esiea
ECOLE D'INGENIEURS
DU MONDE NUMERIQUE

SYSTÈME

Linux : la gestion des processus

By Rexy

Table des matières

GENERALITES

Définitions

Linux est un système d'exploitation Mufti-Tâches et Multi-Utilisateurs. A tout moment, plusieurs tâches ou programmes s'exécutent en même temps.

- **Première définition** : le processus est **l'unité élémentaire de traitement** gérée par Linux.
- **Autre définition** : un processus est **une occurrence** (de tout ou partie) **d'un programme en cours d'exécution.**

En conséquence, plusieurs occurrences d'un même programme peuvent s'exécuter en même temps.

Un processus est constitué d'un programme exécutable (compilé ou assemblé) et de ressources nécessaires à son exécution (pile, zone mémoire, informations de service). On regroupera l'ensemble sous la notion de **contexte**.

C = Code

D = Donnée

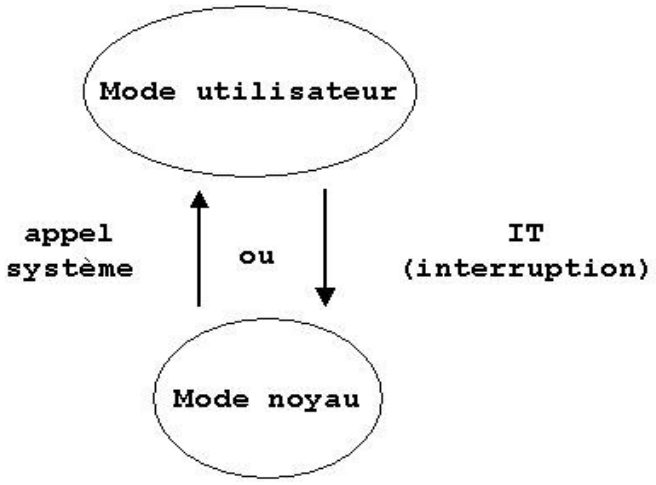
P = Pile

C
D
P

Pile : zone mémoire réservée par le programme où celui-ci empile les adresses de retour au code suite aux appels système successifs.

La phase de chargement d'un programme fait passer celui-ci de l'état inerte à l'**état actif** (processus). Il se retrouve alors en mémoire centrale et dispose de tous les moyens nécessaires à son exécution. Quand le processus est actif, deux modes de fonctionnement sont distingués :

- le mode ***utilisateur*** par défaut : le processus exécute les instructions de son code ;
- le mode ***noyau*** adopté par le processus quand il fait appel à une primitive système pour accéder à des éléments (ressources) inaccessibles en mode utilisateur.



Caractéristiques

Les processus présentent un certain nombre de caractéristiques abordées et détaillées dans chacun des paragraphes qui composent ce support.

- Chaque processus est **identifié** de manière **unique** par un numéro (**PID** = Process Identifier). Ce numéro est l'identificateur utilisé par le système pour gérer le processus.
- Un processus est **toujours engendré par un autre processus**.
- Il existe, dans le fonctionnement de Linux, non seulement un principe important de **parenté** mais aussi un principe d'**héritage entre les processus**, qui conditionne le comportement de

ceux-ci.

TYPES DE PROCESSUS

La commande « ps »

Il existe une commande permettant d'avoir une **photo des processus** (snapshot = instantané) en train de s'exécuter sur la machine : la commande **ps**.
En pratique, cette commande est surtout utilisée pour avoir des informations sur les processus s'exécutant en arrière-plan (programme s'exécutant de manière invisible pour l'utilisateur en tâche de fond).

Syntaxe : ps [-faxl...]

La commande sans options affiche les principales informations sur les processus relatifs à l'utilisateur connecté au terminal.
Les options **-a et -x** permettent de visualiser les processus de **tous les utilisateurs (**notamment ceux de "root"), et les processus qui ne sont pas liés à un terminal (processus système).
Les option **-f** et **-l** permettent un affichage long et sous forme d'arborescence (hiérarchie entre processus).
Unix distingue 2 types de processus :

- Le processus utilisateur,
- Le processus système ou démon.

Le processus utilisateur

- Il est lancé depuis un terminal associé à un utilisateur.
- Il s'exécute en mode utilisateur (écran, clavier, utilisateur)
- Il accède aux ressources de la machine par l'exécution de requêtes système.

Exemple :

```
[root] # ps -f
USER  PID    PPID  C  STIME     TTY       TIME  CMD
lce  22219   32191  5 09:29:36  pts/0     0:00  ps -ef
lce  32191   18878  0 09:29:24  pts/0     0:00 /bin/bash
```

La valeur "pts/0" renseigne sur le nom du terminal (résultat de la commande tty) associé à l'utilisateur "lce". Le chiffre "0" indique qu'il s'agit du premier terminal (associé au ficher spécial "/dev/pts/0").
Dès la connexion de l'utilisateur, le système lance l'interpréteur de commande associé à cet utilisateur (voir fichier de configuration des utilisateurs reconnus par le système = /etc/passwd). Dans l'exemple ci-dessus, il s'agit de « Born Again SHell » ou « bash ».
Pour l'utilisateur, le shell (ici bash) est le père de toutes les commandes lancées par ce dernier (notion vue plus tard).

Le processus démon (DAEMON)

Définitions

DAEMON = Deffered Auxiliary Executive MONitors.

C'est un processus **géré par le système** et donc rattaché à aucun terminal. Il s'exécute en mode utilisateur et effectue des requêtes systèmes.

Nombreux à être chargés lors du démarrage de la machine, ils restent **résidents en mémoire centrale en attente d'une requête** d'un utilisateur ou du système lui-même.

Exemples :

Le démon ***lpd*** guette la requête d'impression de l'utilisateur,

Le démon ***cron*** scrute toutes les minutes les fichiers de demande de travaux des utilisateurs.

Caractéristiques

Les caractéristiques d'un processus démon sont les suivantes :

- ✓ tâche non intégrée dans le noyau,
- ✓ exécuté en arrière-plan.

L'administrateur peut, grâce à des commandes adéquates, voir leur statut (arrêté ou démarré), puis les arrêter et les redémarrer.

L'administrateur utilise ces commandes pour modifier le(s) fichier(s) de configuration associé(s) au processus démon. Pour que ce dernier puisse prendre en compte dynamiquement la modification, le système n'a pas besoin d'être réamorcé (comme dans certaines circonstances sous les systèmes Windows). L'administrateur doit seulement arrêter et redémarrer le processus.

Ainsi, l'administrateur peut, au cours d'une session, lancer des démons qui ne le seraient pas au démarrage du système. La liste des démons à activer dès le démarrage est contenue dans les fichiers d'initialisation du système.

Exemples

Dans le résultat de la commande ps –ef , le caractère "?" (LINUX, SCO), ou le caractère "-" pour AIX, dans la colonne TTY indique les processus démons.

Exemple :

```
[root] # ps –ef | more
UID  PID   PPID   C    STIME    TTY   TIME         CMD
root  1     0     0    10:24     ?   00:00:00     init [5]
root 603    1     0    10:24    tty1 00:00:00     login – lpe
lpe  720   603    0    10:43    tty1 00:00:00     -bash
```

Quelques démons :

- ✓ ***init*** : premier à créer les autres démons,
- ✓ ***swapper*** : gère la zone de swap ou extension de mémoire virtuelle ;
- ✓ ***scheduler*** :attribut des temps processeur aux différents processus à

exécuter (ordonnanceur) ;

- ✓ ***lpd*** ou ***lpshed*** : gestion des impressions (spooler) ;
- ✓ ***cron*** : gestion des tâches
- ✓ ***inet*** : services réseau

CONTEXTE D'UN PROCESSUS

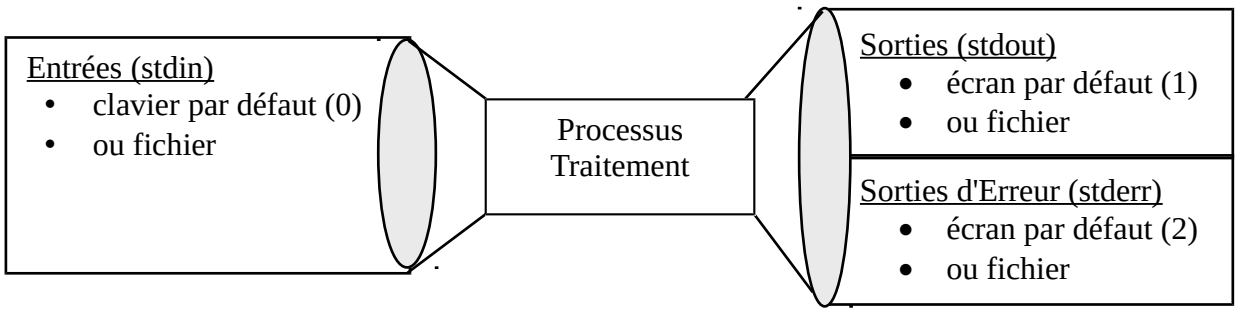
Ce paragraphe est consacré à l'environnement d'exécution d'un processus.

Les fichiers standards

Le noyau d'UNIX gère une table des fichiers ouverts pour chaque processus exécuté. Dans cette table, les fichiers sont référencés par des descripteurs. La table est initialisée à trois entrées : les **fichiers standards**.

Ces descripteurs permettent au processus de dialoguer avec l'environnement extérieur : ce sont les fichiers standards en entrée, en sortie et en sortie d'erreur.

- ✓ Fichier standard d'**ENTREE**
  - ✓ stdin
  - ✓ descripteur **0**
- ✓ Fichier standard de **SORTIE**
  - ✓ stdout
  - ✓ descripteur **1**
- ✓ Fichier standard d'**ERREUR**
  - ✓ stderr
  - ✓ descripteur **2**



Par défaut, ces 3 fichiers sont liés au terminal : un processus lit ses données sur le clavier (entrée standard 0), écrit ses résultats sur l'écran (sortie standard 1), imprime les erreurs d'exécution sur l'écran (sortie d'erreur standard 2).

Il est possible de rediriger les fichiers standards auxquels est lié un processus.

