UNIVERSITÉ DE CAEN **9 octobre 2024**

IUT Grand Ouest Normandie **Dép. Info**

R104 — Introduction aux systèmes d'exploitaion et à leur fonctionnement
TD 5 : Gestion de processus – première approche — 2h

d'après le TD de Jean-François Anne

Objectif(s) :
— Commandes `echo`, `exec`, `.`, création de processus

1 Mise en route

Exercice 1.1. L'éxecution de processus En étudiant le cours et le `man`, répondez aux questions ci-dessous. Lancez ensuite les commandes et vérifiez vos intuitions.

1. Que fait la commande `tty` ?

2. Que fait la commande `ps` ? Décrivez chaque colonne de sa sortie.

3. Que fait la commande `echo $$` ?

4. Que fait la commande `ps -aux` ? Décrivez chaque colonne de sa sortie.

5. Que signifie `TTY = ?` dans la sortie de la commande `ps -aux` ?

6. Proposez une commande qui liste uniquement les processus lancés par « root » ?
7. Observez la session suivante :

```
prompt> echo $$
264536
prompt> bash # (lancement d'un nouveau shell = création d'un nouveau process)
prompt> ps
PID TTY          TIME CMD
264536 pts/5    00:00:00 bash
264576 pts/5    00:00:00 bash
264599 pts/5    00:00:00 ps
prompt> echo $$
264576
prompt>
```

   On rappelle qu'un processus est créé lorsque l'on exécute un programme. Précisez quel processus est parent de quel processus. L'option `--forest` de la commande `ps` affiche l'arborescence des processus. Lancez la commande `ps` avec ces options et vérifiez votre intuition.

Exercice 1.2. Enchaînement des processus

1. Quel est le caractère utilisé pour enchaîner plusieurs commandes sur la même ligne ?

2. Regarderez dans le manuel le rôle de la commande `sleep`. Sans lancer la commande si-dessous, intuitez ce que fait la ligne suivante :
   ```
   echo début ; sleep 5 ; echo milieu ; sleep 5 ; echo fin
   ```
   Vérifiez votre intuition en lançant la ligne de commande.
3. En utilisant les commandes `true` et `false`, qui renvoient respectivement un code de retour qui indique une réussite et un échec, montrer que lors de l'exécution successive (en utilisant l'opérateur « ; ») de deux commandes le résultat de la première n'influence pas celui de la deuxième. Pour cela, lancez les commandes suivantes :

```
prompt> echo debut ; false ; echo fin
…
prompt> echo debut ; true ; echo fin
…
```

> **Remarque**
>
> Ne pas confondre avec :
> — Le pipe `com1 | com2` où l'on redirige la sortie de `com1` vers l'entrée de `com2`
> — La redirection `com1 > fichier.data` où l'on redirige la sortie de `com1` vers le fichier `fichier.data`.

2 Enfantement des processus

2.1 Notions préliminaires

Dans une première partie, on va travailler avec les notions de patch et de PATH dont nous allons nous resservir dans la suite.

Exercice 2.1. Gestion des diff via des patchs

1. Téléchargez l'archive sur la page de cours. Décompressez là. Elle contient un dossier `diff-test` dans lequel sont placés trois fichiers. Lancez la commande `diff toto toto1`.
2. Analysez la sortie de la commande `diff`. Cela vous a été mentionné en CM (cour n°4), mais utilisez le `man` si vous avez besoin de plus d'informations. La commande `diff fich1 fich2` décrit les différences entre deux fichiers `fich1`, `fich2` à la manière d'actions à exécuter sur `fich1` pour le transformer en `fich2`. Cette liste d'action s'appelle un *patch*.
3. Utilisez le `man` de la commande `patch` pour comprendre comment appliquer un patch.
4. Appliquer le patch `toto.patch` au fichier `toto` pour en faire un fichier `toto2`.
5. Si l'application du patch s'est bien passée, les fichiers `toto1` et `toto2` devraient bien entendu être identiques. Vérifiez cela au moyen de la commande `cmp`.

