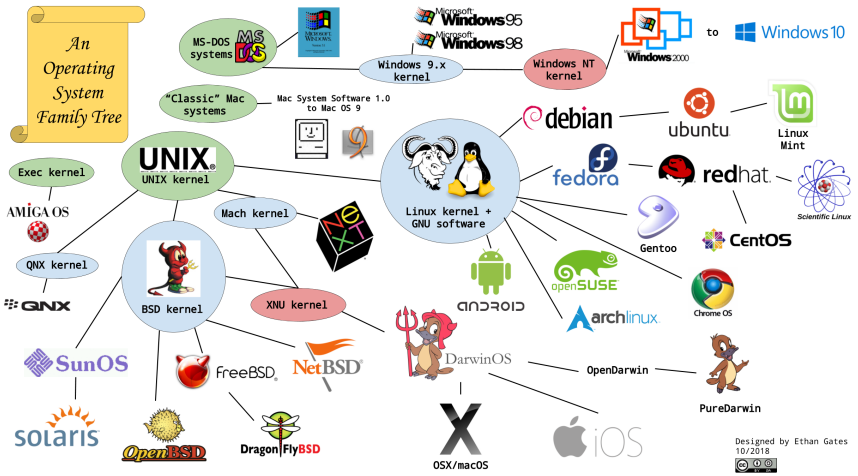
Historique d'UNIX

Un système ancien et toujours moderne !

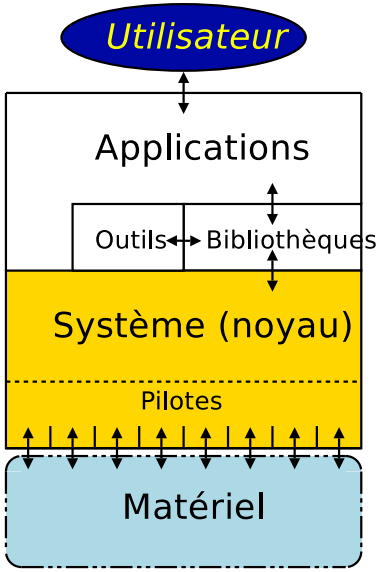
- 1969 : première version d'UNIX (Bell Labs) inspirée par MULTICS et écrite en langage C.
- fin 70 : variantes "Système V" (ATT) et BSD (U. Berkeley) ;
- fin 70 - fin 80 : unix sur stations de travail (Sun, HP, IBM...) ; versions propriétaires, développement d'interfaces graphiques (X11) ;
- années 1990 - 2020 : développement de Linux ;
- actuellement : **GNU/Linux**, Mac OS X, (Free)BSD, iOS, Android.

Arbre généalogique d'UNIX (1969-2008)





Architecture générale d'UNIX



Architecture d'UNIX

Le noyau : "cœur" du système d'exploitation

- Code exécuté en mode "superviseur" : le processeur a accès à toutes les ressources (gestion mémoire, interruptions, ...).
- Responsable :
  - du partage des **ressources**
  - de la gestion des **utilisateurs** et **droits d'accès**
  - de la gestion des **processus**
- accède aux périphériques via des *pilotes*
- défini divers mécanismes utilisés par les applications via des **appels systèmes** : communication, synchronisation, accès aux fichiers et périphériques...

Architecture d'UNIX

Les pilotes (*drivers*)

Interface entre le noyau et les périphériques

- Dans Linux, les pilotes correspondent à des **modules**

Architecture d'UNIX

Les processus

Les *processus*, programmes en cours d'exécution

- se partagent la mémoire et le processeur
- sont associés à un utilisateur
- forment un **arbre** : un processus peut lancer des **processus fils**
- sont identifiés par un **numéro**, le PID *processus identifier*

Architecture d'UNIX

Les outils

Logiciels distribués avec UNIX, accomplissant les tâches de base :

- accès au système : interpréteurs de commandes (**shells**)
- documentation (`man`)
- accès aux fichiers (`ls, mv, cp, rm, mkdir, vi, find...`)
- gestion des processus (`ps, kill, top, ...`)
- administration (par exemple `adduser`)
- développement de logiciels (`cc, lex, ...`)

Architecture d'UNIX

Les utilisateurs

Identifiés par un "login" (pseudonyme), accès protégé par un mot de passe.

