Système utilisateur
.:: Système UNIX ::.

3ème semaine – les processus

Les processus

Définition :

Un processus est tout ou partie d'un programme en cours d'exécution.

Exemple de liste de processus :

```
paul      6875  0.0  0.1  4728 1528 ?        Ss   16:22   0:00 ssh-agent /etc/X11/xinit/xinitrc
paul      6900  0.0  1.2 23040 9816 ?        Ss   16:22   0:00 kdeinit: Running...
paul      6941  0.0  1.7 26228 13908 ?       S    16:23   0:00 kdeinit: kpowersave
paul      6946  0.0  0.8 14760 6436 ?        Sl   16:23   0:00 xmms --sm-client-id
paul      6953  0.2  3.2 38628 24880 ?       S    16:23   0:02 kdeinit: konqueror --preload
paul      6954  0.2  3.3 47632 26184 ?       S    16:23   0:02 kopete -session
paul      7134  0.0  2.1 29548 16340 ?       S    16:43   0:00 kdeinit: konsole
paul      7135  0.0  0.2  3404 1828 pts/2    Ss   16:43   0:00 /bin/bash
```

User | Process ID | CPU | Mémoire | Etat | Heure de démarrage | Nom du processus

Les processus

Un programme qui s'exécute utilise le processeur et la mémoire.
Il est composé de deux parties :

