2 : Services, processus, signaux

Cours 2 : Services, processus, signaux

Le Cocq Michel
lecocq@ipgp.fr

Licence Pro SIL

13 Janvier 2017

Rappel plan général

- Introduction au système UNIX - ssh - mardi 10.01.2017
- Services, processus, signaux - jeudi 13.01.2017
- Scripting shell
- Applications Client/Serveur et Web
- Serveurs et configuration

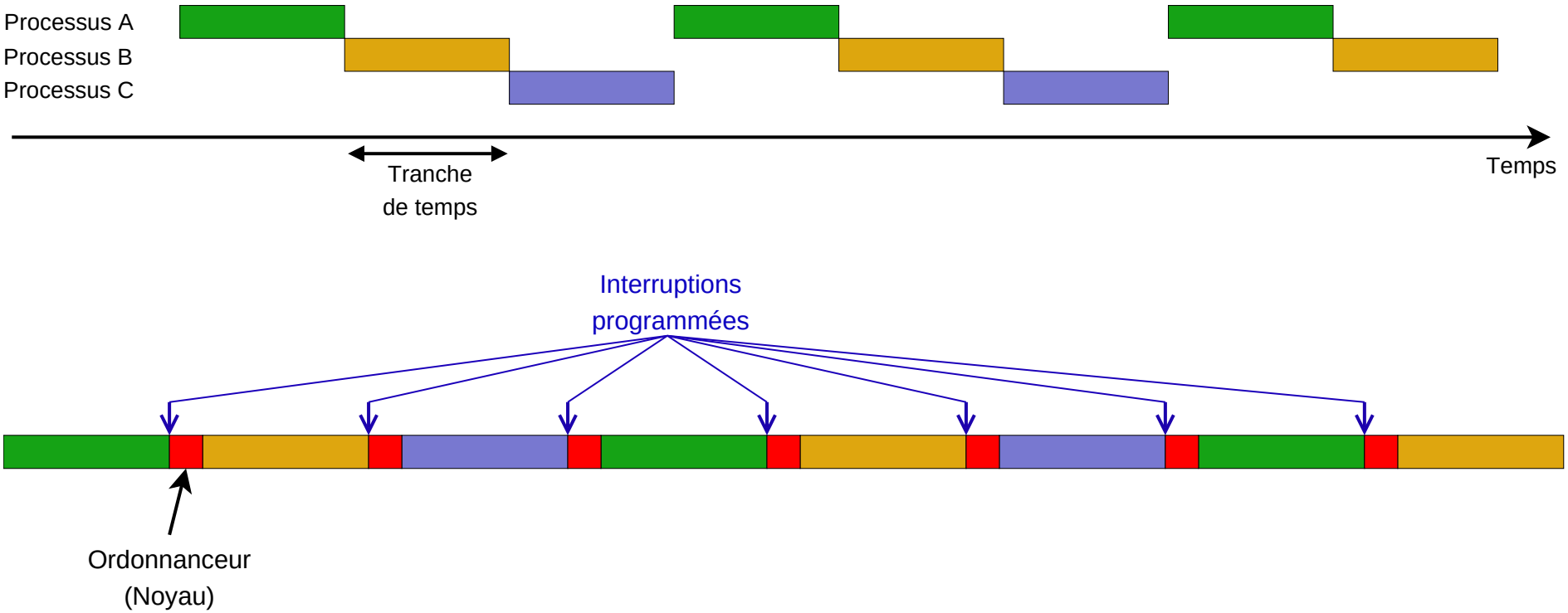
Plan du cours 2 : Services, processus, signaux