- identifié en interne par un numéro (UID, *user identifier*)
- utilisateurs définis localement (`/etc/passwd`) ou via un annuaire partagé en réseau (NIS, LDAP, ...).
- chaque processus est associé a un utilisateur (droits d'accès).

Attributs importants

- répertoire de connexion (HOME)
- shell (interpréteur de commandes lancé à la connexion)

Groupes

- utilisés pour définir des catégories d'utilisateurs
- définis dans `/etc/group`
- chaque fichier appartient à un utilisateur et un groupe

Partie 2

Rappels sur les commandes de base UNIX

Commandes de base

Shell

Il existe plusieurs shells : sh, csh, ksh, tcsh (variante de csh) et bash (variante de sh).
Le standard sous Linux est **bash**.

Fonctionnement en mode interactif

1. afficher l'invite de commande (**prompt**)
2. attendre que l'utilisateur entre une ligne
3. "décoder" la commande et lancer éventuellement les processus correspondants
4. attendre la fin de l'exécution de la commande, sauf si elle est lancée en "tâche de fond" (termine par &)
5. recommencer à l'étape 1.

Script

Le shell interprète les commandes regroupées dans un fichier (script).

Commandes de base

Chemins

Les fichiers sont désignés par des **chemins**

- chemins absolus (commence par `/`) : `/home/dupont/courrier/lettre01.txt`
- chemins relatifs au **répertoire courant** : `toto`, `../travail/commande.txt`, `./configure`

Dans les commandes shell :

- `~` désigne le répertoire de connexion
- `~toto` désigne le répertoire de connexion de l'utilisateur toto

Commandes de base
Commandes supposées connues

`cd, ls, pwd, mkdir, rmdir, rm, cp, mv, man, date, vi, emacs`

Options à connaitre :

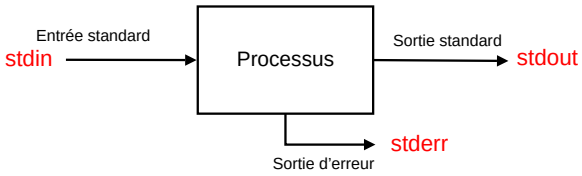
- `ls [-a] [-l] [-t] [-r]`
- `mkdir [-p]`
- `rm [-i] [-r]`
- `cp [-i] [-r] [-p]`
- `mv [-i]`

Commandes de base
autres commandes utiles, à connaitre

- `grep motif [fichier]`
- `wc [-c] [-w] [-l] [fichier]`
- `head [-n N] [fichier]`
- `tail [-n N] [fichier]`
- `tr set1 set2`
- `sort [-n] [-r] [fichier]`
- `cut [-d delim] [-f champs] [fichier]`

Dans toutes ces commandes `fichier` est optionnel : s'il n'est pas spécifié, la commande lit les données sur son **entrée standard** (le clavier).

Commandes de base
Redirections et tubes



redirections

- commande > fichier
- commande > > fichier
- commande &> fichier

tubes

- com1 | com2
- com1 | tee com2

Commandes de base
La commande **cat**

`cat [fichier...]` recopie le(s) fichier(s) (ou l'**entrée standard**) sur la **sortie standard**

Exemples

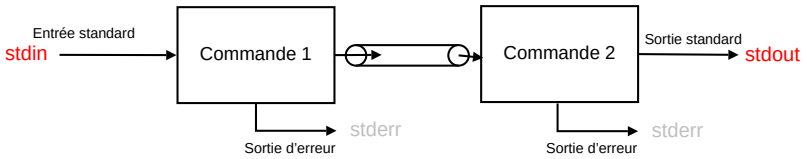
```
cat toto           # affiche toto
cat toto tata      # affiche toto puis tata
cat > toto         # saisie clavier, ecrit dans toto
cat < toto > tata  # copie toto dans tata
cat                # affiche tout ce que l'on tape !
```

Fin de l'entrée standard

lorsqu'une commande lit sur l'entrée standard, terminer par CTRL-d !