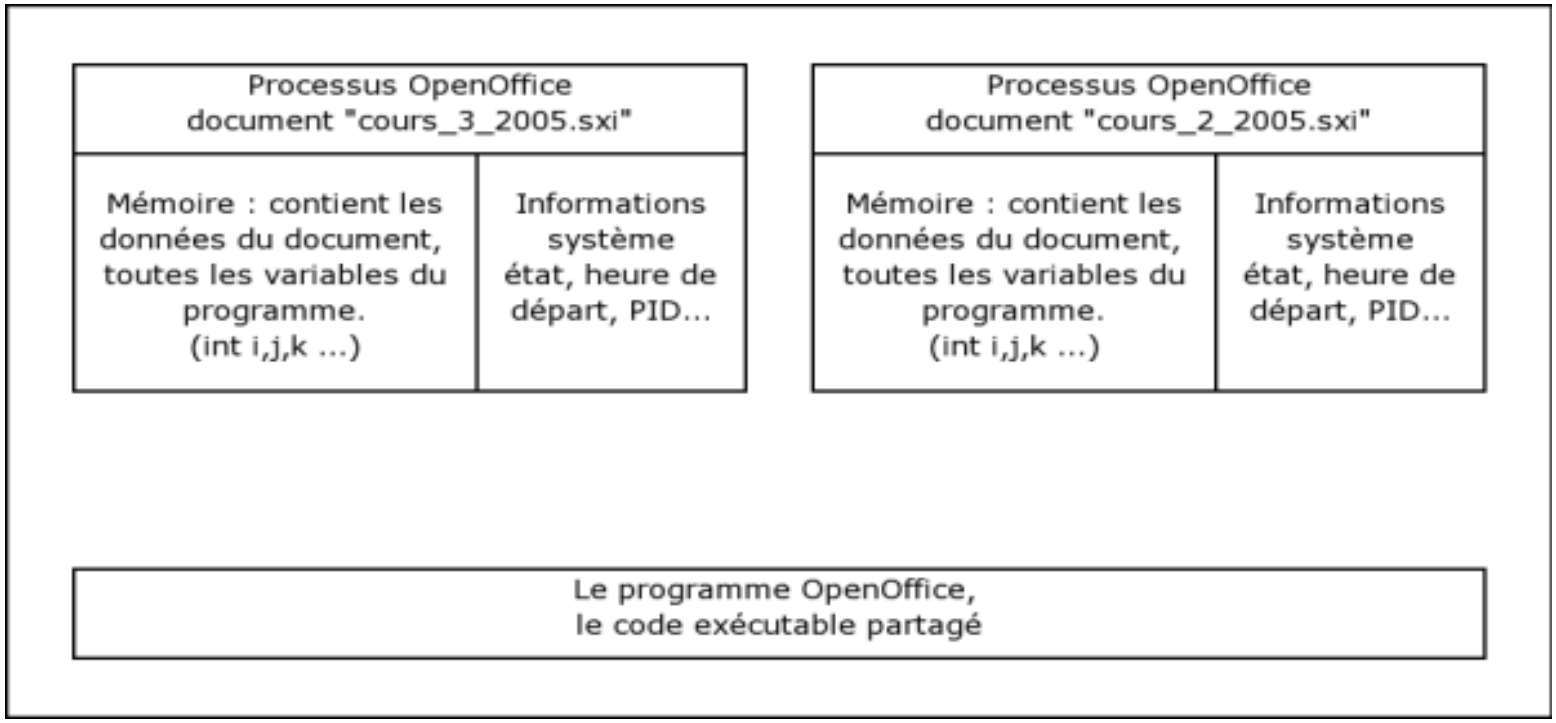
- Un code exécutable

  Ce programme est toujours le même pour toutes les instances lancées en même temps (plusieurs documents Open Office ouverts simultanément par exemple). Il peut donc être éventuellement partagé entre plusieurs processus.

- Un contexte d'exécution

  Ces informations permettent de savoir à quel point de l'exécution du processus nous sommes ;il comprend toutes les données spécifiques au processus.

Les processus



Les processus

Tous les processus ont un père (sauf le premier [init])
Le père d'un processus est celui qui le lance :

Quand est lancée « ls » dans une console, le processus père est la console (bash).

```
init-+-bdflush
     |-cron
     |-dnetc
     |-java
     |-mysqld_safe
     |  |--mysqld
     |-ntpd
     |-snmpd
     |-sshd (649)
     |  `-sshd (17112)
     |      `-bash (17113)
     |         `-pstree (17137)
     |-startpar
     |-syslogd
     `-xinetd
```

Commande *pstree* pour afficher l'arbre des processus

Les processus

Un processus possède :

- Une entrée standard
- Une sortie standard et une sortie erreur
- Un utilisateur propriétaire
- Une priorité
- ...

Toutes ces caractéristiques sont héritées de celles du père.
Donc :

- Les redirections d'E/S d'un processus s'appliquent à ses fils
- La priorité d'un processus aussi..

L'ordonnanceur

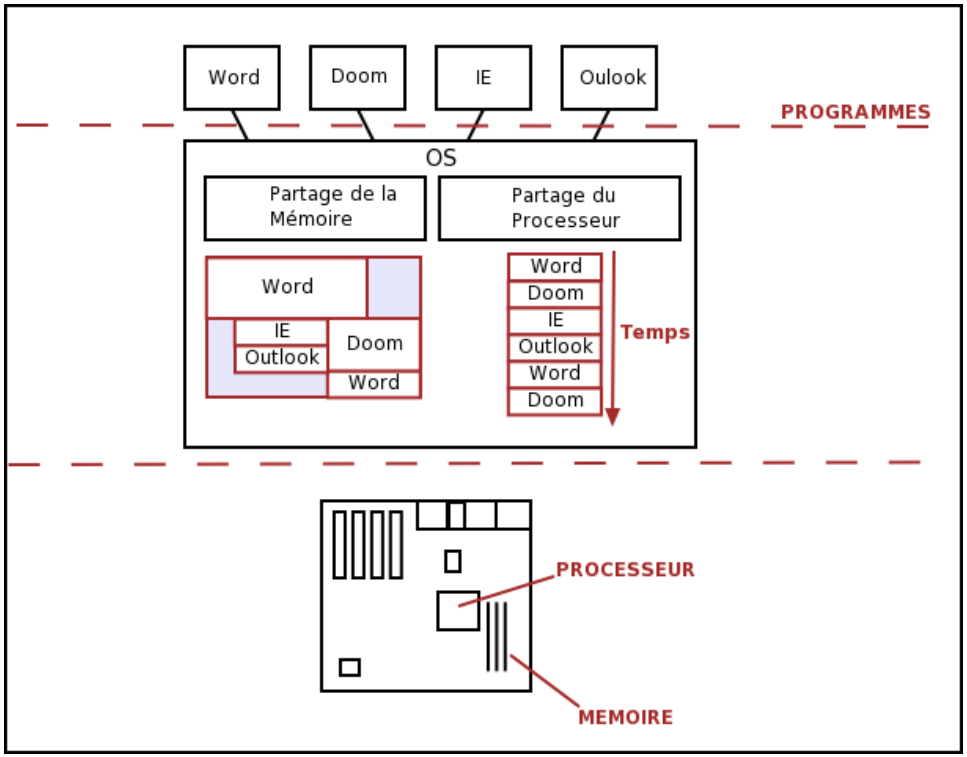
Donner l'impression que plusieurs programmes fonctionnent en même temps

