I2180 PROGRAMMATION SYSTÈME :

FORK & EXEC

FORK

- 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`.

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` (processus « zombie »), `wait` renvoie son pid directement.

EXEC

- Les fonctions de type « **`exec`** » remplacent le processus courant par un nouveau processus exécutant le fichier spécifié en paramètres.
- Ces fonctions utilisent l'appel système **`execve`**.
- Le fichier à exécuter sera soit l'exécutable correspondant à la fonction linux désirée, soit un script de type shell.

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 : 1ier 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
```

EXEMPLE

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>


int main(int argc, char **argv){
```

EXEMPLE

```
int childId; /* pid de l'enfant */
int waitId; /* pid renvoyé par le wait */
int status; /* Statut de l'enfant terminé */

/* Création du processus fils */
if((childId = fork()) == -1){
  perror("Fork failed");
  exit(10);
}
```

EXEMPLE

```
if(childId){ /* Dans le processus parent */
  printf("Processus parent en attente de
          la fin de son enfant.\n");
  if((waitId = waitpid(childId, &status,
                       0)) == -1){
    perror("Wait error\n");
    exit(20);
  } else if(waitId != childId) {
    /* Impossible */
    printf("M'enfin, qui m'a fait un enfant
        dans le dos ?!\n");
```

EXEMPLE

```
    } else {
      printf("Processus parent se termine
              après son enfant.\n");
    }
  } else { /* Dans le processus fils */
    execl("myScript.sh","myScript.sh", NULL);
    perror("Exec failed\n");
    exit(30);
  }
}
```

EXEMPLE : OUTPUT

```
Processus parent attend la fin de son enfant (4965).
…    UID    PID    PPID  …  CMD
   10418   4763   4732      bash
   10418   4964   4763      a.out
   10418   4965   4964      myScript.sh
   10418   4966   4965      ps
Processus parent se termine après son enfant.
```