Exercice 2.2. La notion de PATH

On l'a vu dans le cours, en UNIX, tout est fichier, même les commandes (telles que `ls`) qu'on lance dans le terminal sont en fait des éxécutables qui sont lancés lors de l'appel de la commande. Dans cet exercice, on creuse un peu cette notion d'exécutables, et de comment ils sont trouvés par le shell.

1. Utilisez la commande `which` pour déterminer la localisation de l'exécutable de la commande `ls`.
2. Allez dans le dossier en question, faites un `ls -l` sur le fichier exécutable de `ls`. Quels sont les droits sur ce fichier ?

3. Utilisez la commande `env` pour lister les variables d'environnement de votre session shell actuelle.
4. Repérez la variable `PATH` (vous pouvez vous aider d'un filtre via `grep`). Il s'agit d'une liste. Quel est le caractère utilisé pour séparer les éléments de la liste ? Que représente chaque élément de la liste ? En quelle position se trouve le chemin vers l'exécutable `ls`.
5. Dans votre `home`, créez un fichier exécutable nommé `ls`, et qui contient la commande `echo toto`.
6. Exécutez ce script en utilisant `./ls`.
7. Ajoutez votre `home`, à la fin de la liste des chemins du PATH : `PATH=$PATH:/home/username` (Faites bien attention à changer le username par le vôtre…).
8. Lancez la commande `ls`. Quel est l'exécutable qui a été lancé ? Que concluez-vous quant à l'ordre dans lequel les différents chemins du `PATH` sont fouillés pour trouver la commande demandée ? Vérifiez votre intuition en changeant l'ordre des chemins dans la variable `PATH`, jusqu'à ce que ce soit *votre* script qui s'éxecute lorsque vous tapez `ls`.
9. Quittez votre terminal, rouvrez-en un autre, afficher la valeur de la variable `PATH`. Que constatez-vous quant à la persistence de vos modifications sur cette variable ?

Remarque

`PATH` est une *variable d'environnement*. Elle est réinitialisée à chaque ouverture de shell. Pour que les modifications du `PATH` soit persistantes, il faut modifier sa déclaration dans un des fichiers utilisés pour configurer son shell à son lancement (le fichier `.bashrc`, ou le `.profile` dans son home, par exemple).

3 Les différents types de lancement

Exercice 3.1.

En utilisant le langage de commande `bash` et les commandes `echo`, `exec` et `.`, nous allons voir la gestion des processus, la création de processus fils et l'exécution d'une commande dans un processus.

Pour chacun des cas étudiés ci-après, prenez note des éléments suivants :
- Par quelle commande le premier script a-t-il été lancé ?
- Quelles étaient les droits nécessaires pour l'exécution de la commande ?
- Est-il nécessaire de mettre le chemin du script dans le `PATH` ?

Cas 1 : lancer un script avec la commande `./nomduscript`

1. Dans l'archive associée à ce TD, le dossier `child-proc/script` contient les fichiers `fichier1`, `fichier2` et `fichier3`.
2. Affichez ces trois fichiers (commande `cat`) et étudiez leur contenu.

3. Essayer de lancer la commande `./fichier1`. Que se passe-t-il ? Que devez-vous faire pour que l'exécution puisse être lancée ?

4. Une fois ce problème résolu (et les suivants...), lancez la commande `./fichier1` en redirigeant les traces dans le fichier `trace_script`.

Cas 2 : lancer un script après l'avoir transformé en commande

1. Copiez le dossier `script` dans le dossier `cmd`, puis utilisez le patch nommé `cmd-u.patch` pour modifier les fichiers : `patch -p0 < cmd-u.patch`

   > **Remarque**
   >
   > Prenez le temps d'observer la structure du patch. Il s'agit d'un format différent de celui du premier exo : le format dit « unifié ». De plus, notez que seul les fichiers `cmd/fichier1` et `cmd/fichier2` sont modifiés du fait que ce sont effectivement les seuls à être changés d'après le patch.