Commandes de base
Tubes

```
commande1 | commande2
```



Exemples

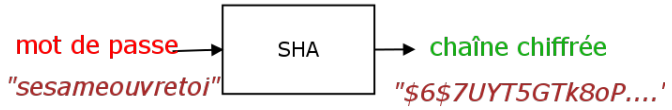
```
ls | wc -l                    # nombre de fichiers
head -n 10 toto | tail -n 1   # 10ieme ligne de toto
```


Partie 3

Utilisateurs, groupes et droits

Utilisateurs et groupes
Mots de passe

On ne conserve jamais un mot de passe “en clair” !



La forme chiffrée est stockée dans `/etc/shadow`, qui n’est lisible que par `root`.

Sécurité

- attaques de type “dictionnaire” (plus facile si on dispose de la forme cryptée)
- approches “sociales” : deviner ou amener l’utilisateur à divulguer son mot de passe...

Utilisateurs et groupes
Fichier /etc/passwd

/etc/passwd

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
viennet:x:626:103:Emm. V.:/users/viennet:/bin/bash
```

Utilisateurs et groupes
Fichier /etc/group

/etc/group

```
ftp::50:
nobody::99:
users::500:adr,ast,ccb,chris,emmanuel
local::501:adr,ast,jm2
```


Utilisateurs et groupes
commandes de base à connaitre

- `su [-] [utilisateur]`
- `passwd [utilisateur]`
- `id [utilisateur]`
- `who [am i]`

Utilisateurs et groupes
gestion des utilisateurs

- Ajout d'un utilisateur :

  `# useradd dupond`

  puis, si on veut lui donner un mot de passe :

  `# passwd dupond`

  Voir `man useradd` pour plus d'options.
  Le répertoire HOME est créé à partir de `/etc/skel`

- Suppression d'un utilisateur :

  `userdel dupond`

  (ne supprime pas le HOME sauf si option -r)

Droits sous UNIX

- types de droits :
  - lecture (r)
  - écriture (w)
  - exécution (x)
- catégories d'utilisateurs vis à vis d'un fichier :
  - propriétaire (u)
  - groupe (g)
  - autres (o)

La commande `ls -l` affiche les droits

```
-rw-r-r- 1 viennet a3 1855 2005-11-29 09:39 toto
```

ordre : propriétaire, groupe, autres

- `chown [-R] propriétaire[:groupe] fichier...`
- `chgrp [-R] groupe fichier...`
- `chmod [-R] categorie+-droit fichier...`

Droits sous UNIX

chmod et notation octale

`rwxrwxrwx` : 9 bits

Exemple

droits	binaire	octal
`-----x`	000000001	1
`rwx----`	111000000	700
`rw-r-r-`	110100100	644
`rwxrwxrwx`	111111111	777

La commande `chmod` avec droits en octal

```
chmod 777 fichier
```


Droits sous UNIX

Cas particuliers

Droit `x` sur les répertoires

Indique que l'on peut *traverser* le répertoire.
Pour lister son contenu, il faut le droit `r`.

Droit SUID

Indique que le fichier est exécutable et que le processus s'exécutera avec *l'identité du propriétaire* du fichier.
Exemple : utilisation de SUID pour accéder à des données protégées