Les re-directions

Il existe deux modes de re-direction en sortie et sortie d'erreur :

- ✓ mode **création**
- ✓ et mode **ajout**.

Le premier mode permet d'écrire les données traitées dans un fichier.

Si ce fichier n'existait pas au préalable, il est créé.

Si ce fichier existait, le fichier est alors réécrit (écrasé) avec les nouvelles données.

La seule différence entre le premier et le deuxième mode est, dans le cas du deuxième mode, d'ajouter les données traitées en fin de fichier existant.

Entrée standard <

Pour rediriger l'entrée standard (clavier) d'un processus à partir d'un fichier, le signe < (**inférieur**) est utilisé.

Au lieu de saisir des données au clavier, le processus redirigé peut les récupérer directement dans un fichier.

```
[root]  # wc -l  < fic1
```

La commande ***wc*** assortie de l'option ***-l*** compte les lignes du fichier "fic1".

Sortie standard > ou 1> (création)

Pour rediriger la sortie standard (écran) d'un processus vers un fichier, on utilise la syntaxe : ***prog 1> fic*** . Si aucun chiffre ne précède le signe > (**supérieur**), le système rajoute ***1*** par défaut.

Le fichier de re-direction est créé s'il n'existe pas ou réécrit (écrasé) s'il existe.

```
[root]  # cat fic1 fic2 > ficsou
```

L'affichage des fichiers *fic1* et *fic2* réalisé par la commande ***cat*** est redirigé dans le fichier ***ficsou***.

Sortie d'erreur 2> (création)

Pour rediriger la sortie standard d'erreur, il faut utiliser la syntaxe suivante :

```
[root]  # ./prog.sh 2> prog.err
```

Après exécution de la commande, le fichier ***prog.err*** contiendra les erreurs survenues lors de l'exécution du processus ***prog.sh*** .

Cette opération est surtout utilisée à l'intérieur d'un script de commandes shell.

Remarque : Il existe sous Linux un fichier spécial correspondant à un périphérique inexistant : ***/dev/null*** .

Dans le cadre de programmes écrits en langage de commandes shell, ce fichier est utilisé comme fichier de re-direction aussi bien pour l'affichage normal que l'affichage des messages d'erreur, pour limiter l'affichage au seul résultat final du programme.

```
[root]  # ls  -l fichier  1> /dev/null   2> /dev/null
```

L'exemple ci-dessus n'a qu'une valeur pédagogique. L'utilisateur dans ce cas redirige à la fois la sortie standard (**1>...**) et la sortie d'erreur (**2>...**) vers le néant.

La syntaxe ci-dessus peut être simplifiée par : ***1> fic 2>&1*** ou ***2> fic 1>&2 .***

Sortie standard >> (ajout)

Avec le signe >> (supérieur, supérieur), les données envoyées sur la sortie standard sont ajoutées en fin de fichier existant.

Sortie d'erreur 2>> (ajout)

Cette opération permet de constituer un fichier historique des erreurs survenues lors de l'exécution d'un processus donné.

Spéciale d'entrée standard <<

Cette opération est utilisée par les utilisateurs ayant déjà une certaine expérience du monde UNIX.

Il est ainsi possible d'écrire un fichier (ou un script) sans ouvrir d'éditeur de texte spécifique.

Une contrainte existe : il faut coller derrière le signe de re-direction une chaîne de caractères (ici "***EOT***") qui sera interprétée comme étant le début du fichier. A partir de là, tout ce qui suit la chaîne de délimitation du fichier sera traité comme venant du clavier. Le fichier est ainsi créé ligne par ligne (taper "entrée" en fin de ligne) jusqu'à rencontrer la même chaîne de délimitation comme fin de fichier (ici "***EOT***").

```
[root] # cat <<EOT
Bonjour aux utilisateurs,
Une opération de maintenance est prévue à partir de 16 heures
Veuillez vous déconnecter un quart d'heure avant !
EOT
```

Dans le cas ci-dessus, on ne fait qu'afficher les ligne jusqu'au EOT (End Of Texte). Cette opération n'est pas très utile...

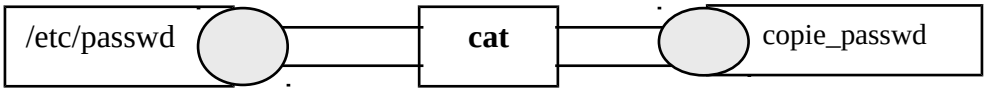
Pour sauvegarder ces quelques lignes dans un fichier, on emploie la double redirection :

```
[root] # cat > message <<EOT
Bonjour aux utilisateurs,
Une opération de maintenance est prévue à partir de 16 heures
Veuillez vous déconnecter un quart d'heure avant !
EOT
```

Ce principe de double redirection peut être utilisée dans n'importe quel ordre comme ci-dessous :

```
[root] # cat < /etc/passwd > copie_passwd
```

équivaut à dupliquer le fichier */etc/passwd.*



Les propriétaires des processus

La règle est que **les droits d'un processus dérivent des droits de son propriétaire**.

Propriétaire réel

Chaque processus possède un propriétaire réel : c'est l'**utilisateur** qui l'a lancé. Par cela même, il possède un groupe propriétaire réel qui est le groupe auquel appartient le propriétaire réel.

La commande ***ps*** affiche en premier champ le nom de l'utilisateur (USER) ou propriétaire réel.

Même si le propriétaire du fichier associé au processus est différent de celui qui le lance, le propriétaire réel reste celui qui lance le processus.

Propriétaire effectif

Le **propriétaire du fichier** associé au processus est appelé le propriétaire effectif. Il en est de même pour le groupe.

Dans le cas strict d'un exécutable binaire devant être lancé par un utilisateur autre que le propriétaire effectif, il faut que ce dernier ait positionné un des deux (ou les deux) bits spéciaux : « ***SET-USER-ID*** » et (ou) « ***SET-GROUP-ID*** » (cf. permissions dans système de fichiers).

Si les bits spéciaux sont positionnés, le propriétaire effectif est le propriétaire du fichier associé au processus. Sinon, le propriétaire effectif et le propriétaire réel sont les mêmes.

Les attributs d'un processus

Les principaux

L'essentiel des attributs des processus est affiché par le retour de la commande "ps -f" :

- ✓ user (ou uid) : propriétaire réel ;
- ✓ pid (process identifier) : identifiant unique ou numéro du processus ;
- ✓ ppid (parent process identifier) : identifiant ou numéro du processus parent (processus père) ;
- ✓ pri (priority) : priorité actuel ;

- ✓ ni : priorité initial ;
- ✓ tty (ou sys) : terminal d'attachement ;
- ✓ cmd (ou command) : commande (avec son chemin).

USER (ou UID) : Il s'agit bien du propriétaire réel (utilisateur), ou plutôt de son nom de connexion, et non du propriétaire effectif (fichier) du processus.

L'***UID*** (User Identifier) correspond au numéro d'identification unique de l'utilisateur.

PID : Pour communiquer avec un processus, il est indispensable de connaître son numéro de PID car le système ne connaît le processus que par son identifiant.

PPID : Tout processus est engendré par un autre processus. Aussi connaître le PPID de ce processus permet de déterminer quel est le processus qui l'a créé (celui qui est son père).

- ✓ Excepté le cas du dysfonctionnement du système d'exploitation, le processus parent est toujours en exécution. Aussi apparaît-il dans l'affichage de la commande "*ps*". Il est ainsi facile de remonter la hiérarchie des processus en notant les différentes valeurs des PID et PPID des processus, ceci jusqu'au premier lancé lors de l'initialisation du système (processus "init" de PID=1).
- ✓ Sous LINUX, la commande "*pstree*" retourne l'affichage de cet arbre « généalogique ».

TTY : Ce champ permet de distinguer s'il s'agit d'un processus utilisateur ou d'un processus système (démon). Parmi les processus utilisateurs, il renseigne sur le terminal de rattachement de l'utilisateur en question.

CMD : Ce champ permet de localiser l'emplacement du processus en question.

PRI et NI : Ces champs sont d'importances secondaires pour l'utilisateur.

A l'aide de la commande "*nice*" (cf. paragraphe "commandes de gestion"), l'utilisateur peut diminuer la priorité d'exécution du processus côté processeur alors que seul l'administrateur est en mesure de l'augmenter.

Le champ **PRI** indique la priorité résultante de la modification effectuée par la commande "*nice*".

Exemple :

```
[root] # ps
USER  PID PPID C STIME TTY TIME CMD
root  1      0  0 Dec 15 -       0:58 /etc/init
```

C : temps d'utilisation du processeur pour le partage

STIME (ou TEMPS) **:** heure de démarrage du processus

TIME (ou HEURE) : temps d'exécution cumulé de la commande

Commande ps (compléments)

L'aide en ligne de cette commande permet de découvrir ses très nombreuses options. Il est inutile de les passer en revue ici dans le cadre d'un module destiné à former les exploitants du système d'exploitation.

Néanmoins, quelques options complémentaires sont explicitées dans l'exemple ci-dessous.

Syntaxe :
ps [options]

Options :

- ✓ -p list processus de PIDs donnés ;
- ✓ -u list processus de certains utilisateurs ;
- ✓ -g list processus de certains groupes.

```
[root] # ps l
F  S  UID   PID  PPID  C  PRI  NI  ADDR   SZ  WCHAN    TTY  TIME  COMD
1  S  10    840  1     0  30   20  1423   16  102652   10   0:04  sh
1  R  10    887  840   6  58   20  1520   32           10   0:02  ps
```

F : identificateur sur la localisation du processus [1= en mémoire vive; 0= terminé]

S : indicateur de l'état du processus [S=en attente; O=actif; Z=zombie].

PRI : priorité (plus le nombre est grand, plus la priorité est petite)

NI : valeur du paramètre *nice* (gentillesse ou amabilité) pour le calcul de la priorité

ADDR : adresse mémoire ou disque du processus

SZ : taille mémoire

WCHAN : événement attendu par le processus

Dans l'exemple cité, le numéro 887 est le PID de la commande ps dont le père est la commande shell de PID 840 (= PPID de *ps*). Le *shell* a pour père le processus de PID 1, c'est à dire le processus init.

