# SY5 – Systèmes d'exploitation
# TD n° 7 : création de processus

### Exercice 1 : `fork` et `wait`

Soit le programme suivant (à lire sans le tester).

```
int main() {
  pid_t r;
  switch (fork()) {
    case -1:
      perror("fork");
      exit(1);
    case 0:
      write(STDOUT_FILENO, "a", 1);
      exit(0);
    default:
      write(STDOUT_FILENO, "b", 1);
      switch (r = fork()) {
        case -1:
          perror("fork");
          exit(1);
        case 0:
          write(STDOUT_FILENO, "c", 1);
          exit(0);
        default:
          waitpid(r, NULL, 0);
          write(STDOUT_FILENO, "d", 1);
          exit(0);
    }
  }
}
```

1. Que fait-il ?
2. Quelles sont les différentes sorties possibles ?

### Exercice 2 : contrôler la généalogie

1. Écrire un programme où le processus père crée $k$ fils, où $k$ est donné en argument.
2. Que faut-il ajouter si on veut que le père attende la fin des $k$ fils avant de se terminer ?
3. Écrire un programme qui crée un unique fils, qui lui-même crée un unique fils et cela jusqu'à une hauteur de hiérarchie de $k$.

### Exercice 3 : `fork`-bombe

Que pensez-vous des deux programmes suivants (à lire sans les tester) ?

```
int main() {
  while (1) {
    fork(); /* gestion d'erreur omise */
    sleep(2);
    write(STDOUT_FILENO, "*", 1);
  }
}
```

```
int main(int argc, char* argv[]) {
  fork(); /* gestion d'erreur omise */
  sleep(2);
  write(STDOUT_FILENO, "*", 1);
  execvp(argv[0], argv);
  exit(1);
}
```

À votre avis, pourquoi a-t-on mis des `sleep` ?

### Exercice 4 : `exec`

**1.** Écrire un programme `execute` qui prend en argument une commande et ses éventuels arguments, l'exécute, puis affiche un message indiquant que l'exécution est terminée.
Par exemple :

```
$ execute gzip -v gros_fichier
gros_fichier:    30.3% -- replaced with gros_fichier.gz
... exécution terminée
```

**2.** Modifier le programme précédent pour afficher le code de retour de la commande exécutée. Par exemple :

```
$ execute ls ugwujhdgaj
ls: ugwujhdgaj: No such file or directory
... exécution terminée avec le code de retour 1
```

**3.** Simuler la commande « `sleep 5 ; ps aux` », c'est-à-dire écrire un programme qui exécute d'abord la commande « `sleep 5` », puis, une fois celle-ci terminée, la commande « `ps aux` ».

**4.** Simuler la commande « `ls fic && echo "fichier trouvé"` », *i.e.* écrire un programme qui prend en paramètre un nom de fichier *fic*, et exécute d'abord la commande « `ls` *fic* » puis, *si elle s'est bien passée*, exécute la commande « `echo "fichier trouvé"` » *(on demande ici d'utiliser la commande* `echo` ***externe****).*

**5.** Que faut-il modifier pour simuler « `ls` *fic* `|| echo "fichier introuvable"` », *i.e.* exécuter d'abord la commande « `ls` *fic* » puis, *si elle s'est mal passée*, la commande « `echo "fichier introuvable"` » ?

### Exercice 5 : `dup`

On souhaite combiner les programmes des questions 4 et 5 de l'exercice 4, et en améliorer l'affichage en éliminant celui de « `ls` *fic* ». Une solution consiste à rediriger les deux sorties standard sur `/dev/null`, autrement dit à simuler :

« `ls` *fic* `&> /dev/null && echo "fichier trouvé" || echo "fichier introuvable"` »

Écrire le programme correspondant.