```
$ ls -l /usr/bin/passwd
-rwsr-xr-x  1 root root 26616 2005-05-18 08:33 /usr/bi
```

Droit `t` sur les répertoires

Tout le monde peux écrire, les fichiers ne peuvent être effacés que par leur propriétaire.

Partie 4

Linux : distributions, paquetages, configuration

Linux : une version libre d'UNIX

- Linux est une version *libre* d'UNIX
  - libre car gratuite (mais peut être vendue avec des services ajoutés)
  - libre comme la parole (code source consultable et modifiable)
- Le projet Linux débute en 1991, version sur PC
- ... actuellement disponible sur de nombreuses architectures
- ... est devenu la version d'UNIX la plus répandue
- ... et en constante évolution

Principaux composants

- noyau ("kernel")
- outils (logiciels GNU)
- serveur graphique (X-Windows ou "X11")
- environnements graphiques (Gnome, KDE)

Distributions Linux

Une *distribution* est un ensemble de logiciels configurés autour du système linux. Chaque distribution définit son mode de distribution des logiciels (paquetages).

Principales *distributions* en 2020

- Contributives (libres) :
  - Plutôt pour les serveurs : Debian
  - Plutôt pour les clients (desktop, portables) : Ubuntu, Ubuntu MATE, Mint, Mageia...
  - Pour les systèmes embarqués Raspberry Pi : Snappy Ubuntu Core
- Commerciales :
  - RedHat, Suse

+ Nombreuses distributions spécialisées pour des usages spécifiques : sécurité, jeux, embarqué, etc.

Installation de logiciels et gestion de paquetages

- différentes techniques d'installation (sources, binaires)
- gestion des dépendances
- différents gestionnaires de paquetages (formats tgz, rpm, deb) et outils associés (apt, yum, up2date...).
- problèmes de mises à jour (actualisation, sécurité)

Installation de logiciels et gestion de paquetages (RPM)

Commande rpm (RedHat)

- paquetages = fichiers `.rpm` (archive + scripts de configuration)
- installation : `rpm -i fichier.rpm`
  (ou mieux `rpm -Uvh fichier.rpm`)
- suppression : `rpm -e nom_de_paquet`
- information sur un paquet installé : `rpm -qi nom_de_paquet`
- liste des paquets installés : `rpm -qa`
- de quel paquet vient un fichier ? `rpm -qf fichier`

Installation de logiciels et gestion de paquetages (Debian)

Commandes Debian, Ubuntu, etc.

- paquetages = fichiers `.deb` (archive + scripts de configuration)
- installation : `apt-get install nom_de_paquet`
- suppression : `dpkg -remove nom_de_paquet`
- information sur un paquet installé : `dpkg -L nom_de_paquet`
- liste des paquets installés : `dpkg -l`
- de quel paquet vient un fichier ?
  `dpkg -S /chemin/vers/fichier`

Installation et configuration de Linux

Types d'usages courants

- serveur (avec ou sans GUI, réseau, services)
- station de travail (GUI, réseau)
- usage personnel (dual-boot ?)

Installation (exemples sur PC)

- matériel, pilotes, auto-détection
- affichage : notions sur X11 (paramétrage de base, résolution)
- partitions
- chargeur : grub, lilo
- choix des logiciels (paquets)
- méthode d'installation (Clé USB, DVD, NFS, images)

Partie 5

Configuration de base de Linux

Configuration de base de Linux

Notion de "niveau d'exécution" (runlevel) de system V

Le système fonctionne dans un "runlevel"

- niveau 0 : arrêt
- niveau 1 : mono utilisateur (« single user »)
- niveau 2 : multi-utilisateur, sans NFS
- niveau 3 : fonctionnement normal
- niveau 5 : normal, avec graphique (X11)
- niveau 6 : reboot en cours

Dans chaque niveau, certains services sont activés ou désactivés.

Ce système est remplacé par **systemd**, plus efficace et flexible.

Contrôle des services avec systemd

`systemctl start <service>`	démarre un service
`systemctl stop <service>`	arrête un service
`systemctl restart <service>`	redémarre un service
`systemctl reload <service>`	recharge fichiers de configuration
`systemctl status <service>`	affiche état du service
`systemctl enable <service>`	active au *boot* (démarrage)
`systemctl disable <service>`	désactive au *boot*

Exemple : lance serveur ssh