ETATS D'UN PROCESSUS

Dans ce paragraphe est abordé le cycle de vie d'un processus actif. L'existence d'un processus peut être divisée en un ensemble d'états permettant de décrire le processus.

En cours d'exécution, un processus exécute des instructions qui composent le code et accède à ses propres données. Il dialogue aussi avec le reste des ressources disponibles par l'intermédiaire des fonctions d'appel au système.

Du fait du caractère multitâche d'UNIX, un processus passe par plusieurs états différents tout au

long de son existence.

L'ordonnanceur (ou scheduler), le périphérique d'entrée sortie, le système, le propriétaire ou l'administrateur, sont les acteurs qui peuvent décider des changements d'états du processus.

Le processus passe en ***état de sommeil*** *(ou endormi)* quand le processus n'est plus exécuté de manière momentanée par le processeur. Le processus reste en général chargé en mémoire.

La transition " actif ⇔ état de sommeil " est gérée par ***l'ordonnanceur*** *(ou scheduler).*

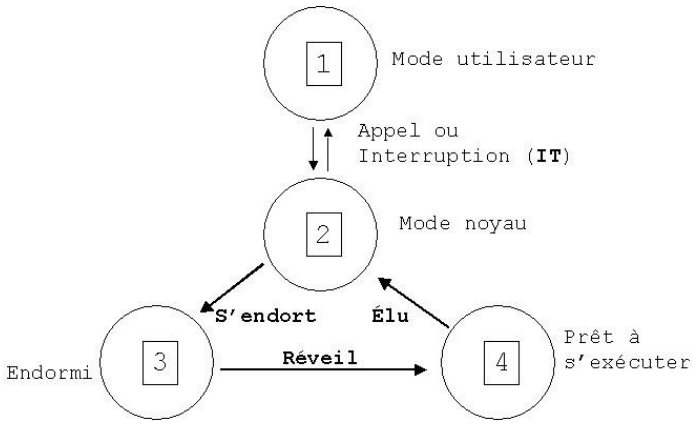
Le processus est en ***attente d'exécution*** *(prêt à ...)* quand il est prêt à être choisi par l'ordonnanceur.

Rappel des conditions d'attribution du processeur à un processus :

Cette attribution est à la charge de l'ordonnanceur qui applique des algorithmes complexes tenant compte de plusieurs paramètres :

- ✓ Quantum de temps
  - ➢ un processus perd la main au bout d'un quantum de temps paramétrable.
- ✓ Entrée / Sortie
  - ➢ un processus perd aussi la main s'il effectue une opération d' E / S.
- ✓ Niveau de priorité des processus (nice)

Le graphe ci-dessous permet non seulement de résumer les quelques principaux états à connaître, mais aussi de visualiser les deux modes de fonctionnement évoqués dans le premier paragraphe : mode utilisateur, mode noyau.



D'autres états existent. Leur étude détaillée n'apporterait rien à une première approche du sujet.

GESTION DES PROCESSUS

Le système UNIX utilise des fonctions systèmes du langage C pour gérer les processus depuis leur activation (création) à leur mort (terminaison).

Parenté des processus

Sous UNIX, un processus est toujours créé par un autre processus.

Un abus de langage consiste à transposer la notion de parenté à ce phénomène : le **processus fils** est créé par un **processus père**.

Seul le premier processus échappe à cette règle dans la mesure où c'est le **noyau** lui-même, une fois chargé au démarrage du système, qui se transforme en processus de **PID égal à 0**. Il amorce le phénomène en créant le processus **init** de **PID égal à 1**, et considéré comme **l'ancêtre de tous les processus**.

Cette notion de parenté entre les processus crée une espèce d'arborescence à partir du processus **init**.

Remarque : Pour UNE seule commande lancée, le processus père est l'interpréteur de commande shell.

La création d'un processus se fait par la fonction C-Système *fork().*

La fonction système *fork ( )*

C'est la seule primitive qui permette de créer un processus sous UNIX. Elle provoque la création, en mémoire centrale, d'un nouveau processus qui est la **copie exacte** du processus " père " qui a appelé la fonction.

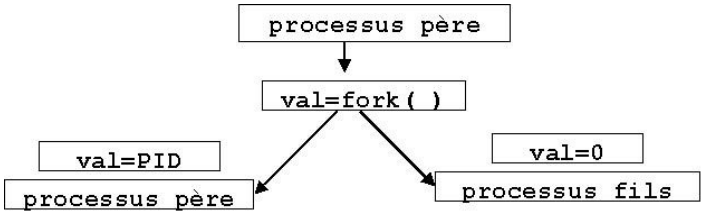
Le processus "fils" **hérite** du contexte du processus père.

A l'issue de fork(), le dialogue entre le processus "père" et le processus "fils" se fait par l'intermédiaire des moyens de communications inter-processus (voir module "Langage C-Système").

La valeur de retour de cette fonction fork( ) sera :

- ✓ **-1** si la création du processus "fils" échoue,
- ✓ **0** pour le processus "fils" créé,
- ✓ **pid** identificateur du processus "fils" renvoyé au processus "père"

Les valeurs ***0*** et ***PID*** sont les seules valeurs qui permettent au système d'exploitation, comme au programmeur en langage C du système UNIX, de différencier les deux processus.



Après exécution de la fonction système, le processus fils est reconnu par le noyau et possède des références dans les tables du système qui lui sont propres. Le processus fils peut alors mener sa propre existence.

Remarque : le processus père attend la valeur de retour du *fork( )* du 1er fils créé pour lancer éventuellement la création d'un 2ème fils, et ainsi de suite.

Fonctionnement séquentiel ou asynchrone

Les processus père et le fils peuvent avoir deux modes de fonctionnement distincts :

Mode séquentiel : le père attend la fin d'exécution du fils pour reprendre la sienne ;
- *Mode asynchrone* : le père et le fils s'exécutent en parallèle.

Dans le cas d'une exécution asynchrone, si le père termine son exécution avant le fils, celui-ci devenu ***orphelin***, est " adopté " par le processus **init** afin de maintenir le principe de filiation.

Si c'est le fils qui s'achève le premier, il deviendra *processus zombie* tant que le père ne sera pas lui-même terminé, ou que le père n'aura pas interrogé le système sur l'état de son fils.

La fonction système de la famille des "*exec... ( )"*

Il existe plusieurs fonctions exec (execl, execlp,...) ayant chacune des paramètres subtilement différents.

Le but de la création d'un nouveau processus est de lui faire exécuter une tâche particulière. Un appel à une des fonctions système *exec...(*) va permettre de pouvoir disposer de deux processus aux comportements réellement différents.

L'appel de cette fonction système provoque le remplacement du contenu du processus appelant par le fichier exécutable du programme indiqué en paramètre à cette fonction. L'exécution de ce programme est automatiquement démarrée.

Une fonction *exec...(*) réalise un ***mécanisme de recouvrement*** car, lors de son exécution, l'ensemble du contexte est recouvert [= écrasé] par celui du nouveau processus.

Le mode d'exécution d'une fonction *exec...( )* est le suivant :

- Une fonction *exec...(*) suit dans presque tous les cas un appel à la fonction *fork( ) ;*
- La fonction *exec...( )* ne renvoie pas de valeur de retour si elle réussit parce que, lors son exécution, le contexte change totalement (le contexte du processus appelant est écrasé). Toutefois, si le programme mesure une valeur de retour égale à "-1", cela signifie que la fonction a échoué.

Malgré le changement de contexte, les autres données système, notamment le PID, sont conservées.

Il est par contre possible de modifier les propriétaires et groupes effectifs si les attributs spéciaux ont été activés au préalable (modification du SET_USER_ID ou du SET_GROUP_ID).

Le propriétaire et le groupe effectif sont alors différents du propriétaire et groupe réel qui représentent 1'utilisateur ayant lancé l'exécution du processus.

La fonction système *exit ( )*

Lorsqu'un processus se termine normalement, il doit appeler la fonction système ***exit( )*** afin de clore proprement l'activité du processus au sein du système.

Cette fonction exécute les opérations suivantes :

- elle ferme tous les fichiers ouverts ;
- elle libère la mémoire utilisée ;
- elle met le processus appelant à l'état de zombie ;
- elle signale ensuite l'événement à tous les processus concernés, notamment au processus père.

Pour réaliser la dernière opération, la fonction système *exit( )* attend un paramètre qui est renvoyé comme valeur de retour au processus père.

La fonction renvoie "0" s'il n'y a pas de problème, sinon renvoie une valeur différente de "0".

La fonction système *wait ( )*

Un processus père peut **synchroniser** son exécution avec celles de ses fils en appelant la fonction système *wait ( ).*

Après l'appel de *wait ( ),* le processus père attend la fin d'un de ses fils. Il ne peut pas attendre la fin d'un de ses fils en particulier. Tout au plus peut-il tester la valeur de retour et appeler de nouveau *wait (*) si la valeur renvoyée n'est pas celle souhaitée.

Si un processus fils se termine avant le père, deux cas peuvent se présenter :

- Si le père s'était mis en attente, avec un appel *à wait ( ),* il sort de cet état. Le fils est effacé des tables système.
- Si le père n'a pas fait appel à *wait ( )* [il a continué à s'exécuter], le processus fils ne peut l'avertir de sa terminaison, il passe alors à l'état *zombie.* Il ne sortira de cet état que lorsqu'un de ses processus " ascendant " réalise lui-même un *exit ( )* ou un *wait ( ).*

Processus orphelin :
Lorsqu'un processus père meurt, ces processus fils sont dits " orphelins ". Ils sont alors adoptés par le processus **init**. Ce processus devient donc le père de tous les processus orphelins.

