

Systemes et reseaux



Module 1 Système



Pablo Rauzy <pr@up8.edu>
pablo.rauzy.name/teaching/sr

Systeme

- ▶ Dans les systèmes modernes, il est rare de programmer sur du matériel directement.
 - Une exception pourrait être faite pour le monde des systèmes embarqués, et encore.
- ▶ Un système d'exploitation sert d'intermédiaire entre le code applicatif et le matériel :
 - le processeur communique avec la RAM mais aussi les périphériques d'entrées (clavier, souris, webcam, ...), de sorties (écran, imprimante, voyants, ...), de stockage de masse (disque dur, SSD, ...)
- ▶ En pratique, il y a souvent plusieurs couches d'abstraction qui sont utilisées :
 - Non seulement le système sert d'intermédiaire avec le matériel, mais il *virtualise* aussi celui-ci.

- ▶ Le système que nous allons utilisé concrètement dans ce cours est Linux.
- ▶ Cependant ce qu'on va apprendre s'applique à tout la famille de système POSIX.
- ▶ POSIX signifie *Portable Operating System Interface*, il s'agit de la norme IEEE 1003.
- ▶ La plupart des Unix modernes sont en grande partie compatibles POSIX
 - en plus de Linux : FreeBSD, OpenBSD, NetBSD, macOS, Minix, ...
- ▶ Pour être le plus générique possible, la norme définit des types spécifiques :
 - Exemple : le type `pid_t` est simplement défini comme un entier signé sans préciser sa taille (cf. la page de manuel), de sorte à ce que si ça change d'une version à l'autre ou d'un système POSIX à l'autre, le code utilisant ces abstractions continuent de fonctionner.

- ▶ Un *appel système* (*syscall*) est une fonction qu'on appelle depuis l'environnement utilisateur pour demander au noyau du système d'exploitation d'effectuer une tâche dans l'environnement noyau.
- ▶ Sous Linux, les appels systèmes sont accessibles indirectement via des “wrappers” présents dans la librairie C standard.
 - Ces fonctions sont documentées dans la section 2 des pages de manuel.
 - Les “vraies” fonctions étant documentées dans la section 3.
- ▶ <https://filippo.io/linux-syscall-table/>

La commande `strace`

- ▶ La commande `strace` nous permet de tracer tous les appels systèmes effectués par la commande qu'on lui passe en argument.
- ▶ Voyons ensemble quelques exemples.

La variable `errno`

- ▶ En cas de succès, les appels système renvoie le plus souvent 0 ou un valeur positive.
- ▶ La variable `errno` (`#include <errno.h>`) est assignée par (notamment) les appels systèmes lorsque une erreur survient.
- ▶ Cela permet d'identifier l'erreur.
- ▶ Cette variable est utilisée par les fonctions `perror` (`#include <stdio.h>`) et `strerror` (`#include <string.h>`) pour afficher le message d'erreur correspondant.
 - C'est ce qu'on voit dans la sortie de `strace`.

- ▶ Ce module est constitué de quatre chapitres :
 - Entrées/sorties, fichiers, répertoires ;
 - Mémoire et processus ;
 - Signaux et communication inter-processus ;
 - Module noyau.

- ▶ Chaque chapitre fera l'objet d'un TP.