2. Ajoutez votre dossier de travail à votre PATH et lancez la commande `fichier1` en redirigeant les traces dans le fichier `trace_commande`.

Cas 3 : Utilisation de la commande `.`

1. Créez une copie du dossier `cmd` appelée `point`. Éditez chaque fichier de manière à remplacer les appels aux commandes `fichier?` par `. fichier?`.

2. Lancez la commande `. fichier1` en redirigeant les traces dans le fichier `trace_point`.

Cas 4 : Utilisation de la commande `source`

1. Créez une copie du dossier `cmd` appelée `point`. Éditez chaque fichier de manière à remplacer les appels aux commandes `fichier?` par `source fichier?`.

2. Lancez la nouvelle commande `source fichier1` en redirigeant les traces dans le fichier `trace_source`.

Cas 6 : Utilisation de la commande `bash`

1. Créez une copie du dossier `cmd` appelée `bash`. Éditez chaque fichier de manière à remplacer les appels aux commandes `fichier?` par `bash fichier?`.

2. Lancez la nouvelle commande `bash fichier1` en redirigeant les traces dans le fichier `trace_bash`.

3.1 Cas 6 : Utilisation de la commande `exec`

1. Créez une copie du dossier `cmd` appelée `exec`. Éditez chaque fichier de manière à remplacer les appels aux commandes `fichier?` par `exec fichier?`.

2. Lancez la nouvelle commande `exec fichier1` en redirigeant les traces dans le fichier `trace_exec`.

3.2 Conclusions

1. Pour chacun des cas étudié :
   — Est-il nécessaire que le répertoire de travail soit dans le PATH ?
   — Est-il nécessaire d'avoir le droit d'écriture sur les fichiers pour pouvoir les lancer ?
   — Est-il nécessaire d'avoir le droit de lecture sur les fichiers pour pouvoir les lancer ?
   — Est-il nécessaire d'avoir le droit d'éxecution sur les fichiers pour pouvoir les lancer ?
2. À partir des fichiers de traces générés, réalisez les diagrammes de traces, tels que ceux présentés en cours : une ligne verticale pour représenter un processus légendé par la commande associé et leur pid, et relié entre eux au moyen de flèches pour indiquer les liens de parenté.
3. À partir de vos diagrammes, expliquez les différences entre l'appel d'une commande `./commande`, `commande`, `source commande`, `bash commande`, `. commande`, et `exec commande`. Quelles sont les commandes qui créent un nouveau processus et celles qui n'en créent pas ?
4. Nous avons étudié plusieurs méthodes pour appeler des scripts. Essayez de les appliquer à des binaires, tel que `ls`. Est-ce que ça fonctionne ?

Remarque

Le jargon
Quand on utilise la syntaxe `./fichier` ou fichier, on dit qu'on *éxecute* le fichier. Quand on utilise la syntaxe `source fichier` ou . fichier, on dit qu'on le *source*. L'utilisation du point est la syntaxe "officielle" (norme POSIX). Elle est donc censée marcher dans tous les shells POSIX-compatibles (dont bash, zsh, …). Avoir un script respectant la norme POSIX, c'est s'assurer de sa portabilité. Le shell `bash` définit quant à lui `source` comme un *alias* de `.`.

4 Premier plan et arrière plan

Exercice 4.1.

1. Lancez la commande `date`. Le terminal nous rend-t-il la main ?
2. Lancez la commande `xterm`. Le terminal nous rend-t-il la main ? Qu'attend-t-il ?

3. Dans `xterm`, affichez l'arborescence des processsus liés au terminal (commande `ps l --forest`).
4. Fermez le xterm (via la commande `exit` ou via un `Ctrl+D` dans la fenêtre créée).
5. Par défaut, les commande dans le terminal sont lancées en premier plan. Le terminal attend la fin de l'exécution de la commande pour nous redonner la main. On peut forcer le terminal à nous redonner la main en apposant le caractère `&` à la fin de la commande. Essayer `xterm &`. Le terminal vous rend-il la main ? Expliquez les informations qui se sont affichées au lancement dans la commande. Vérifiez grace à `ps l --forest` dans le terminal initial, puis dans le xterm lui-même.