Commandes synonymes :

S'il n'existe pas de commande *fork*, il existe par contre des commandes synonymes des autres fonctions système de gestion des processus :

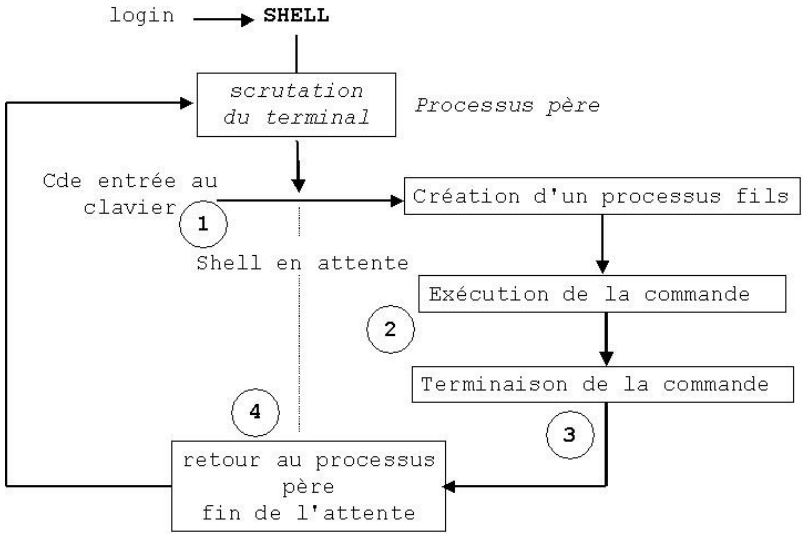
- exec la commande passée en paramètre de *exec* est exécutée à la place de

	l'interpréteur de commande ou shell (sans création d'un nouveau processus).
- wait [n]	le processus invoquant cette commande attend la fin de l'exécution du processus fils de PID égal à "n" ou de tous les fils si *wait* est lancé sans paramètre.
- exit	termine l'activité du programme appelant.

Lancement d'une commande

Il est plus aisé de comprendre la vie d'un processus en étudiant celle d'une commande lancée au clavier par un utilisateur.

Elle s'exécute en quatre phases.



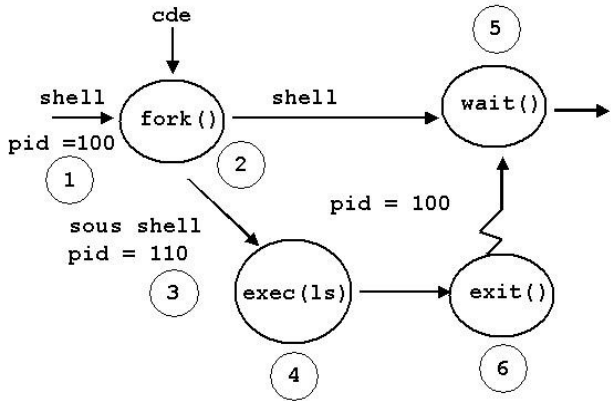
1. Le ***shell***, recevant la demande d'exécution d'une commande saisie au clavier du terminal X, crée un processus à l'aide de la fonction ***fork()***.

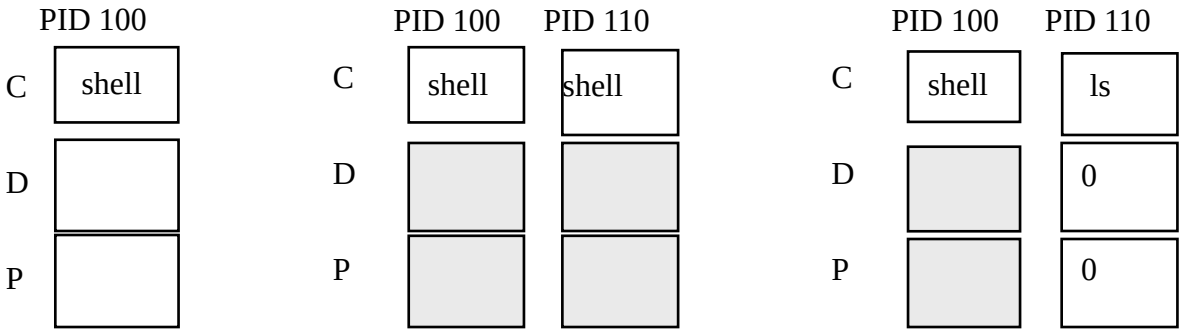
2. La commande (processus fils) s'exécute (appel de la fonction ***exec()***) pendant que le ***shell(père)*** attend (appel de la fonction ***wait()***).

3. L'exécution de la commande terminée, le fils meurt en envoyant un message à son père : 'Je vais mourir ...' (signal ou fonction ***Exit()***).

4. Le père se réveille et reprend le contrôle du terminal.

Déroulement d'une commande





C = Code, D = Donnée, P = Pile.

Phase 1 *Phase 3* *Phase 4*

Les phases :

1. L'interpréteur de commande (shell) est actif depuis la connexion de l'utilisateur. Il dispose de sa propre zone mémoire (code, données, pile), il a un PID égal à 100.

2. La commande saisie au clavier entraîne le shell à créer un processus fils pour exécuter la commande. Il se duplique donc en faisant appel à la fonction fork( ).

3. A l'issu de cette duplication réussie, le processus fils (le shell dupliqué) dispose de son propre identificateur (PID = 110).

4. Le système fait alors appel à la fonction exec( ) qui procède au recouvrement du contexte du shell dupliqué par le contexte spécifique à la commande. Cette dernière peut s'exécuter.

5. Le système met le processus père (shell) en attente grâce à la fonction d'appel wait( ).

6. Une fois que la commande est exécutée, cette dernière sollicite la fonction d'appel exit( ) pour terminer son activité proprement. Il prévient notamment le père qui sort de son attente pour se mettre à l'écoute d'une autre commande.

TERMINAISON D'UN PROCESSUS

La terminaison peut être de deux natures :

- **Volontaire** : arrêt d'activité voulu par le système. Le processus se termine normalement [exit( )].

- **Involontaire :** arrêt d'activité non voulu par le processus mais provoqué par l'utilisateur ou le système.

  Un arrêt encore contrôlé par le système crée un fichier "core" ou "coredump" (vidage mémoire faisant au minimum 100Ko). Une division par 0 peut, par exemple, provoquer ce genre d'arrêt.

  L'emploi de combinaisons de touches ou de la commande **kill** est utilisée pour arrêter involontairement l'activité d'un processus.

  La commande **kill**, étudiée dans le dernier paragraphe du document, permet d'envoyer un signal dont le numéro est indiqué en option au processus dont le PID est indiqué en paramètre.

  Des équivalents existent entre les combinaisons de touches et l'envoi de certains signaux :

  - Arrêt par ctrl+C (ou kill -2 PID) dans le cas d'un processus qui part en boucle infinie ;

  - Arrêt par ctrl+D (ou kill -15 PID) pour terminer proprement l'activité d'un processus (style exit( )) ;

  - Arrêt par une commande kill : ***kill -9 PID*** pour terminer brutalement l'activité du processus ;

  - Suspension par ctrl+Z (ou kill -18 PID)

  Quelle que soit la terminaison , le père est prévenu.

Remarque : On ne tue que les processus dont on est propriétaire.

TUBE DE COMMUNICATION

Le premier moyen créé sous UNIX pour permettre les échanges d'informations entre les processus est le *PIPE* ou *tube de communication.* Il fait partie de l'ensemble des IPC (Inter Process Communication) "Intra-UNIX", c'est-à-dire des moyens de communication entre processus internes à une station donnée.

Il existe deux types de tubes :

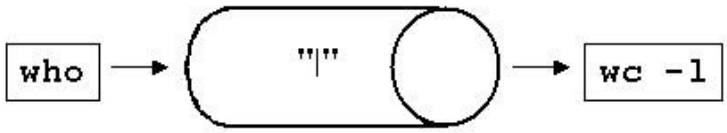
- Le tube mémoire ou simple caractérisé par le signe "|"
- Le tube nommé ou tube fichier

Tube simple

Le principe est celui d'un canal ouvert entre deux processus de façon à ce que les sorties de l'un soient les entrées de l'autre.

On dit que les **processus** sont **affiliés** (il doivent se connaître pour communiquer)**.**

Le principe réside en une **table en mémoire** accessible par les 2 processus.



Exemple d'application :

(a) who > ficwho # rediriger l'affichage de la commande "who" vers le fichier "ficwho"

(b) wc -l < ficwho # compter les lignes de ce fichier grâce à la commande "wc -l" dont l'entrée standard est redirigé.

(c) who | wc -l # le même résultat est atteint en utilisant le pipe mémoire

Remarque : Le père d'une commande utilisant un tube mémoire est toujours (**sauf sous LINUX**) la dernière commande.

En effet, il faut que la dernière commande soit lancée en premier pour qu'elle puisse récolter le fruit de l'exécution de la commande précédente, et ainsi de suite.

Exemple : ls | grep ^p | wc -l

La première commande à fournir un résultat et mourir est la commande « ls ». Elle rend son résultat à son père qui est la deuxième commande « grep ^p », et ainsi de suite.

MAUVAIS	BON
sh fork exec(ls)	sh fork exec(wc -l)

wait	fork	exec(grep ^p)		wait	fork	exec(grep ^p)	
	wait	fork	exec(wc -l)			wait	fork	exec(ls)
		wait					wait	

Tube nommé

Le principe est le même, mais le système utilise comme zone tampon **un fichier** tube nommé (cf . systèmes de fichiers) généralement désigné sous le nom de **FIFO**, et non une table en mémoire**.** Les **processus** sont dits **non affiliés** puisqu'ils ne se connaissent pas.

Les informations écrites sont lues dans l'ordre « premier arrivé, premier sorti » et détruites après consultation.

Quiconque connaît le nom du tube et possède les droits d'accès, pourra ouvrir ce fichier et communiquer par là même avec d'autres processus.

Exemple : le gestionnaire d'impression.

```
prw------  1    root  system     0  Sep   03  15:12    /var/adm/cron/FIFO
```

Il existe d'autres moyens de faire communiquer deux processus : les IPC (Inter Processus Call) ou le mécanisme des sémaphores.

LES SIGNAUX

A tout moment l'utilisateur peut intervenir sur le déroulement d'un processus lui appartenant, c'est le principe des signaux.

Généralités

Des événements extérieurs ou intérieurs au processus peuvent survenir à tout moment et provoquer une interruption de l'exécution du processus en cours.

Lors de la réception d'une interruption, le noyau reconnaît l'origine de celle-ci, sauve le contexte du processus actuel et provoque l'exécution d'une routine appropriée.