Définition

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

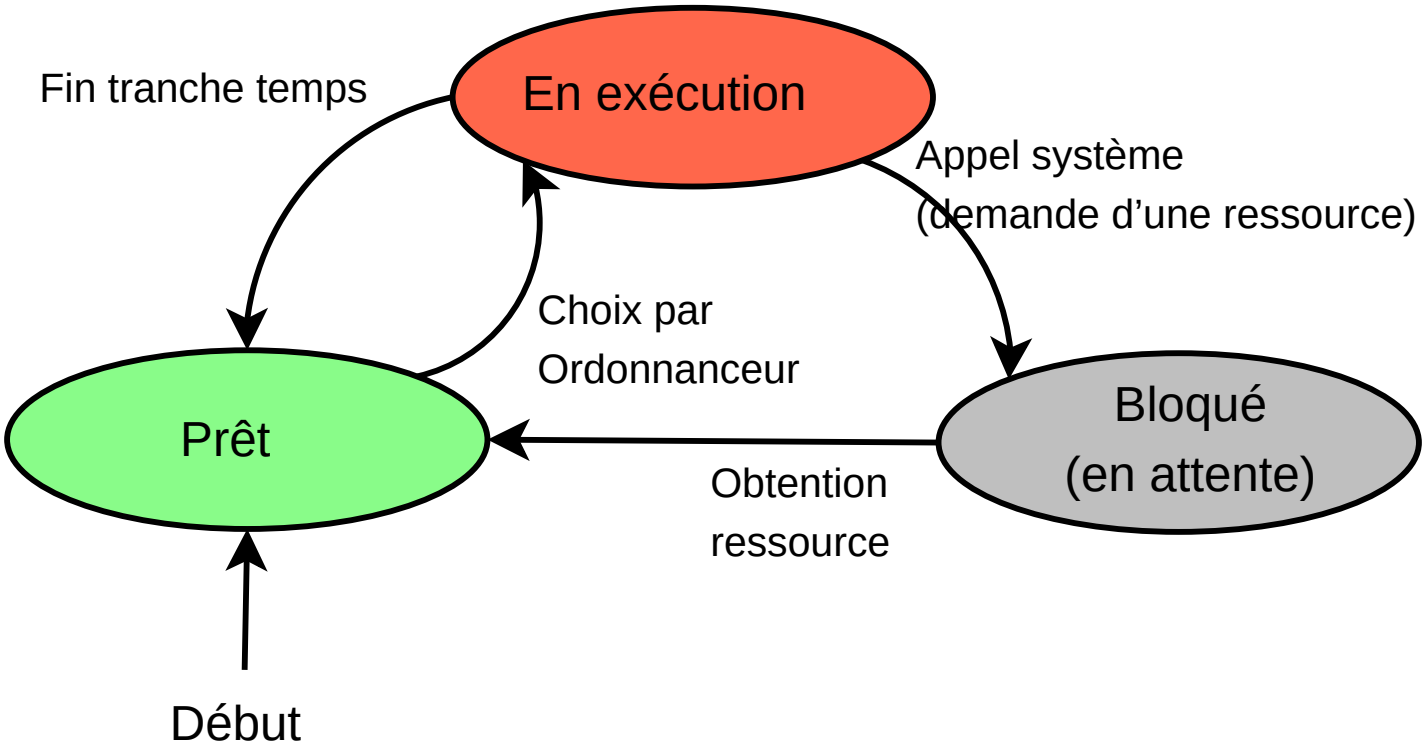
Partage du temps entre processus

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

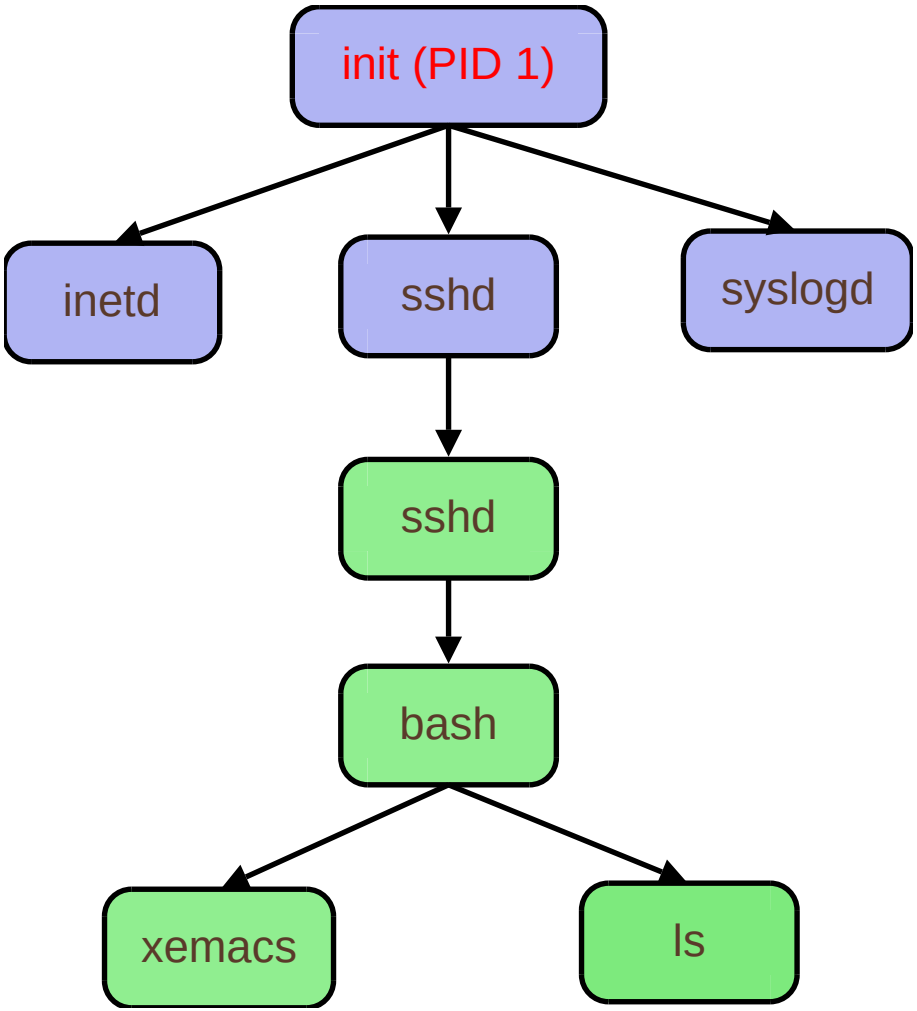
Quelques attributs importants :

- 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.

Examples

`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**.

Examples

```
$ 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*:

Examples

```
$ 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.

Examples

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

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

Examples

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

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

Examples

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
```

Le retour des dæmons !

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.