I2181-B
LINUX :
APPELS SYSTÈME

FORK & EXEC

FORK

- Appel système responsable de la création de processus.
- Crée un nouveau processus (appelé processus fils), en dupliquant le processus qui appelle le fork.
- Le processus appelant (celui qui s'exécute lors de l'appel à fork) est le processus parent.
- Le processus fils possède son propre espace mémoire, qui réplique celui du parent au moment de sa création.

FORK

- L'enfant et le parent ont leur « process ID » (pid) propres. Le « parent process ID » (ppid) de l'enfant est le pid du parent.
- Le programme continue à s'exécuter dans les deux processus ! Du code **non conditionné** à l'un des processus en particulier s'exécute donc 2 fois (une fois chez le parent et une fois chez l'enfant).

FORK

- Les *file descriptors* chez l'enfant pointent sur les mêmes ressources que chez le parent.
- Consulter le man pour le détail de ce qui est hérité par / dupliqué chez l'enfant et ce qui ne l'est pas.
- Objectif : déléguer un travail à un autre processus, tout en continuant son travail à soi en parallèle.
- Un processus parent peut attendre explicitement la fin d'un processus fils : appel système `wait`.

FORK

```
#include <sys/types.h>
#include <unistd.h>
pid_t fork()
```

- Si tout va bien, renvoie 0 dans le processus fils et renvoie le pid de l'enfant dans le processus parent.
- Renvoie -1 en cas d'erreur.

FORK

- Pour conditionner l'exécution au fait de se trouver dans le processus fils ou dans le processus parent, on utilisera une conditionnelle (if) sur la valeur de retour du `fork`.

```
int childId = fork();
if (childId != 0) {
    /* In the parent process */
} else {
    /* In the child process */
}
```