La technique des signaux est utilisée par le noyau pour prévenir un processus de l'existence d'un événement extérieur le concernant. Elle correspond chez les autres systèmes d'exploitation au principe des interruptions logicielles. L'origine d'un signal peut être variée : elle peut être matérielle, système ou logicielle. Ce peut être :

- la fin d'exécution d'un processus fils ;
- 1'utilisation d'une instruction illégale dans le programme ;
- la réception d'un signal d'alarme ;
- une demande d'arrêt de la part du processus…

A la réception d'un signal, le système (noyau) prévoit une réaction par défaut du processus. Mais il est possible de redéfinir le comportement du processus (cf. C-Système). Il peut ainsi se prémunir de l'effet de certains signaux, et même les ignorer.

Les différents signaux sont représentés par des numéros auxquels sont associés des noms symboliques utilisés par les programmeurs.

L'ensemble des signaux est décrit dans le fichier ***/usr/include/sys/signal.h.***

La commande kill suivie de l'option "-l" renvoie la liste des signaux.

Selon les systèmes UNIX, leur nombre varie : il y en a au moins une vingtaine.

Fonctionnement d'un signal

Un signal peut être envoyé par le noyau ou un autre processus.

Ce signal est mémorisé dans un champ de la table *proc* du processus récepteur. Chaque signal active un bit particulier de ce champ.

Lorsque le processus récepteur doit être activé ou lorsqu'il réalise un appel système, l'ordonnanceur (***scheduler)*** lit ce champ et fait exécuter les routines correspondantes aux différents bits de signal activés.

- Le délai entre l'émission du signal et son exécution par le processeur est donc indéterminé. Cette caractéristique, qui se rajoute à celle de l'éligibilité d'un processus, fait qu'on ne peut pas considérer UNIX comme capable de gérer des primitives "**temps réel**".

En règle générale, la réception d'un signal provoque l'arrêt involontaire de l'exécution d'un processus avec éventuellement génération d'une image mémoire *core*.

COMMANDES DE GESTION DES PROCESSUS

Il existe un certain nombre de commandes qui sont mises à la disposition d'un utilisateur pour gérer les processus.

Processus en arrière-plan

Un processus utilisateur s'exécute par défaut en avant-plan, et affiche ses résultats par défaut sur l'écran du terminal associé. Pendant le temps de son exécution, l'utilisateur reste en attente (il perd l'invite du terminal) jusqu'à la fin du processus.

L'utilisateur peut décider pour une raison ou une autre de l'exécuter en arrière-plan. Pour cela, il doit lancer le processus en ajoutant à la fin de la commande le signe "**&**" ou esperluète.

Une fois le processus lancé, le système lui rend la main (le père ne fait pas appel à *wait( )* et peut donc continuer à travailler).

Il existe sur certains systèmes UNIX des commandes permettant :

- de ramener un processus en avant-plan (commande **fg** ou foreground),
- de basculer un processus en arrière-plan (commande **bg** ou background),
- de lister les processus en arrière-plan (commande **jobs**).

En effet, le système numérote les processus lancés ainsi qu'il appelle "jobs" selon l'ordre chronologique de lancement (numéros de job).

Ces commandes s'utilisent en leur indiquant le numéro de PID du processus donné (ou numéro de job, ou de nom de processus sous LINUX).

Exemple :

```
sleep 30 &
     [1] 612



CTRL+Z
      [1]+ stopped sleep 10
fg 1
     sleep 30
```

1. La commande "sleep 30" suspend le processus lancé pendant 30 secondes. Le paramètre "&" lançant "sleep 30" en arrière-plan, le système affiche entre crochets le numéro de job suivi du numéro de PID. pour permettre à tout moment de basculer ce dernier en avant-plan. Le système rend la main à l'issu.
2. L'utilisateur suspend le processus grâce à la combinaison "CTRL+Z", le système confirme en retour la suspension.
3. L'utilisateur ramène en avant-plan le job n°1 (processus "sleep 30") qui continue son exécution.

Commande kill

Cette commande permet d'**envoyer un signal** à un processus. En général, cette commande est utilisée pour terminer l'activité d'un processus. Mais certains signaux peuvent être programmés

différemment.

Syntaxe :

kill [-signal] PID

- Le ou les processus destinataires sont identifiés par leur PID.
- La commande ne peut être appelée que pour les processus dont l'utilisateur est propriétaire. L'administrateur peut l'utiliser pour tous les processus.
- Par défaut le signal envoyé est le signal 15 ou SIGTERM (*signal terminate)* qui termine proprement l'exécution du processus qui le reçoit.
- Quand cela ne suffit pas (à cause de masquage), le signal N°9 OU SIGKILL est utilisé. Il est impossible de se prémunir contre la réception de ce dernier.

La liste des signaux disponibles est affichée par la commande : ***kill -l***.

Commande sleep

Le lancement de cette commande suspend l'activité du processus lancé pendant le nombre de secondes indiqué en paramètre.

Syntaxe :

sleep [nombre de secondes]

Commande nice

Cette commande permet d'influencer l'ordre d'exécution des processus. UNIX attribue une priorité d'exécution par défaut aux différents processus. Cette priorité peut être modifiée par le système en fonction de divers paramètres dont le paramètre *nice.* Ce paramètre est tout simplement modifié par la commande ***nice*** associée.

Syntaxe :

nice [-incrément] commandes

La valeur incrément est ajoutée à la valeur courante (maximum 20). Plus le nombre est grand, plus la priorité est faible.

Seul l'administrateur peut augmenter la priorité du processus.

Exemple 1 :

```
ps  -f                # constater les niveaux de priorité avant la
modification
...      PRI   NI
...      62     20  ps

nice –10 ps     # réduire de 10 la priorité actuelle
```

```
...      PRI   NI
...       72    30  ps    # constater la modification de la valeur de
NI répercutée sur celle de PRI
```

Exemple 2 : augmenter la priorité d'un processus est réservé à l'administrateur.

```
nice --10 commande       # augmente de 10 la priorité de commande.
```

Commande nohup

La commande *nohup* permet à la commande passée en paramètre de continuer son exécution même si 1'utilisateur se déconnecte du système.

Normalement lorsqu'un utilisateur se déconnecte, un signal est envoyé aux différents processus rattachés au terminal, signal entraînant leur arrêt d'exécution. La commande *nohup* permet de se prémunir de ce signal.

Syntaxe :

nohup commandes

Tous les messages non-redirigés, qu'une commande mise en *nohup* est susceptible de produire, sont placés dans un fichier *nohup.out* créé dans le répertoire courant si c'est possible ou bien dans le répertoire personnel de 1'utilisateur.

Commande trap

Cette commande permet de modifier le traitement associé par défaut à un signal. On parle alors de *" détournement d'un signal "*.

Syntaxe :

trap commande signaux

- sans argument la commande *trap* affiche la liste des signaux " détournés ";
- avec argument permet de détourner un signal donné.

Exemple 1 :

```
trap 'echo suppr' 2  # affichage du mot suppr lorsque le signal 2 est
reçu .
trap 2               # restitue le comportement initial du signal 2.
trap " " 1 2 3       # ignore les signaux 1, 2 et 3.
trap : 1 2 3         # reprend en compte les signaux 1, 2 et 3 ignorés
mais ne fait rien.
```

Dans l'exemple précédent de détournement, la modification n'est pas prise immédiatement en compte : il faut dans un premier temps vider le tampon associé au clavier (par la commande *sync*) et réaliser un appel système.

Exemple 2 :

```
trap 'rm /tmp/fich.$$ ; exit 1'   1 2 3     # détruit un fichier
temporaire et provoque la fin
# du processus à la réception des signaux 1, 2, 3.
```

Commande top

Cette commande permet de visualiser l'activité processeur en temps réel. Elle affiche la liste des processus s'exécutant actuellement et la rafraîchit automatiquement.

Syntaxe :

top

Cet outil reprend les mêmes libellés de colonne que ***ps*** .

- La touche [q] permet de quitter ***top***
- Pour obtenir de l'aide sur les différentes touches de commandes, il suffit d'appuyer sur [?] ou [h].

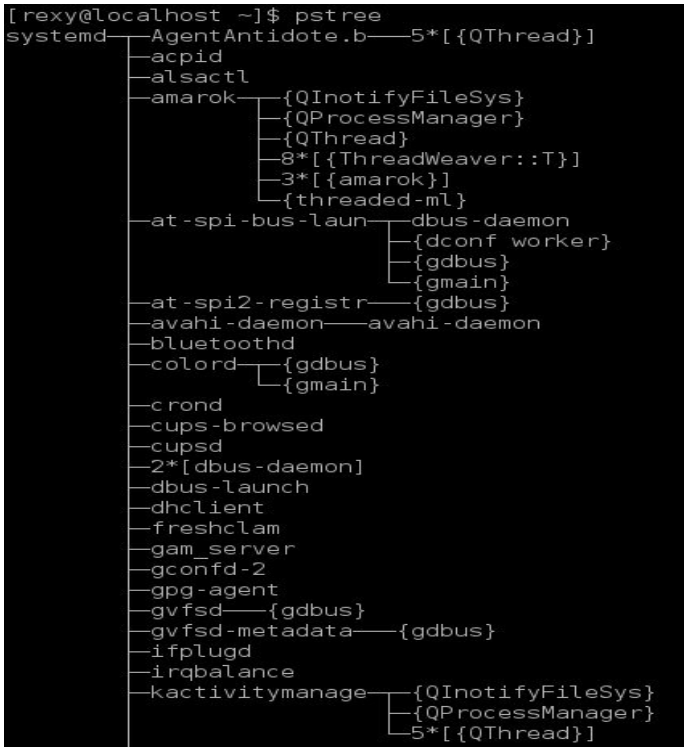
Commande pstree

Cette commande permet d'afficher les noms des processus de façon hiérarchique. Il est alors facile de trouver la filiation de ceux-ci.

Les processus de même niveau et issus de la même commande sont regroupés en une seule ligne.

Syntaxe :

pstree



INTERFACE GRAPHIQUE

KSysguard est le gestionnaire de tâches et le surveillant des performances de ***KDE***

Il présente une architecture **client/serveur** qui permet la surveillance aussi bien d'un **hôte local** que d'un **hôte distant**.