Plusieurs processus fonctionnent en parallèle sur le système et un seul processeur pour les exécuter ...

L'ordonnanceur permet de donner le processeur aux programmes en leur partageant son temps.

=> Un processus est élu
- celui qui est en cours d'exécution

=> Les autres sont bloqués ou mis prêts (mis en attente).



.:: L'ordonnanceur ::.

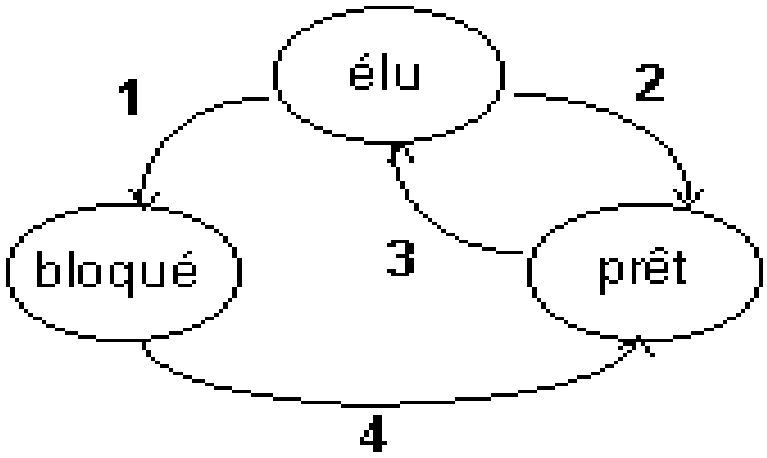
L'ordonnanceur

=> Le processus **élu** est en cours d'exécution durant une période d'ordonnanceur (quelques millisecondes)

=> Le **processus suivant** à être exécuté sera le processus **prêt** dont la priorité est la plus forte.

La priorité tient compte de la « gentillesse » du processus (définie au lancement pas la commande *nice* et du temps d'attente.
A chaque élection la priorité des processus non élus est incrémentée pour qu'ils soient tous, à leur tour, élus.

=> L'ordonnanceur est **préemptif** : il est donc capable d'arrêter un processus en cours d'exécution.
=> Différent des systèmes coopératifs