```
systemctl start ssh
```


Principaux fichiers de configuration

`/etc/passwd, /etc/shadow`	définition des utilisateurs
`/etc/group`	définition des groupes
`/etc/hosts`	noms <-> adresses IP
`/etc/resolv.conf`	adresses serveurs DNS

Sous RedHat et autres linux semblables :

`/etc/sysconfig/network` variables pour config. réseau
`/etc/sysconfig/network-scripts/ifcfg-eth0` config. eth0

Sous Debian et apparentés :

`/etc/network/interfaces`
configuration des interfaces réseau (eth0, ...)

Configuration réseau de base (RedHat)

- Fichier `/etc/sysconfig/network`

Exemple

```
NETWORKING=yes
HOSTNAME=totoro.univ-paris13.fr
GATEWAY=10.10.0.1
NISDOMAIN=tonarino.univ-paris13.fr
```


Configuration réseau de base, suite (RedHat)

- Configuration de chaque interface (eth0, eth1, ...)
  Fichier `/etc/sysconfig/network-scripts/ifcfg-eth0`

Exemple (configuration client DHCP)

```
DEVICE=eth0
BOOTPROTO=dhcp  # pour un client DHCP
ONBOOT=no
TYPE=Ethernet
```

Exemple (avec une IP fixe (statique))

```
DEVICE=eth0
BOOTPROTO=none # IP fixee dans ce fichier
IPADDR=10.23.0.1
NETMASK=255.255.255.0
ONBOOT=no
TYPE=Ethernet
```

Configuration réseau de base, famille Debian

- Configuration des interfaces (eth0, eth1, ...)
  Fichier `/etc/network/interfaces`

Exemple (configuration client DHCP sur eth0 (en IPv4))

```
auto eth0
allow-hotplug eth0
iface eth0 inet dhcp
```

Exemple (avec une IP fixe (statique))

```
auto eth0
iface eth0 inet static
        address 192.0.2.7
        netmask 255.255.255.0
        gateway 192.0.2.254
```


Utilisation simple des journaux (logs)

- Journaux dans `/var/log/`

Exemple

exemple : `/var/log/message` :

```
# tail -2 /var/log/messages
Nov 17 21:40:11 paris sshd(pam_unix)[10483]: session closed for us
Nov 17 21:43:42 paris su(pam_unix)[10600]: session opened for user
```

- "rotation" des journaux (**logrotate**)
- Centralisation des messages par le service **syslog**
  Les processus envoient leurs messages au daemon syslogd
  (configuration dans /**etc/syslog.conf**)
- Commandes utiles :
  - `dmesg`
  - `tail -f /var/log/messages`

Services réseaux

Service (daemon) écoutant sur port TCP ou UDP

Liste des ports standards dans /etc/services

Nom	N° port	Protocole	Commentaire
FTP-DATA	20	TCP	Données de transfert de fichier
FTP	21	TCP	Commandes de transfert de fichier
SSH	22	TCP	Secure Shell (connexion à distance sécuri
Telnet	23	TCP	Terminal virtuel distant
SMTP	25	TCP	le courrier électronique
daytime	13	UDP, TCP	Donner l'heure du jour ! (utile pour des tes
domain	53	UDP, TCP	Serveur de noms
TFTP	69	UDP	Trivial File Transfer Protocol
WWW	80	TCP	World Wide Web
RPC	111	UDP, TCP	Remote Procedure Call
login	513	TCP	Utilisé pour rlogin : se "loguer" sur une ma
NFS	2049	UDP	Network File System daemon : montage d

Services réseaux : exemples du service daytime

Service daytime sur port 13 (TCP)

1. Le client n'envoie rien (ouvre connexion TCP)
2. le serveur renvoie la date (chaine de caractère ASCII).

Ce type de service peut s'utiliser avec telnet :

