Langage C et Programmation Système
TP n° 9 : fork, exec, wait

Exercice 1 :

1. Écrivez un programme `print.c` qui prend comme arguments deux entiers, et affiche le premier entier un nombre de fois égal au deuxième entier.
2. Écrivez un programme qui utilise `execl` pour appeler le programme de la question précédente dans deux processus différents (utilisez `fork`). Testez votre programme avec des grands arguments pour `print.c`. Qu'est-ce que vous observez ?

Exercice 2 :

Écrivez un programme qui demande à l'utilisateur d'entrer un entier `n` et une suite de `n` entiers qui va être stockée dans un tableau `tab`.

1. Complétez le programme pour calculer (et afficher) la somme et le produit des éléments dans cette suite dans deux processus différents (utilisez `fork`).
2. Créez une autre version du programme qui calcule la somme des éléments dans la suite en utilisant deux processus fils. Plus précisément, chaque fils va calculer la somme des elements dans une moitié du tableau, envoyer son résultat au processus père (en utilisant les fonctions `exit` et `waitpid`). Le processus père est responsable d'additionner les résultats, et d'afficher la somme finale.
   Attention : Cette démarche abuse de la valeur retournée par `exit`*, qui sert normalement à signaler au processus père quelle erreur s'est produite (ou* 0 *en cas de succès). Cela ne fonctionne pas correctement pour des valeurs en dehors de l'intervalle* $[0, 255]$.
3. *Optionnel :* Étendez le programme de la question précédente à un nombre arbitraire de fils, donné par l'utilisateur.

Exercice 3 :

Écrivez un programme qui lit les fichiers dans un répertoire donné comme argument, crée un nouveau sous-repertoire appelé `backup` dans le même repertoire, et copie tous les fichiers du repertoire initial dans `backup`. Utilisez une des fonctions de la classe `exec` avec la commande `cp`. Créez un processus fils pour copier chaque fichier.

Utilisation de tubes : Pour créer un tube on utilise l'appel système `pipe()` qui reçoit comme argument un tableau de deux entiers :

```
int fd[2];
pipe(fd);
```

À la suite de cet appel, le premier entier `fd[0]` est le descripteur utilisé pour lire dans le tube et `fd[1]` est le descripteur utilisé pour écrire.
Le code suivant décrit une situation où on veut créer un processus fils qui envoie des messages vers le processus père en utilisant le tube créé ci-dessus :

```
pid_t   childpid;
if ( (childpid = fork()) == -1 ) {
   perror("fork");
   exit(1);
}
if (childpid == 0) {
   // Le fils ferme le coté lecture du tube
   close(fd[0]);
   // Le fils écrit un message
   char    string[] = "Hello, world!\n";
   write(fd[1], string, (strlen(string)+1));
   exit(0);
}
else {
   // Le parent ferme le cote écriture du tube
   close(fd[1]);
   // Le parent lit un message
   char    readbuffer[80];
   read(fd[0], readbuffer, sizeof(readbuffer));
}
```

La fermeture des cotés lecture ou écriture est nécessaire pour éviter des problèmes d'interblocage. Les descripteurs stockés dans `fd` peuvent être utilisés comme des descripteurs de fichiers habituels, en particulier comme arguments de `read` et `write`.

Exercice 4 : Crible d'Ératosthène

Le crible d'Ératosthène est un algorithme pour calculer les nombres premiers jusqu'à une borne donnée `n`. Le comportement de l'algorithme pour `n`=26 est décrit ci-dessous :

$2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26$

$\mathbf{2}, 3, \not{4}, 5, \not{6}, 7, \not{8}, 9, \not{10}, 11, \not{12}, 13, \not{14}, 15, \not{16}, 17, \not{18}, 19, \not{20}, 21, \not{22}, 23, \not{24}, 25, \not{26}$

$\mathbf{3}, 5, 7, \not{9}, 11, 13, \not{15}, 17, 19, \not{21}, 23, 25$

$\mathbf{5}, 7, 11, 13, 17, 19, 23, \not{25}$

$\mathbf{7}, 11, 13, 17, 19, 23, 25$

$\mathbf{11}, 13, 17, 19, 23, 25$

...

Écrivez un programme qui affiche tous les nombres premiers inférieurs à un entier `n` donné comme paramètre, et qui implémente le crible d'Ératosthène. Le calcul sur chaque ligne se fait dans un processus différent et les processus communiquent à travers des tubes. Plus précisément,

- le processus principal crée un processus fils et il lui envoie à travers un tube tous les entiers entre 2 et `n`,
- le processus qui simule la ligne `i` affiche le premier entier `p` de la suite qu'il reçoit (du processus qui simule la ligne `i-1` ou du processus principal), crée le processus qui simule la ligne `i+1`, et il lui envoie tous les entiers qu'il reçoit sauf ceux qui sont des multiples de `p`.

Faites attention à la condition d'arrêt.