6. Comparez le résultat de chacune des deux lignes de commandes suivantes. Expliquez ce qui se passe dans le second cas, lorsque l'on ferme le `xterm` ?

   ```
   prompt> xterm & ps l --forest
   ```

   avec

   ```
   prompt> xterm ; ps l --forest
   ```

5 Interruption d'un processus

Comme vu en cours, les processus peuvent communiquer via des signaux. Il nous est possible d'envoyer des signaux aux processus. En particulier des signaux permettant d'interrompre ces processus. Les processus peuvent réagir différemment à ces signaux.

5.1 Envoyer des signaux localement

Dans les cours et TD précédent, nous avons déjà mentionné les séquences de touches Ctrl C et Ctrl D.

Reproduisez les sessions ci-dessous et essayer d'expliquer ce qu'il se passe :

```
prompt> bash
prompt> echo $$
...
prompt> (Ctrl C)
prompt> echo $$
...
prompt>
```

```
prompt> bash
prompt> echo $$
...
prompt> (Ctrl D)
prompt> echo $$
...
prompt>
```

```
prompt> cat > trace
Du texte (Ctrl D)
prompt> cat trace
...
prompt>
```

```
prompt> cat > trace
Du texte (Ctrl C)
prompt> cat trace
prompt>
```

```
prompt> xterm
(CTRL C dans le xterm)
```

```
prompt> xterm
(CTRL C dans le terminal initial)
```

```
prompt> xterm
(CTRL Z dans le terminal initial)
(le xterm est-il utilisable ?)
```

```
prompt> xterm &
(CTRL D)
```

```
prompt> xterm &
(Ctrl D dans le xterm)
```

Les raccourcis clavier utilisés par votre terminal sont configurables via la commande `stty`. Utilisez `stty -a` pour voir votre configuration actuelle et vérifier vos intuitions.

5.2 Envoyez des signaux « à distance »via la commande `kill`

1. Ouvrez un terminal
2. Ouvrez-y un `xterm` en arrière plan et un `xterm` en premier plan
3. Endormez le `xterm` en premier plan (Ctrl-Z)
4. Utilisez la commande `jobs` pour vérifier que les deux `xterm` sont là. L'un tourne et l'autre est endormi.
5. Récupérez le PID du terminal endormi de au moins 2 manières différentes.
6. Affichez la liste des signaux qu'il est possible d'envoyer avec la commande `kill`
7. Trouvez dans le `man` de la commande `kill` quel est le signal envoyé par défaut ?

   ```
   prompt> kill PIDxtermpremierplan
   prompt> kill PIDxtermpendormi
   ```

   Ces kill vont t'ils fonctionner ?

Remarque

N'hésitez pas à tester d'autres signaux.

6 Exécutables vs commandes intégrées

Comme nous l'avons vu au cours de ce TD, par défaut, lorsqu'on éxécute une commande, un processus fils est créé et exécute la commande. Cela est vrai pour la grande majorité de commande, mais pour des raison techniques et de performances, certaines commandes ne sont pas des éxécutables qui vont être lancés dans un processus fils, mais des commandes directement intégrées au shell. On appelle ses dernières des *directives intégrées* (« built-in »).

La commande `type` permet de déterminer si une commande est de type *éxécutable* ou de type *directive intégrée*.

1. Utilisez la commande `type` sur les commandes `ls` et `cd` afin de déterminer leur catégorie respective.
2. Utilisez la commande `type` sur au moins 5 autres commandes que vous connaissez afin de les classer dans ces deux catégories.
3. Utilisez la commande `help`. Les commandes que vous avez catégorisées dans les "built-in" y-sont-elles listées ?