L'interface graphique utilise ce qu'on appelle des **mesures** pour recevoir les informations qu'elle affiche. Une **mesure** peut retourner des valeurs simples ou plus complexes, comme des **tableaux**. Pour chaque type d'information, un ou plusieurs modes d'affichage sont proposés. Les modes d'affichage sont organisés en feuilles de données qui peuvent être enregistrées et chargées indépendamment les unes des autres.

KSysguard n'est donc pas seulement un simple gestionnaire de tâches, c'est aussi **un outil de contrôle** très puissant pour de larges baies de serveurs.

L'interface graphique est disponible sur toutes les plates-formes supportant KDE. Cette interface est pour le moment disponible pour différentes déclinaisons d'UNIX :

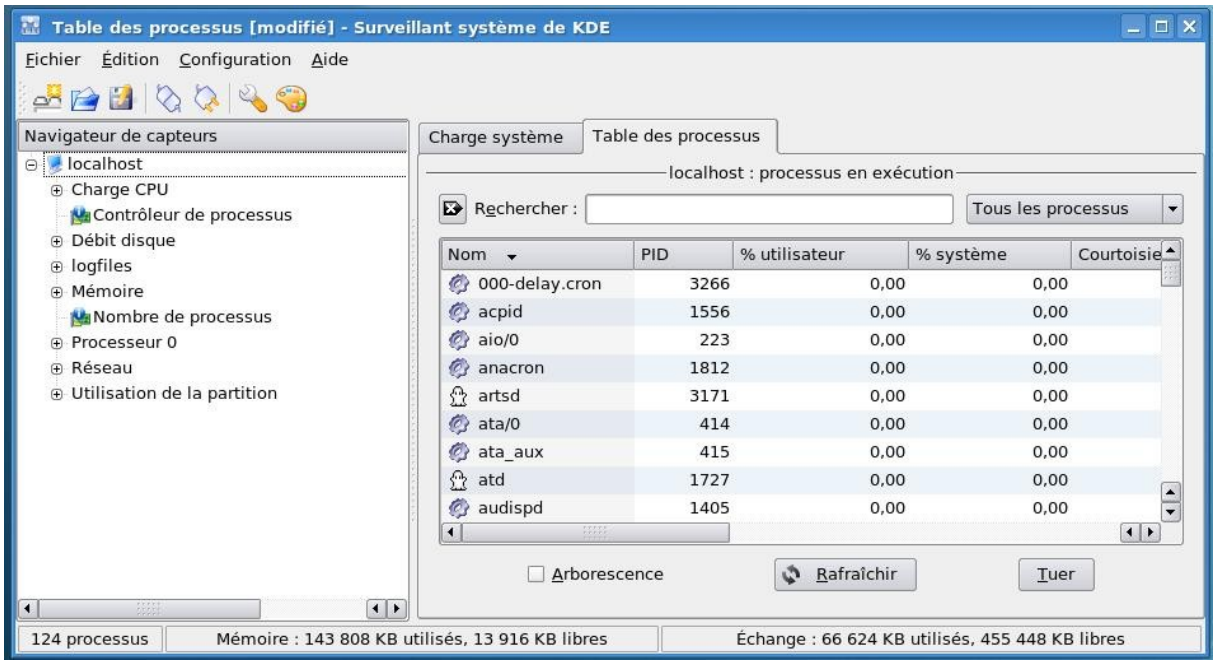


Figure 7.1

Lancement

KSysguard peut être lancé à partir du menu de **démarrage**, en utilisant l'entrée **Surveillance du système** dans le menu **Système.**

En ligne de commande depuis un terminal

Syntaxe :

ksysguard

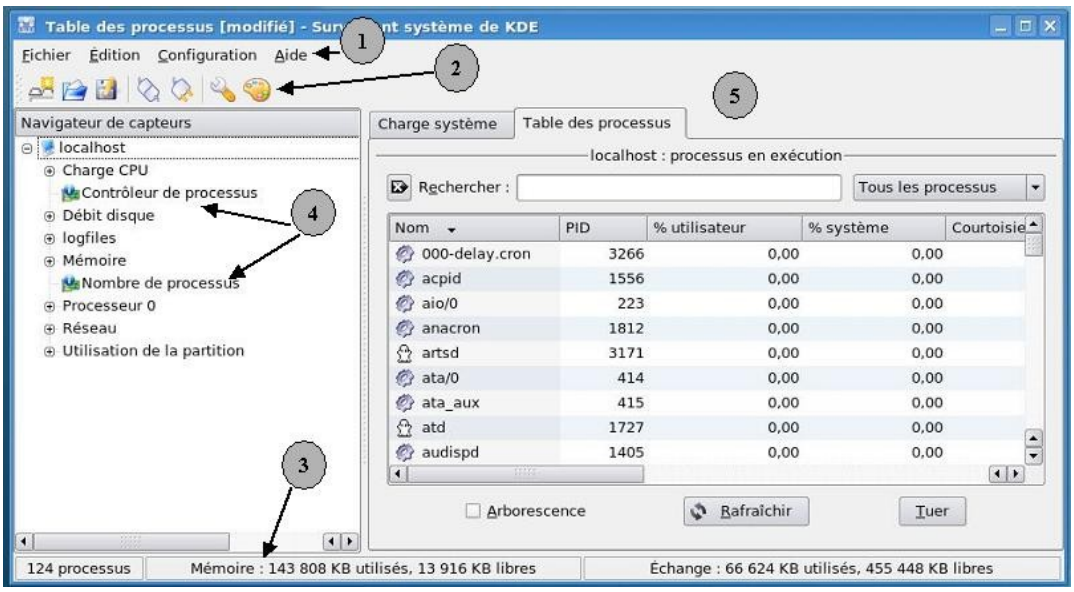


Figure 7.2

La fenêtre principale de ***KSysguard*** se compose d'une barre de menus (1), d'une barre d'outils optionnels (2), d'une barre d'état (3), de la liste des mesures (4) et de l'espace de travail (5).

Pour les experts : ***KSysguard*** est un tout petit programme qui ne dépend que de libc. Il peut ainsi être utilisé sur des machines où KDE n'a pas été installé en totalité, comme par exemple des serveurs. Si vous choisissez l'option de commande personnalisée dans la fenêtre de connexion à un nouvel hôte, vous devrez spécifier la commande complète pour démarrer ***KSysguard***.

L’espace de travail

L'espace de travail est organisé en feuilles de données.

Pour créer une nouvelle feuille de données vous devez sélectionner dans le menu « ***Fichier/Nouvelle feuille de travail*** ».

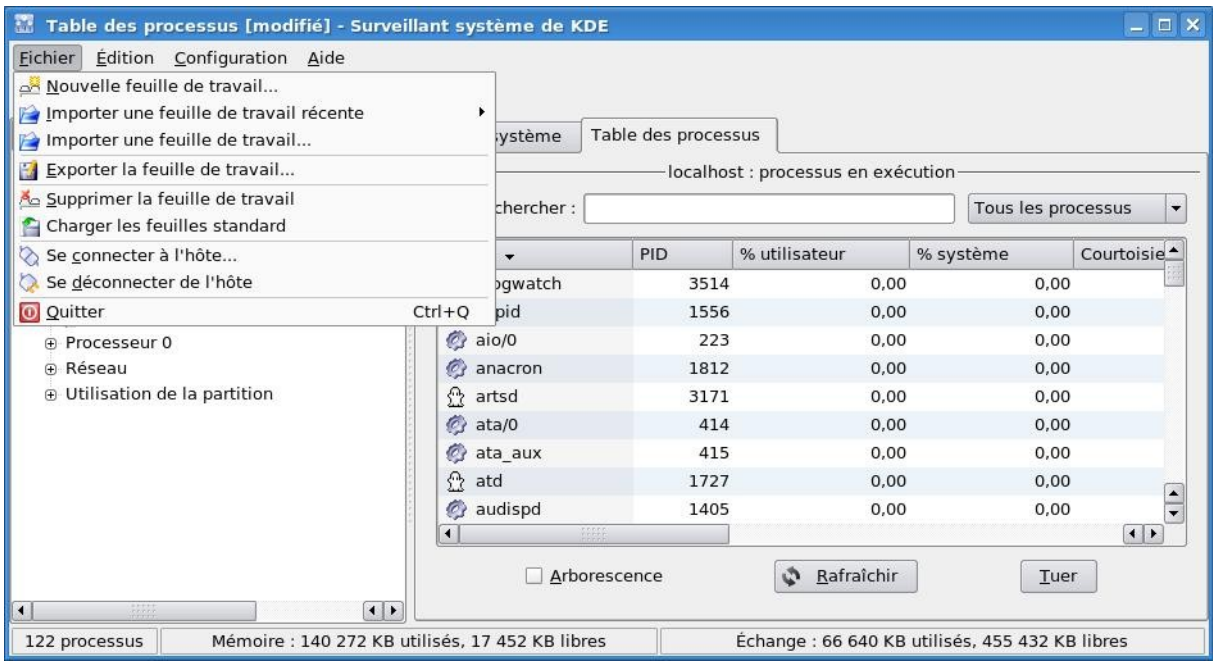


Figure 7.3

Une boîte de dialogue apparaîtra alors dans laquelle vous pourrez indiquer le nom, la dimension et l'intervalle de mise à jour de la feuille de données.

Figure 7.4

La dimension permet de spécifier le nombre de mesures que vous souhaitez avoir par ligne et par colonne. Par exemple dans la feuille de données standard « ***Charge système*** » est composée de deux lignes et colonnes.

Cette nouvelle feuille de données apparaît sous la forme d'un onglet dans l'espace de travail, partie droit de l'applicatif ***KSysguard***. Elles sont composées de cellules présentées sous forme de grille. Chaque cellule peut afficher une ou plusieurs mesures. Vous pouvez remplir une cellule en glissant déposant une mesure à partir de la liste des mesures dans une cellule.