```
# telnet localhost 13
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
17 NOV 2004 22:00:01 CET
Connection closed by foreign host.
```

Services réseaux : problèmes associés

- Utilisation d'une communication réseau : vérifier la configuration du firewall (pare-feu) : **netfilter (iptables)** sous linux
- Diagnostics en cas de problème :
  - depuis le serveur : utiliser la commande `netstat -a`
  - depuis l'extérieur : scanner, par exemple **nmap** (à utiliser avec précaution)
  - dans certains cas, recourir à l'analyse de trafic (tcpdump, ethereal) pour cerner le problème.
- Problèmes de sécurité si vulnérabilité du logiciel serveur...
  - dépassement de tampons (*buffer overflow*)
  - dénis de services (DoS)
  - injection de code SQL, ...

Dæmons (ou services)

Les dæmons (*Disk and Execution Monitor*)

- Processus s'exécutant en arrière plan
- souvent lancés au démarrage
- répondant à des requêtes

Exemples : `inetd, httpd, nfsd, sshd, named, ...`

Les dæmons unix correspondent aux services de Microsoft Windows.

(x)inetd : le méta-daemon

- Surtout utile pour services réseaux peu utilisés
- Écoute plusieurs ports à la fois
- Ne démarre le daemon correspondant qu'au besoin
- le "tue" après un certain délai de non utilisation
  - Economie de CPU et de mémoire
- sous UNIX : inetd ou xinetd (version étendue)
  - `/etc/inetd.conf` : une **ligne** par daemon/service
  - `/etc/xinetd.d` : un **fichier** par daemon/service
    - ⋆ `/etc/xinetd.conf` : les valeurs par défaut.

Partie 6

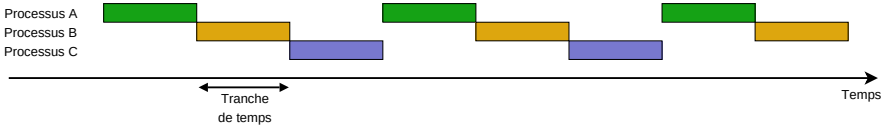
Les processus d'UNIX

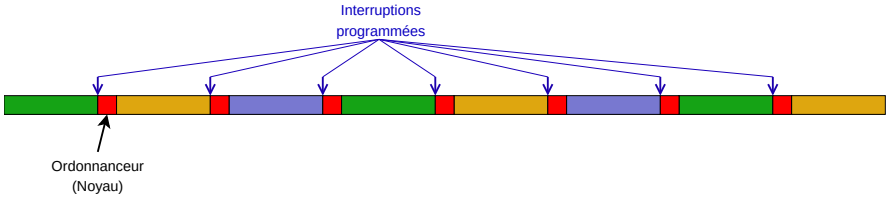
Processus : définition

Un processus est un programme en “cours d’exécution”.

Partage du temps entre processus

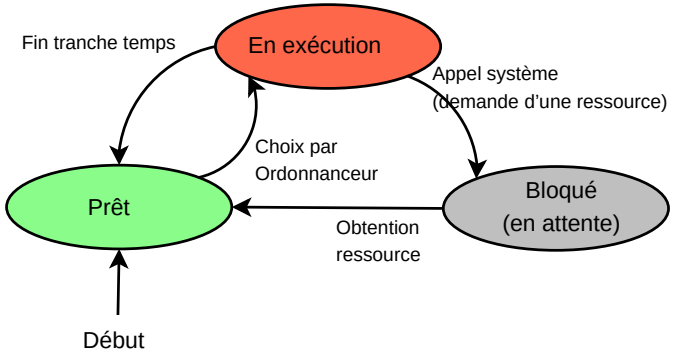
Principe : exécuter à tour de rôle les processus





L’*ordonnanceur* est le programme du noyau qui gère l’ordre d’exécution des processus. Il utilise un *timer* (programmeur d’interruptions) pour interrompre l’exécution de chaque processus à la fin de sa tranche de temps.

États d’un processus



Un processus “en exécution” occupe le processeur.

Chaque appel système bloquant (ex : entrées/sorties) met le processus en attente ; le processeur est alors affecté à un autre processus pendant que le périphérique utilisé travaille (affichage, disque dur, ...).

Attributs d’un processus UNIX

Le système attribue à chaque processus :

- de la mémoire
- du temps processeur

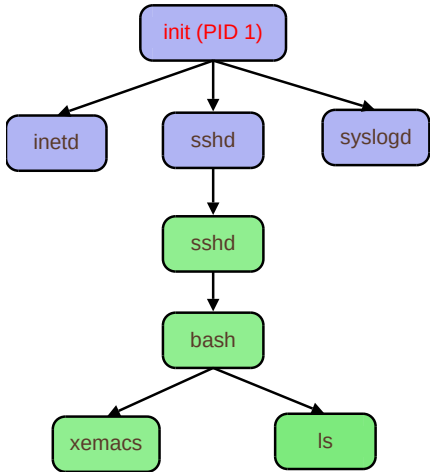
Quelques attributs importants :

- PID : nombre entier 16 bits
- PPID : PID du processus parent
- UID : utilisateur propriétaire