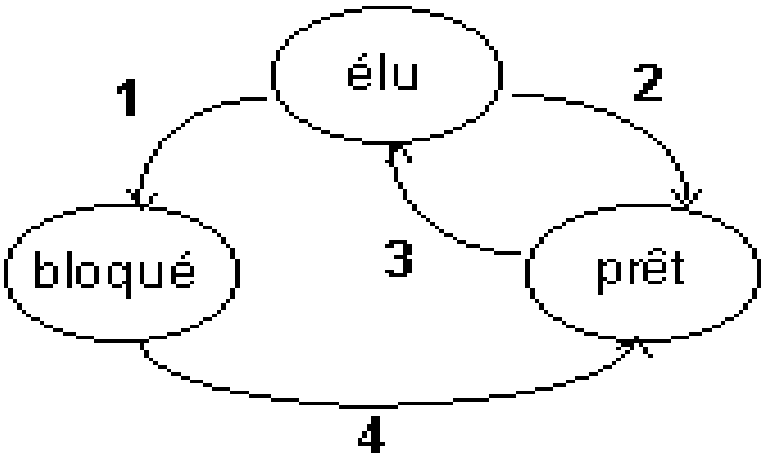


L'ordonnanceur

=> Un **processus bloqué** est un processus en attente de résultat :
Par exemple un processus qui attend une frappe clavier de la part de l'utilisateur ; un processus qui attend une lecture sur le disque dur ...

=> Un processus bloqué ne participe pas à l'élection : il n'a rien à exécuter tant qu'il n'a pas eu ce qu'il attendait.

=> lorsque le processus reçoit les informations attendues (frappe de touche réponse du HD) il est remis en attente et participe aux élections suivantes.



L'ordonnanceur

=> Il existe deux politiques d'ordonnanceurs:

=> **A temps partagé** : donner l'illusion à l'utilisateur qu'il dispose seul du processeur en donnant la main aux programmes de façon équitable.
Contexte bureautique où plusieurs programmes sont utilisés en même temps, sans qu'il n'y ait de contraintes spécifiques.

=> **Temps réel** : assurer des contraintes temporelles, très utile en informatique industrielle (ABS...)
Garantir la prise en compte d'évènements sous certains délais comme par exemple l'usage du frein dans une voiture.

=> L'ordonnancement est une des principales tâches du système d'exploitation

.:: La commande PS ::.

La commande Ps

PS affiche les informations relatives aux processus

-> Options classiques : auxw

-> : -efl

```
paul@linux:~> ps auxw
...
root      1180 99.1  0.1   640  444 ?        RN   Aug29 28861:46 ./dnetc
...

paul@linux:~> ps -elf
...
0 R root     1180     1 99  99  19 -   160 -      Aug29 ?       20-01:03:39 ./dnetc
...
```

Les mêmes informations sont présentées de façon différentes selon plusieurs standards

.:: Retour d'un processus ::.

Le retour d'un processus

Un processus retourne un code qui permet de savoir s'il s'est déroulé correctement.

La valeur 0 est retournée lorsque tout s'est bien passé.

Une valeur différente de zéro indique une erreur, il sera possible d'identifier cette erreur en fonction du numéro retourné.

Le code de retour est renseigné dans la variable d'environnement « ? » ; il peut donc être consulté de la façon suivante :

```
paul@linux:~> ls
...
paul@linux:~> echo $?
0
```

Le retour d'un processus

```
paul@linux:~> ls /toto
/bin/ls: /toto: Aucun fichier ou répertoire de ce type
paul@linux:~> echo $?
1
```

Pour savoir à quelle erreur correspond cette valeur, il faut consulter le manuel ...

.:: Enchainement de processus ::.

Enchaînement de processus

Lancement d'une commande :

paul@linux> ls

Lancement d'une commande en arrière plan :

paul@linux>ls &

Le shell n'attend pas la fin de la commande pour rendre la main.

exemple avec ***find / -name "bash" &***

Lancement de commandes successives :

paul@linux> cd /etc/ **;** ls

Les commandes sont séparées par des « ; »

Permet de lancer cd /etc **puis** ls

Enchaînement de processus

Enchaînement **conditionnés** :

paul@linux> cd /toto && ls
La commande **ls** n'est exécutée que si la commande **cd /toto** termine sans erreur

paul@linux> cd /toto || echo "répertoire inexistant"
La commande **echo** n'est exécutée que si la commande **cd /toto** se termine avec erreur

Lancement en **parallèle** :

Utilisation du mécanisme des tubes.

paul@linux> ls -l /etc | grep passwd

Les deux commandes sont lancées en même temps, elles se synchronisent ensuite grâce aux flux d'entrée et de sortie.

