# TP 2 (noté) : processus

Université Paris 7 — L3MI — Systèmes et Réseaux

Ce TP est noté. Vous devez rendre un seul fichier archive TAR, sur Didel, cours L3MISR. Cette archive ne contiendra que les fichier *.c et les commenatires ou réponses aux questions dans un fichier reponses.txt. Elle est créée par une commande telle que
`mkdir MonNom ; cp *.c reponses.txt MonNom ; tar cvf MonNom.tar MonNom/`
Notez que votre nom doit être le nom de fichier tar ET le nom du répertoire archivé.

Ce TP dure deux semaines :
- les exercices 1 et 2 doivent être rendus aujourd'hui 17 fevrier
- l'exercice 3 doit être rendu le 24 fevrier

## Exercice 1 : Deux processus concurrents

Ecrire un programme où
- Le processus père lance un fils grâce à `fork()`, puis écrit des 'p' de façon ininterrompue.
- Le fils écrira lui des 'f' en boucle infinie.

Interprétez l'affichage (répondre dans un fichier `reponses.txt`). On finira par un bon Control-C !

## Exercice 2 : Synchronisation père-fils

Écrire un programme qui lance dix fils, puis attend que tous soient terminés. En phase de lancement, le père affiche un message pour chaque fils lancé. En phase d'attente de fin, il annoncera le PID de chaque fils terminé. Quand les dix sont terminés, le père meurt. Chaque fils, lui, tirera un nombre $n$ au hasard entre 2 et 20. Il affichera son pid, celui de son père, et $n$. Puis il attendra $n$ secondes avant de mourir. Exemple d'affichage, en se limitant à 3 fils :

```
Lancement du fils 1
Lancement du fils 2
Bonjour je suis le processus 10124 de père 10117, attente 10s
Lancement du fils 3
Bonjour je suis le processus 10125 de père 10117, attente 8s
Bonjour je suis le processus 10132 de père 10117, attente 17s
Mon fils 10125 est mort
Mon fils 10124 est mort
Mon fils 10132 est mort. Plus de fils, je me termine.
```

- L'appel système `wait()` permet à un processus de se placer en attente de la terminaison d'un de ses fils. Donc l'appel de fonction bloque le père : il n'en revient que lorsqu'un fils vient de se terminer. C'est un mécanisme de synchronisation. `wait` renvoie le PID du fils qui vient de se terminer. Enfin, il faut fournir un paramètre, qui peut être `NULL` (on ne s'en sert pas dans ce TP). Voir `man 2 wait`.
- Pour l'attente voir `man 3 sleep`
- Pour tirer un nombre de secondes au hasard, voir `man 3 rand` et `man 3 srand`, ou mieux, `man 4 urandom`. Attention que, à cause des limitations de `rand()`, les dix fils ne tirent pas le même nombre "aléatoire" !
- Pour connaître les pid, voir manuel de `getpid()` et `getppid()`.

## Exercice 3 : Mini shell

Il s'agit d'écrire un *shell* (interpreteur de commandes) minimal. Votre programme
- Affiche un petit *prompt* (du style $) puis attend que l'on tape une commande au clavier
- Quand cela est fait, lance un fils
- Le fils doit executer la commande par `execv` (si possible, sinon signaler une erreur)
- Le père doit attendre la fin du fils (par `wait`) et recommencer (attend une autre commande)

L'appel système `execve()` permet de changer le code du programme correspondant au processus, c'est-à-dire que le système d'exploitation remplace le programme en cours par le nouveau, sans toucher au PID (voir `man 2 execve` pour l'appel système, ou éventuellement `man 3 exec` pour des variantes offertes par la librairie). On remarquera que si `execve()` retourne, c'est qu'il y a eu une erreur (commande non-executable).

Les trois paramètres demandés par `execve()` sont exactement les trois paramètres de `main()`. Car la syntaxe la plus complète est `int main(int argc,char *argv[], char *envp[])` où :
- `argc` contient le nombre d'arguments qui ont été passés lors de l'appel du binaire exécutable (nombre de mots dans la ligne de commande)
- `argv` contient les arguments de la ligne de commande au niveau du shell. Ces arguments sont découpés en mots et chaque mot est référencé par un pointeur dans le tableau. `argv[0]` correspond toujours au nom du binaire exécutable appelé. Le nombre de pointeurs valides dans le premier tableau est donné par le contenu de la variable `argc`.
- `envp` contient les *variables d'environnement* transmises par le shell au moment de l'exécution. Contrairement au premier tableau, la taille de ce deuxième tableau n'est pas donnée par un nombre de mots valides, mais sa fin est donnée par un pointeur NULL. On a donc une listes de chaînes de caractères, chacune étant une variable d'environnement non vide, sauf la dernière qui est vide.

Il est recommandé d'afficher pour débuggage le contenu des tableaux `argv` et `envp` avant appel. Pour `envp`, vous pouvez transmettre au fils l'environnement que vous avez reçu sans le modifier, c'est la méthode habituelle. Pour construire `argv`, on peut utiliser `strtok()` de la bibliotheque standard, qui transforme une chaine (lue par `fgets()` au clavier) en tableau de chaines en coupant à chaque espace, ou faire soi-même le travail.