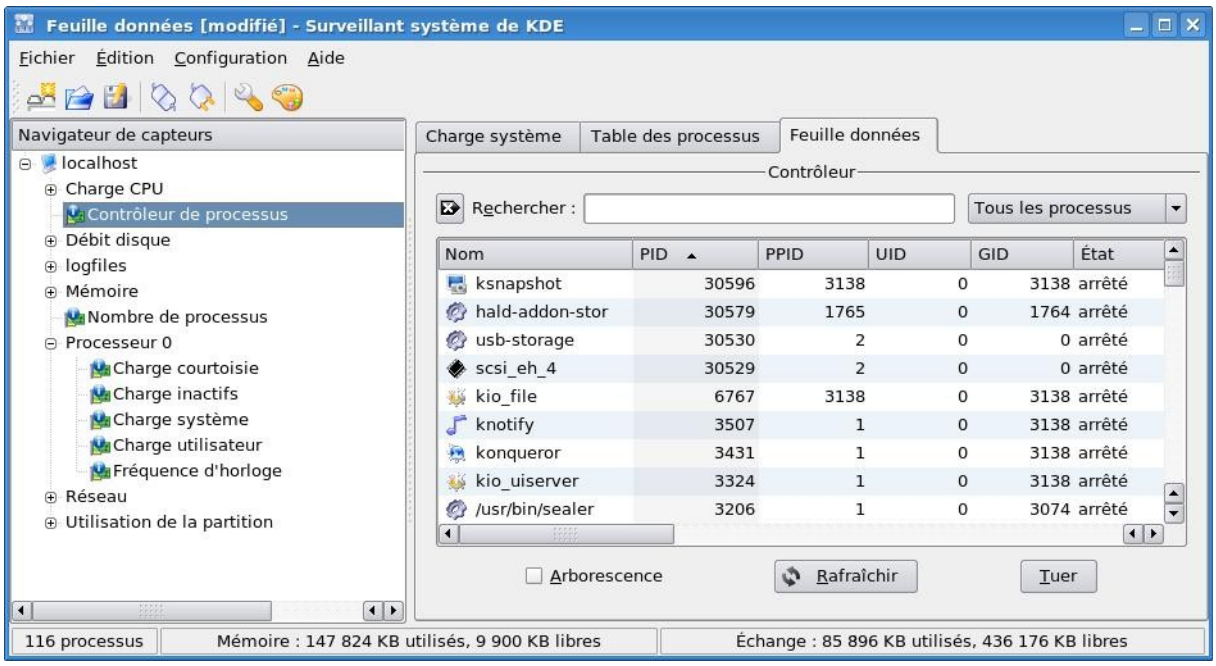


Figure 7.5

Dans les cas où il y a plus d'un mode d'affichage disponible pour le type de mesure sélectionné, un menu surgissant apparaît. Vous pouvez alors sélectionner le mode d'affichage que vous préférez utiliser.

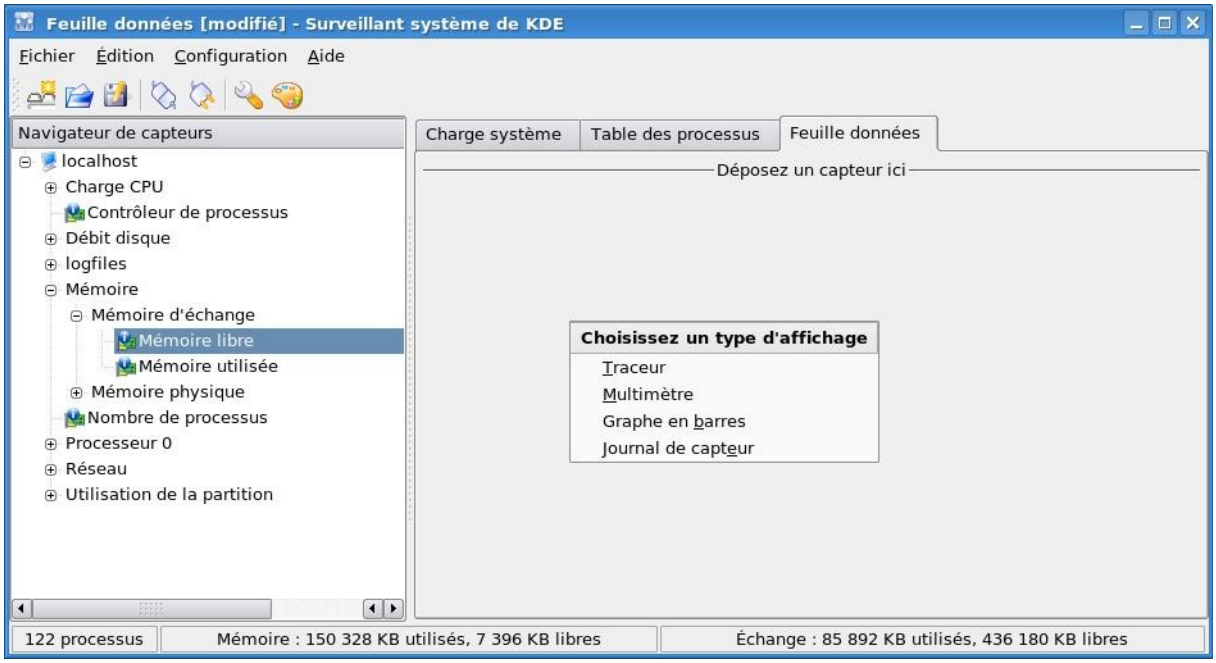


Figure 7.6

Certains modes d'affichage peuvent montrer plus d'une mesure, c'est le cas de l'affichage « ***Journal de capteur*** ». Ajoutez des mesures supplémentaires à un mode d'affichage en glissant déposant ces mesures à partir de la liste des mesures dans l'affichage existant.

Les feuilles de données peuvent être configurées en cliquant dans le menu « ***Édition/Propriétés de la feuille de données*** ». Dans la boîte de dialogue qui apparaît, vous pouvez ajuster la

dimension et l'intervalle de rafraîchissement. Cet intervalle de mise à jour est utilisé par tous les affichages de la feuille de données qui ont l'option « ***intervalle de rafraîchissement*** ».

L'option « ***Configurer le style*** » dans le menu « ***Configuration*** » vous donne la possibilité de configurer les attributs de style globaux, puis de les appliquer aux feuilles de données.

Les affichages peuvent être configurés en cliquant dessus avec le bouton droit de la souris. Un menu contextuel apparaît, dans lequel vous pouvez choisir de modifier les propriétés de l'affichage sélectionné, de le supprimer de la feuille de données, de changer le type et la valeur de son intervalle de mise à jour ou d'arrêter ou recommencer sa mise à jour momentanément.

Pour supprimer cette feuille de données, il vous suffit de sélectionner le menu « ***Fichier/Supprimer la feuille de travail*** ».

Toutes les modifications seront enregistrées dans le fichier de la feuille de données. Si la feuille de données n'a jamais été enregistrée, on vous demandera de lui donner un nom de fichier.

Les différents modes d'affichage

Voici un exemple des différents modes d'affichage que vous pouvez insérer dans une feuille de données.

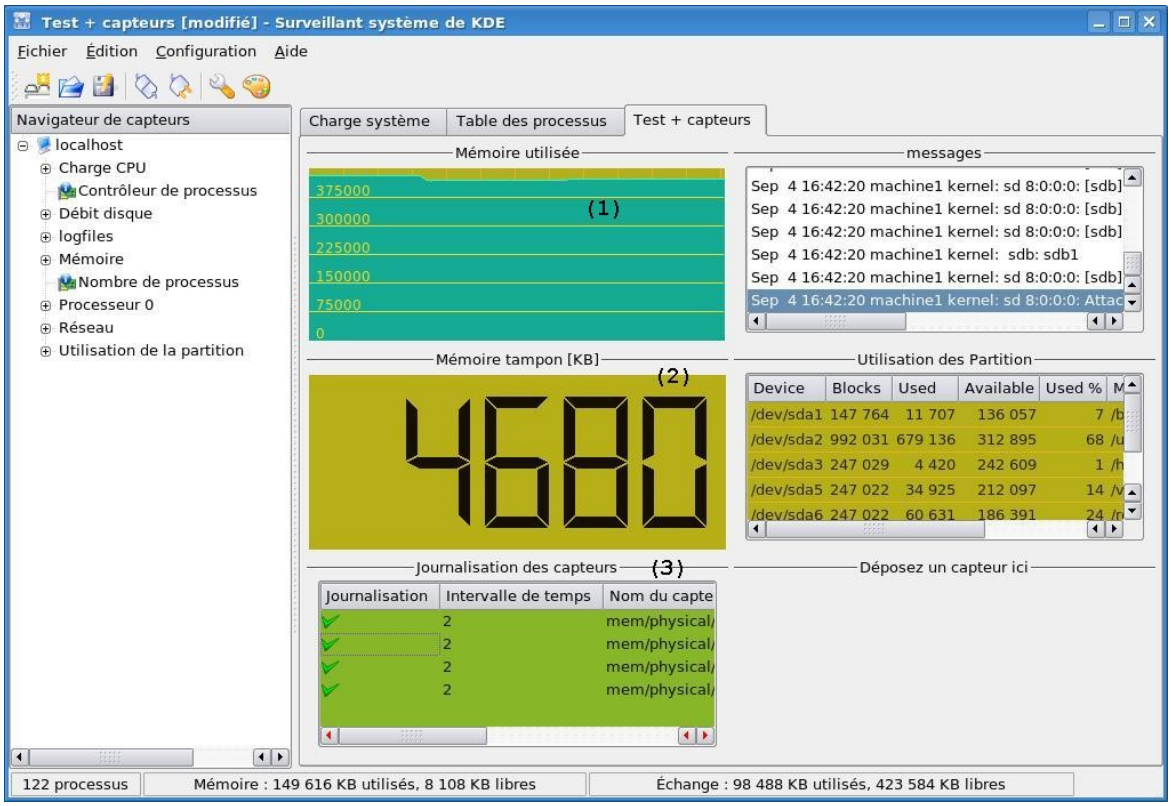


Figure 7.7

Traceur (1)

Le traceur de signaux affiche les niveaux d'une ou plusieurs mesures sur la durée. Dans les cas

où plusieurs mesures sont affichées en même temps, les niveaux sont affichées de différentes couleurs. Si l'affichage est suffisamment grand, une grille sera affichée afin de montrer la valeur des niveaux tracés. Comme, par défaut, la mise à l'échelle automatique est active, les valeurs minimales et maximales seront fixées automatiquement. Parfois, vous voudrez des valeurs minimales et maximales déterminées. Dans ce cas, vous pouvez désactiver la mise à l'échelle automatique et ajuster les valeurs dans la boîte de dialogue des propriétés.