Enchaînement de processus

Regroupement de commandes :

paul@linux> (cd /toto || echo "repertoire inexistant" >&2) > resultat.txt

Les commandes placées entre parenthèses sont exécutées dans un sous processus du shell il est ainsi possible de rediriger toutes les E/S de ce nouveau shell.

Le code de retour d'un enchaînement ou d'un regroupement est celui du dernier processus exécuté

Pour **forcer un code de retour** il est possible d'utiliser la commande *exit n* où *n* prend la valeur à retourner :

paul@linux> (cd /toto 2> /dev/null || (echo "repertoire inexistant" >&2 ; exit 1)) && ls

La commande *ls* n'est exécutée que si le répertoire existe, les sorties sont redirigées ...

.:: Variables shell ::.

Les variables Shell liées aux processus

? - Code de retour du dernier processus terminé
$ - PID du shell courant
! - PID du dernier processus lancé en arrière plan

Affichage de ces informations :

> echo $?
> echo $$
> echo $!

.:: Les signaux ::.

Les signaux

Les signaux permettent la **communication entre les processus**. C'est un message qu'un processus peut envoyer à un autre processus.
Ce mécanisme peut aussi être utilisé par un utilisateur au travers de la commande ***kill***.

Les signaux sont des **messages prédéfinis** pour lequel le processus va avoir une réponse prédéfinie (SIGQUIT pour terminer) ou une réponse spécifique (SIGUSR1,...) libre.

Kill n'est pas une instruction qui tue un programme c'est une **instruction qui envoie un signal quelconque**.

Les signaux

Signaux existants :

--> Le comportement par défaut est indiqué en *italique* ; la plupart peuvent être redéfinis.

SIGHUP : mort du processus de contrôle (fermeture du terminal par exemple). *Termine le programme récepteur.*

SIGINT : Interruption (émis lors d'un CTRL+C) . *Termine le programme.*

SIGQUIT : Demande de quitter (émis lors d'un CTRL+\). T*ermine le programme en créant un "fichier core".*

SIGILL : Instruction illégale. *Termine le programme en créant un fichier core.*

SIGABRT : Terminaison liée à l'exécution de la fonction "abort" (terminaison anormale). *Termine le programme en créant un fichier core.*

Les signaux

SIGFPE : Erreur arithmétique dans un calcul en virgule flottante. *Termine le programme avec un fichier core.*

SIGKILL : Demande au processus de se terminer. *Termine le programme.* **Signal non redéfinissable.**

SIGSEGV : Accès à une zone mémoire invalide (Par exemple lorsque vous dépassez les limites d'un tableau...). *Termine le programme avec création d'un fichier core.*

SIGPIPE : Ecriture dans un tube sans lecteur (Broken pipe). *Termine le processus.*

SIGALRM : Temporisation lancée par la fonction **alarm** écoulée. *Termine le processus.*

SIGTERM : Demande la terminaison du processus. *Termine le processus.*

Les signaux

SIGUSR1 : Signal utilisateur 1. *Termine le processus.*
SIGUSR2 : Signal utilisateur 2. *Termine le processus.*
SIGCHLD : Signal indiquant qu'un processus fils s'est arrêté. *Sans effet*.
SIGSTOP : Arrêt du processus. *Arrête le processus qui prend l'état bloqué au lieu d'élu.* Non redéfinissable.
SIGTSTP : Arrêt du processus demandé depuis le terminal. *Arrête le processus.*
SIGCONT : Continuer un processus arrêté. *Le processus reprend l'état prêt*. Non redéfinissable.

