Dans ce TP nous nous proposons de découvrir la famille des fonctions `exec*` (en particulier `execvp`) et la fonction `dup`. La fonction `execvp` permet de lancer un programme plus ou moins de la même façon que vous le feriez dans un terminal (recherche dans les répertoires de la variable d'environnement `PATH` incluse.) Le prototype de cette fonction est le suivant :

```
int execvp(const char *file, char *const argv[]);
```

Le paramètre `file` est le nom du programme ou un chemin vers le programme à exécuter. Si `file` ne contient pas de le caractère `'/'` alors la variable d'environnement `PATH` est utilisée pour trouver le chemin vers le programme.

Le paramètre `argv` correspond aux paramètres passés au programme à lancer, ce sont les valeurs qui se retrouveront en paramètres de la fonction `main` du programme. Comme dit précédemment dans le cours, le premier élément du tableau `argv` du `main` (`argv[0]`) est le nom du programme lancé et l'élément suivant le "dernier" élément du tableau `argv` (`argv[argc]`) a pour valeur `NULL`. Cette valeur sert de sentinelle pour connaître la fin du tableau `argv` sans connaître sa taille (comme `'\0'` dans une chaîne de caractères.)

Un exemple d'utilisation de `execvp` pourrait être le suivant :

```
char* args[] = {"ls", "-l", NULL};

execvp(args[0], args);
perror("");
```

Comme vous pouvez le voir dans cet exemple, le dernier élément du tableau est la valeur `NULL`.

Vous pouvez aussi observer dans cet exemple que nous ne testons pas la valeur de retour de la fonction `execvp`. Ceci est normal car la fonction `execvp` est une fonction dite recouvrante dans le sens où le programme qui appelle cette fonction est remplacé (recouvert) par le programme spécifié en paramètre de la fonction `execvp`. Ainsi, si la fonction `execvp` échoue, le code qui se trouve après sera exécuté car non recouvert. N'hésitez pas à lire la page de la fonction `execvp` : `man execvp`.

Q 1. Écrivez un programme qui lance la commande `ls -l` comme montré précédemment.

Q 2. Modifiez le programme pour faire exécuter la commande `ls -l` dans un processus fils et afficher un message de votre choix dans le père.

Q 3. En utilisant la fonction `wait` ou `waitpid` vue précédemment, faites en sorte que le message affiché par le père le soit après la fin de l'exécution de la commande `ls -l`.

Comme vous avez pu le voir, la fonction `execvp` recouvre le programme qui l'appelle. Cependant ce recouvrement "épargne" les descripteurs de fichiers déjà ouverts ainsi que les handlers de signaux déjà installés (nous verrons dans un prochain TP ce que sont les signaux.) Le fait que les descripteurs de fichiers soient épargnés nous permet de mettre en place de la redirection d'entrée/sortie.

La redirection d'entrée/sortie consiste à faire lire ou écrire un processus sur un descripteur de fichier dont la valeur est connu à l'avance (généralement l'entrée standard, la sortie standard ou la sortie standard des erreurs) mais qui a été redirigé vers un autre descripteur de fichier. Ainsi un processus pensant lire sur son entrée standard pourrait lire dans un fichier ou sur un pipe par exemple.

Cette redirection est mise en place grâce à l'utilisation de la fonction `dup` dont voici le prototype :

```
int dup(int oldfd);
```

La fonction `dup` permet de dupliquer le descripteur de fichier `oldfd`. Le duplicata utilisera le descripteur de fichier le plus petit disponible. Ainsi, si on ferme l'entrée standard d'un processus et qu'on duplique le descripteur de fichier issue de l'ouverture d'un fichier en lecture, le duplicata prendra le descripteur de fichier de valeur `0` celui-ci venant d'être libéré par la fermeture de l'entrée standard.

Le code suivant illustre cet exemple :

```
int pid = fork();
if(pid == 0) {
    int fd = open("file", O_RDONLY);
    if(fd != -1) {
        close(0);
        dup(fd);          /* dup doit retourner 0 */
        close(fd);
        getchar();        /* Récupère le premier caractére du fichier file */
    }
}
```

On se rappellera que :
- Entrée standard :
  - Lecture
  - Descripteur de fichier : 0
- Sortie standard :
  - Écriture
  - Descripteur de fichier : 1
- Sortie standard des erreurs :
  - Écriture
  - Descripteur de fichier : 2

Q 4. Écrivez un programme qui crée un processus fils dans lequel un fichier est ouvert en lecture seule, l'entrée standard est redirigée vers ce fichier et la commande `cat` est exécutée au moyen de la fonction `execvp`. Le nom du fichier à ouvrir devra être passer en paramètre de votre programme.

Nous avons vu précédemment comment faire communiquer deux processus au moyen d'un pipe. Nous savons que la fonction `pipe` nous fournit deux descripteurs de fichiers pour lire et écrire sur un pipe.

Q 5. Écrivez un programme qui crée un pipe et un processus fils. Dans le processus fils, l'entrée standard sera redirigée vers le descripteur de fichier en lecture du pipe et la commande `grep -e ^-.w.*$` sera exécutée au travers d'un appel à la fonction `execvp`. Dans le processus père, la sortie standard sera redirigée vers le descripteur de fichier en écriture du pipe et la commande `ls -l` sera exécutée au travers d'un appel à la fonction `execvp`. On fera attention à fermer les extrémités non utilisées du pipe dans chacun des processus.

Q 6. Modifiez le programme précédent pour que la commande `ls -l` s'exécute dans une processus fils et pour que le père attende la fin de ces deux fils.

Q 7. Modifiez le programme précédent pour ajouter les commandes `cut -d " " -f 5-` et `sort -n` à la suite des commandes `ls -l` et `grep -e ^-.w.*$`.

Q 8. Modifiez le programme précédent pour que la valeur du paramètre `-e` de la commande `grep` soit en paramètre de votre programme.