W4-S1-C4-fichiers-systeme

December 14, 2014

1 Fichiers systèmes

1.1 Complément - niveau avancé

Dans ce complément, nous allons voir comment un programme python interagit avec ce qu'il est convenu d'appeler le système d'entrées-sorties standard du système d'exploitation.

1.1.1 Introduction

Dans un ordinateur, le système d'exploitation (Windows, Linux, ou MacOS) est un logiciel (*kernel*) qui a l'exclusivité pour interagir physiquement avec le matériel (CPU, mémoire, disques, périphériques, etc.); il offre aux programmes utilisateur (*userspace*) des abstractions pour interagir avec ce matériel.

La notion de fichier, telle qu'on l'a vue dans la vidéo, correspond à une de ces abstractions; elle repose principalement sur les 4 opérations élémentaires * open * close * read * write

Parmi les autres conventions d'interaction entre le système (pour être précis: le *shell*) et une application, il y a les notions de * entrée standard (*standard input*, en abrégé `stdin`) * sortie standard (*standard output*, en abrégé `stdout`) * erreur standard (*standard error*, en abrégé `stderr`)

Ceci est principalement pertinent dans le contexte d'un terminal. L'idée c'est qu'on a envie de pouvoir *rediriger* les entrées-sorties d'un programme sans avoir à le modifier. De la sorte, on peut également *chaîner* des traitements à l'aide de *pipes*, sans avoir besoin de sauver les résultats intermédiaires sur disque.

Ainsi par exemple lorsqu'on écrit

```
$ monprogramme < fichier_entree > fichier_sortie
```

les deux fichiers en question sont ouverts par le *shell*, et passés à `monprogramme` - que celui-ci soit écrit en C, en python ou en Java - sous la forme des fichiers `stdin` et `stdout` respectivement, et donc **déjà ouverts**.

1.1.2 Le module `sys`

L'interpréteur python vous expose ces trois fichiers sous la forme d'attributs du module `sys`

```
In []: import sys
       for channel in (sys.stdin, sys.stdout, sys.stderr):
           print channel
```

Dans le contexte du notebook vous pouvez constater que les deux flux de sortie sont implémentés comme des classes spécifiques à IPython. Si vous exécutez ce code localement dans votre ordinateur vous allez sans doute obtenir quelque chose comme

```
<open file '<stdin>', mode 'r' at 0x10c3b40c0>
<open file '<stdout>', mode 'w' at 0x10c3b4150>
<open file '<stderr>', mode 'w' at 0x10c3b41e0>
```

On n'a pas extrêmement souvent besoin d'utiliser ces variables en règle générale, mais elles peuvent s'avérer utiles dans des contextes spécifiques. Par exemple, l'instruction `print` écrit dans `sys.stdout` (c'est-à-dire la sortie standard). Mais comme `sys.stdout` est une variable (plus exactement `stdout` est une variable dans le module `sys`) et qu'elle référence un objet fichier, on peut lui faire référencer un autre objet fichier et ainsi rediriger depuis notre programme tous les sorties sur le terminal vers un fichier

```
In []: import sys
       autre_stdout = open('ma_sortie.txt', 'w')
       tmp = sys.stdout  # on garde un lien vers le fichier sortie standard, pour le récupérer plus tar
       print 'sur le terminal'
       sys.stdout = autre_stdout
       print 'dans le fichier'
       sys.stdout = tmp
       print 'de nouveau sur le terminal'
```