Interception des signaux

Il est possible de redéfinir le comportement de signaux avec la commande *trap*. (Pour ceux pour lesquels c'est autorisé)

Exemple :

```
$ cat naif
#!/bin/bash

trap 'echo "Je suis immortel !"' SIGINT SIGQUIT

i=0
while [ 0 ]; do
   echo $i
   i=$(($i+1))
   sleep 1
done
```

Demo ...

Interception des signaux

Exemple :

```
$ cat alarme
#!/bin/bash

trap 'echo "Le temps est écoulé !"; exit 1' SIGALRM

(/usr/bin/sleep 3; kill -SIGALRM $$ 2> /dev/null )&
echo -n "Vous disposez de 3 s. pour entrez un mot: "
read MOT
trap - SIGALRM; kill -SIGKILL $! 2> /dev/null
echo "le mot lu: $MOT"
```

Demo ...

Suspension d'un processus

La commande ***suspend*** (CTRL z dans le shell) permet d'endormir le processus qui l'exécute. Il sera réveillé par un message SIGCONT.

Suspend nécessite de valider le mécanisme de job control par l'instruction *set -m*.

Quelques commandes :
bg : Permet de placer le dernier processus arrêté en tache de fond
fg : Permet de placer le dernier processus arrêté au premier plan.

1. Quelle commande permet de rechercher une chaine de caractère dans un fichier
a – grep **b - find** **c – cat**

2. Comment aller directement dans le répertoire home de l'utilisateur courant ?
a – cd /home **b – cd ~** **c – cd ..**

3. Comment savoir à qui appartient un fichier ?
a – ls -tr **b - ls -a** **c – ls -l**

4. Quelle commande permet d'extraire une colonne dans une ligne ?
a – cut **b - column** **c - grep**

5. Dans quel répertoire sont les fichiers de configuration du système ?
a – /etc **b – /proc** **c - /var**

6. Linux est issu ?
a – d'un UNIX d'AT&T **b - de XENIX** **c – de rien**

7. Pour rediriger la sortie standard vers la sortie erreur, il faut utiliser :
a - >&2 **b – 2> stderr** **c - >> stderr**

8. Quelle commande permet de modifier les droit d'accès à un fichier ?
a – chmod **b – chown** **c - chrights**

9. Que contient le fichier /etc/passwd ?
a – les mot de passe **b - les droits d'accès** **c – la liste des utilisateurs**

10. Sur un répertoire, le droit d'exécution permet-t-il :
a – d'aller dans le repertoire **b- de créer de nouveaux fichiers** **c- de lister le répertoire**

1. Quelle commande permet de rechercher une chaine de caractère dans un fichier
a - find **b – grep** **c – cat**

2. Comment aller directement dans le repertoire home de l'utilisateur courant ?
a – cd home **b – cd /** **c – cd**

3. Quelle commande permet de connaître l'espace occupée par un répertoire ?
a - df **b - quota** **c - du**

4. Quelle commande permet d'extraire une colonne dans une ligne ?
a – cut **b - column** **c - grep**

5. Dans quel répertoire sont rattachés les périphériques du système (carte son ....) ?
a – /etc **b – /proc** **c - /dev**

6. Linux est issu ?
a – d'un UNIX d'AT&T **b - de XENIX** **c – de rien**

7. pour rediriger la sortie d'erreur d'un programme vers un fichier il faut ajouter à la fin de la commande :
a - >&2 fichier **b – 2> fichier** **c - >> fichier**

8. Quelle commande permet de modifier le propriétaire d'un fichier ?
a – chmod **b – chown** **c - chrights**

9. Quel fichier contient, sur hina, les mots de passe encryptés des utilisateurs ?
a - /etc/passwd **b - /etc/shadow** **c - /etc/group**

10. Sur un répertoire, le droit d'écriture permet-t-il de :
a – écrire dans les fichiers existants **b- créer de nouveaux fichier** **c- lister le répertoire**