Multimètre (2)

Le multimètre affiche les niveaux de mesure sous forme de multimètre numérique. Dans la boîte de dialogue des propriétés, vous pouvez spécifier les valeurs minimales et maximales. Dans les cas où la limite est dépassée, l'affichage est coloré dans la couleur d'alarme.

Graphe en barres

Le graphe en barres affiche les valeurs des capteurs sous forme d'histogramme. Dans la boîte de dialogue des propriétés, vous pouvez spécifier les limites inférieures et supérieures. Dans les cas où la limite est dépassée, l'affichage est coloré dans la couleur d'alarme.

Journalisation des capteurs (3)

La journalisation des capteurs n'affiche aucune valeur, mais enregistre ces valeurs dans un fichier journal avec des informations supplémentaires à propos de la date et de l'heure. Si vous ne spécifiez pas le chemin de votre fichier journal ce dernier est sauvegardé dans votre dossier personnel.

Pour chaque capteur, vous pouvez spécifier les limites inférieures et supérieures dans la boîte de dialogue des propriétés. Au cas où la limite est dépassée, l'entrée dans la table des capteurs est colorée dans la couleur d'alarme et un événement ***knotify*** est envoyé.

Journal

Le moniteur de journal affiche le contenu d'un fichier, par exemple /var/log/messages. Dans la boîte de dialogue des propriétés, vous pouvez définir une liste d'expressions rationnelles qui seront comparées avec le contenu du fichier. Si l'une de ces expressions correspond, un événement knotify sera envoyé.

Contrôleur de processus

Le contrôleur de processus vous donne une liste des processus de votre système. La liste peut être triée par colonne. Il suffit de cliquer sur l'en-tête de la colonne choisie avec le bouton gauche de la souris.

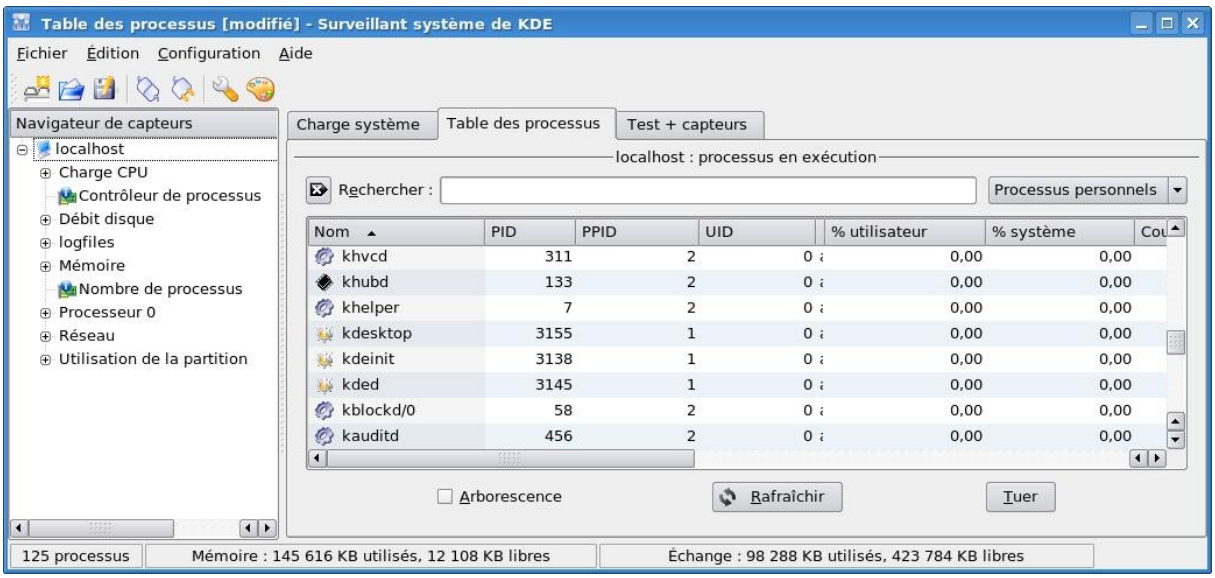


Figure 7.8

La liste affiche les informations suivantes pour tout processus. Veuillez noter que toutes ces propriétés ne sont pas disponibles sur tous les systèmes d'exploitation.

- **Nom :** Le nom de l'exécutable qui a lancé le processus.
- **PID :** L'ID du processus. Un chiffre unique pour chaque processus.
- **PPID :** L'ID du processus parent.
- **UID :** L'ID de l'utilisateur qui a lancé le processus.
- **GID :** L'ID du groupe auquel le processus appartient.
- **État :** L'état du processus.
- **Utilisateur% :** La charge du processeur liée au processus dans l'espace utilisateur (en pourcentage).
- **Système% :** La charge du processeur liée au processus dans l'espace système (en pourcentage).
- **Courtoisie :** L'indice de priorité du processus.
- **VmSize :** L'espace total de mémoire virtuelle (en kilo-octets) utilisé par le processus.
- **VmRss :** L'espace total de mémoire physique (en kilo-octets) utilisé par le processus.
- **Connexion :** Le nom de connexion de l'utilisateur qui a lancé le processus.
- **Commande :** La commande de démarrage complète du processus.

Vous trouvez dans l’espace de travail trois boutons et une option (arborescence).

L’option Arborescence

L'arborescence est destinée à montrer les rapports entre les processus exécutés. Un processus qui est exécuté par un autre processus est appelé l'enfant de ce dernier processus. Une arborescence est une manière élégante de montrer les rapports parent-enfant entre les processus. Le processus *init* est leur ancêtre à tous.

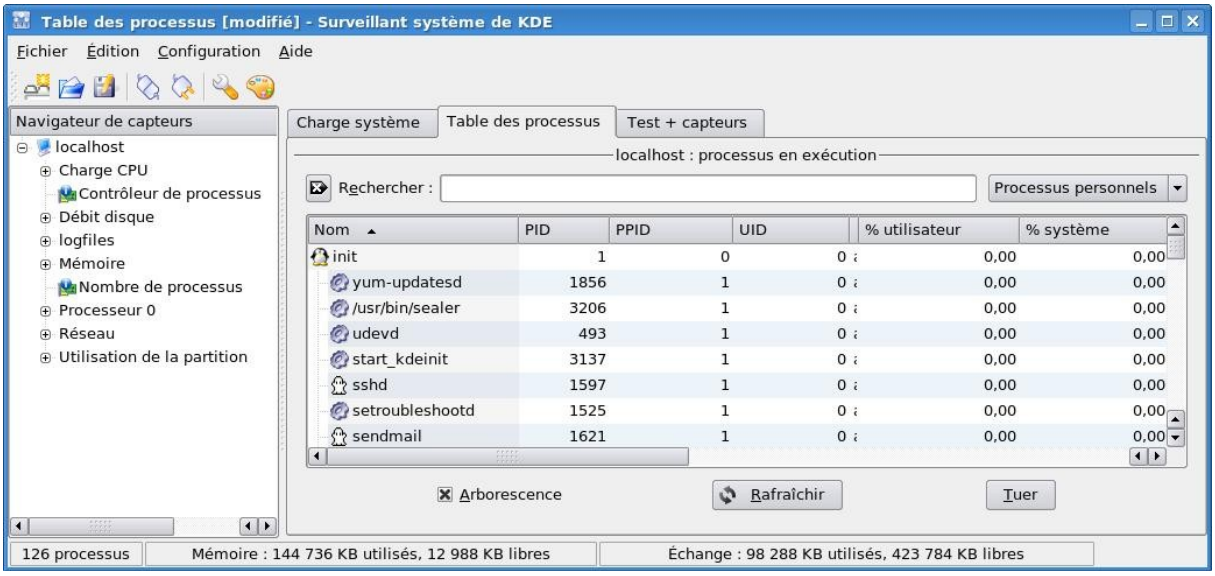


Figure 7.9

Si vous ne vous intéressez pas aux enfants d'un processus particulier, cliquez sur la petite case à gauche du parent et la sous-arborescence disparaîtra. Un autre clic sur cette même case déploiera à nouveau la sous-arborescence.

Le filtre de processus

Le filtre de processus peut être utilisé pour réduire le nombre de processus affichés dans le tableau. Vous pouvez filtrer les processus qui ne vous intéressent pas. Vous pouvez afficher tous les processus, seulement les processus système, les processus utilisateurs ou bien vos propres processus.

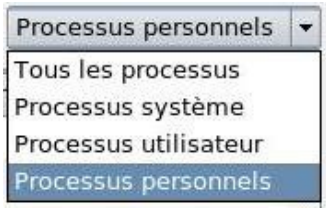


Figure 7.9

Le bouton Rafraîchir

Ce bouton peut être utilisé pour forcer une mise à jour immédiate de la liste des processus.

Le bouton Tuer

Si vous avez sélectionné un ou plusieurs processus, vous pouvez les tuer en appuyant sur le bouton Tuer.

Un signal SIGKILL est envoyé aux processus, ce qui les force à se terminer immédiatement. Si ces applications possèdent des données non enregistrées, ces données sont perdues. Il faut donc utiliser ce bouton avec précaution.

Limites de l'interface graphique

- **On ne peut pas lancer un processus en arrière plan avec l'interface graphique ;**
- **On ne peut pas utiliser les mécanismes de redirection en utilisant l'interface graphique :** redirection entrée < ; redirection sortie >> ; redirection des erreurs 2>> .

EN CONCLUSION : L'étude de la gestion des processus sous UNIX peut se révéler très complexe.

Ce support de cours, s'adresse principalement aux exploitants et permet d'appréhender le sujet sans être exhaustif.

Pour que les utilisateurs avancés ou administrateurs puissent bénéficier de compléments sur le sujet, la consultation d'ouvrages spécifiques est conseillée et Internet est riche de documents très complets et à jour.