- priorité : nombre entre -20 (très prioritaire) et +20 (non prioritaire)
- temps d’exécution (réel, processeur)
- terminal de contrôle (tty)
- répertoire courant

Tâche (*Threads*)

- Un thread (ou “processus léger”) est un processus à “l’intérieur d’un processus”.
- Les ressources allouées à un processus (temps processeur, mémoire) sont partagées entre les threads qui le composent.
- Un processus possède au moins un thread.
- Contrairement aux processus, les threads partagent la même zone mémoire (espace d’adressage), ce qui rend très facile (et périlleux !) la communication entre threads.
- Chaque thread possède son propre environnement d’exécution (valeurs des registres du processeur) ainsi qu’une pile (variables locales).
- Certains langages, comme JAVA, définissent leur propre mécanisme de threads, afin de permettre l’utilisation facile et portable des threads sur tous les systèmes.

Hiérarchie des processus



Le premier processus lancé est init, avec le PID 1.
Les processus se créent par “duplication” (appel système `fork()`)

Communication inter-processus : signaux

Les signaux offrent un mécanisme de communication et de contrôle inter-processus simple.

- chaque signal est identifié par un numéro de 1 à 31
- certains signaux déclenchent des actions prédéfinies (par ex. terminer le processus)
- le programmeur peut décider d’ignorer ou de réagir différemment à certains signaux
- le shell envoie automatiquement certains signaux en réponse à ces combinaisons de touches (par ex. CTRL-c ou CTRL-z)

Principaux signaux

Numéro	Nom	Signification
1	HUP	fin de session
2	INT	interruption clavier (ctrl-c)
8	FPE	exception calcul virgule flottante
9	KILL	fin du processus (non modifiable)
10	USR1	définissable par l’utilisateur
11	SEGV	référence mémoire invalide
12	USR2	définissable par l’utilisateur
15	TERM	signal de fin
17	CHLD	terminaison d’un fils
18	CONT	reprise d’un processus stoppé
19	STOP	stoppe (non modifiable)
20	TSTP	stoppe (depuis clavier (ctrl-z))
29	WINCH	redimensionnement de fenêtre

Autres mécanismes de communication : tubes, sockets unix

Permettent l'échange de flux de données entre processus.

Tubes

- communication unidirectionnelle
- entre deux processus ayant un ancêtre commun
- en langage C, voir appels `pipe()` et `popen()`

Sockets UNIX

- communication bidirectionnelle
- objet dans le système de fichier
- communication entre processus quelconques

Liste : commande ps

- par defaut : liste les processus attaché à ce terminal
- trop nombreuses options...
  1. pour sélectionner les processus à lister
  2. pour spécifier les informations à afficher, et leur format

Exemples utiles :

- tous les processus : **ps auxww**