WAIT

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t waitpid(pid_t pid,
              int *wstatus, int options)
```

où **pid** indique le(s) enfant(s) concerné(s)

wstatus = statut de l'enfant déclencheur

options = 0 pour comportement par défaut

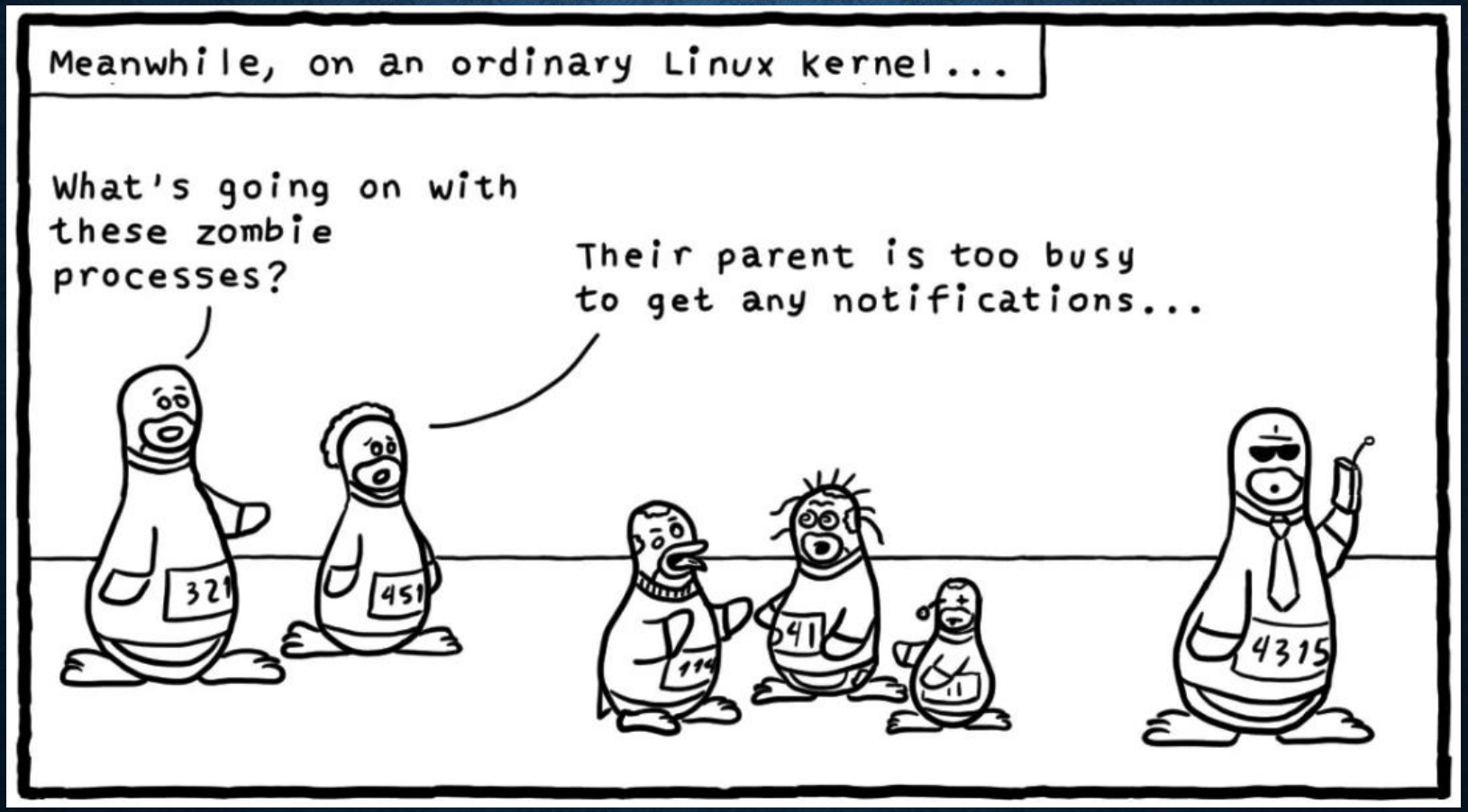
WAIT

- Voir le man pour les différentes possibilités et utilisations des arguments.
- Appel bloquant ; attend qu'un processus fils se termine.
- Renvoie le pid du processus fils terminé, ou -1 en cas d'erreur.

WAIT

- Si un processus fils est déjà terminé, mais non encore sollicité par un `wait` de son processus parent, il est qualifié de « zombie »
- `wait` renvoie son pid directement.

Processus zombie



EXEC

- Les fonctions de type « **`exec`** » remplacent le processus courant par un nouveau processus (ayant le même pid) exécutant le fichier spécifié en paramètre.
- Ces fonctions utilisent l'appel système **`execve`**.
- Le fichier à exécuter sera soit un exécutable correspondant à la fonction linux désirée,
  soit un script (commençant par un *shebang* :
  #! *interpréteur*).

EXEC

```
#include <unistd.h>

int execl(const char *path,
          const char *arg, …)
```

- où `path` indique le pathname du fichier à exécuter

  `arg0` est le nom de l'exécutable (qui apparait aussi dans le path !)

  arg1, arg2… sont les arguments éventuels

EXEC

- Renvoie `-1` en cas d'erreur
- Sinon, ne renvoie rien : le nouveau processus remplace le processus courant, donc il n'y a rien à renvoyer à ce dernier …

EXEMPLE

- Créer un processus fils
- Dans le processus parent : 1er affichage,
  puis attendre fin du processus fils,
  puis 2ième affichage
- Dans le processus fils : remplacement par une exécution de la commande `ps -l` via un script de type bash

EXEMPLE

- Ecriture du script à exécuter par l'enfant (fichier « myScript.sh » qu'il faut rendre exécutable (chmod)).
- Première ligne = *shebang* (#!) indiquant où se trouve l'interpréteur permettant d'exécuter le script.

```
#!/bin/bash
ps -l
```

→ **Voir archive *LAS_02_exemple***
(4 versions du programme)