```
$ ps auxww | head -1
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
```

- groupement hiérarchique ("généalogique") **ps afu**

```
$ ps afu | head -1
USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
```


Signaux : commande kill

```
kill [-signal] pid...
```

Envoie un signal à un ou plusieurs processus.
Par défaut, envoie le signal TERM (15), qui peut être ignoré ou redéfini.

Exemples

`kill -9 1480`	envoie le signal 9 (KILL) au processus 1480
`kill -USR1 1480`	envoie le signal 10 (USR1)

Lancement d'une commande par le shell

Après décodage de la ligne de commande (substitution des variables, métacaractères, ...), le shell recherche la commande :

- dans les fonctions internes ("builtins")
- puis dans chacun des répertoires indiqués par la variable d'environnement **PATH**.

Exemples

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/usr/games:
/home/viennet/bin
```

which

La commande **which** indique la commande utilisée par le shell :

```
$ which ls
/bin/ls
```

Variables d'environnement

- ensemble de variables (chaînes de caractères) associées aux processus
- modifiables, héritées d'un processus à l'autre
- chaque processus sa propre copie des variables.

Utilisation dans le shell bash

```
export X="toto"
echo $toto
printenv # affiche toutes les variables
```

Utilisation en C

voir fonctions `setenv()` et `getenv()`.

Gestion des processus interactifs (jobs)

- Le shell gère une liste des processus qu'il a lancé (commandes), qu'on appelle des travaux (jobs).
- Les *jobs* sont indentifiés par un numéro (1, 2, 3...)
- La commande `jobs` affiche la liste des *jobs* :

Exemples

```
$ jobs
[1]   Running    xemacs CoursLDAPGTR2.tex &   (wd: \~/tmp)
[2]   Running    xpdf ../Cours-01/Cours-IntroSE.pdf &
[3]   Running    oocalc ../plan2005.sxc &
[6]   Running    xpdf Cours-Processus.pdf &
[7]-  Running    dia partagetemps.dia &
```

- CTRL-c et CTRL-z affectent le job en premier plan (*foreground*).
- Manipulation des jobs : commandes `fg`, `bg`, `kill %n`

Traitements différés

Les commandes lancées par le shell sont attachées à son terminal et sont tuées lorsqu'on le quitte.

- `nohup` permet d'indiquer que la commande doit être détachée du terminal, et continuer à s'exécutée après le départ de l'utilisateur.

Exemples

```
$ nohup du -sm /* &> /tmp/resultat
```

- `at` spécifie que la commande doit être lancée à l'heure indiquée.

Exemples

```
$ at 23:59 12/05/05
warning: commands will be executed using /bin/sh
at> echo "dodo les enfants" > /tmp/toto
```

Plus d'infos...

Voir commandes `atq`, `atrm`, et `/etc/at.allow` et `/etc/at.deny`

Traitements périodiques (cron)

Commandes que l'on veut lancer régulièrement (chaque minute, chaque jour ou chaque année...)

- "tables" de traitements : crontab associées à chaque utilisateur et au système (`/etc/crontab`)
- crond est un dæmon qui vérifie chaque minute les tables et lance les traitements.
- la commande `crontab -e` permet à un utilisateur de modifier sa crontab

Exemples

Faire le café tous les jours à 8h30, mais le dimanche à 11h45 :

```
# m h  dom mon dow      command
30 08 * * 1,2,3,4,5,6  /usr/local/bin/faire_cafe
45 11 * * 7            /usr/local/bin/faire_cafe
```