Arguments et Fichiers

1 Écrire un programme compte_arg.c qui prend en argument une liste d'entier et qui affiche la somme de ces entiers. Attention au type de argv - la conversion en entier doit donc être faite :

— soit par atoi :

```
n = atoi(argv[i]);
```

— soit par sscanf :

```
sscanf(argv[i], "%d", &n);
```

□

2 Écrire le programme cp.c qui réalise la copie d'un fichier donné en premier paramètre dans un autre fichier donné en second paramètre. Vous utiliserez les primitives systèmes suivants : open, read, write, close. □

3 Écrire le programme cp_reverse.c qui réalise la copie d'un fichier donné en premier paramètre dans un autre fichier donné en second paramètre mais en inversant le contenu du fichier. Si le premier fichier contient abcde, sa copie contiendra edcba. Vous utiliserez les primitives précédentes ainsi que lseek. □

Processus

(4) Écrire un programme qui affiche les information suivantes associées à un processus :

- Le numéro du processus (`pid`)
- le numéro du père du processus (`ppid`)
- l'UID réel du processus (`uid`)
- l'UID effectif du processus (`euid`)
- le GID réel du processus (`gid`)
- le GID effectif du processus (`egid`)

Un exemple d'exécution est :

```
./a.out
Je suis le processus de pid      : 20011
Mon père est le processus de pid :  5411
Mon uid                          :   322
Mon euid                         :   322
Mon gid                          :   100
Mon egid                         :   100
```

□

(5) Écrire un programme qui crée un processus fils et qui affiche les informations `pid` et `ppid` de chaque processus créé. Un exemple d'exécution est :

```
./a.out
Valeur de fork = 22723
Je suis le processus père : pid=22722, ppid=5411,pid fils = 22723
Valeur de fork = 0
Je suis le processus fils : pid=22723, ppid 22722
```

□

6 Reprendre l'exercice 1 et affichez les informations relatives aux processus père et fils comme suit :

```
/a.out

Valeur fork = 0
Je suis le processus de pid      : 22851
Mon père est le processus de pid : 22850
Mon uid                          : 322
Mon euid                         : 322
Mon gid                          : 100
Mon egid                         : 100
Mon repertoire de travail        : "/pau/homep/profs/pr/exemple"

Valeur fork = 22851
Je suis le processus de pid      : 22850
Mon père est le processus de pid : 5411
Mon uid                          : 322
Mon euid                         : 322
Mon gid                          : 100
Mon egid                         : 100
Mon repertoire de travail        : "/pau/homep/profs/pr/exemple"
```

□

Multiprocessing

7 **Exécution concurrente des processus père et fils :**

Lire le code suivant :

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void main()
{
 pid_t p;
 p=fork();
 switch(p)
 {
  case (0):
    //sleep(15);
    printf("Le fils pid est =%d et mon ppid est=%d\n", getpid(), getppid());
    break;
  case (-1):
    //sleep(15);
    printf("Erreur fork\n");
    break;
  default:
    printf("Le pere pid est =%d et mon ppid est=%d\n", getpid(), getppid());
 }
 printf("Fin du processus %d\n",getpid());
}
```

Reprendre le programme ci-dessous et complétez en affichant l'`uid`, le `gid`, et le contenu d'une variable `x` initialisée à `2` (avant le fork) et modifié selon `x+3` par le fils et selon `x*5` par le père. Que remarquez-vous ? □

8 **Fork avec `wait, waitpid` et macros :**

- Écrire un programme fork1.c qui utilise la primitive fork() pour créer un processus fils :
  - Dans le processus père vérifier que le fork s'est bien passé (code de retour différent de -1).
  - Le processus père devra afficher son PID, le PID de son propre père et celui de son fils, attendre la terminaison du fils et quitter. Vous utiliserez pour cela la fonction wait.
  - Le processus fils devra afficher son PID, le PID de son propre père.
- Écrire un programme fork2.c, modification de fork1.c, où la fonction wait est remplacée par la fonction waitpid.
- Modifiez le code pour que le père affiche le code de retour du fils en utilisant les macros : WIFEXITED et WEXITSTATUS.

□

9 **Boucle de fork :**

- Écrire un programme fork_loop.c qui réalise une boucle for variant de 0 à $n-1$ et qui, à chaque itération, effectue un fork. n est un paramètre du programme lu sur la ligne de commande (i.e. en utilisant argv et argc).
- D'après vous, pour n fixé, combien de processus fils sont créés par ce programme ? Pour vérifier, après le fork, placez un affichage (pid, valeur retour du fork, pid du père).

□

10 **Synchronisation des processus père et fils par la commande `wait` :**
Écrire un programme qui prend une matrice de taille 2*2 en paramètre (on peut entrer 4 paramètres correspondant au 4 valeurs de la matrice) et crée quatre fils. Chaque fils calcule un élément du carré de la matrice initiale et le renvoie au processus père comme code de retour. □

Recouvrement

(11) Compiler et exécuter le code `TP1_execl.c` (disponible sur Teams). Modifiez le code pour utiliser le programme précédent en appelant la commande `execlp` au lieu de `execl`. □

(12) Compiler les codes `TP1_recouv.c` et TP1_calc.c (disponibles sur Teams). Expliquez comment invoquer `TP1_recouv` et donnez la suite d'affichage réalisée par son exécution. □

(13) Compiler le code `TP1_nouveau.c` (disponible sur Teams) et modifiez le programme `TP1_recouv.c` pour l'exécuter en utilisant une instruction `execvp()`. □

(14) Création d'un mini-shell :
Écrire un interpréteur de commandes externes (exemples ps, ls, gcc, ...). Les étapes (simplifiées) d'un shell sont les suivantes :

- le shell lit une ligne de commande sur son entrée standard.
- le shell interprète cette commande (on ne s'intéresse pas au détail de cette analyse ici) et l'exécute.
- il recommence à l'infini jusqu'à l'introduction de la commande `exit`.

□

Entrée-Sortie

(15) Écrivez un programme qui ouvre un fichier nommé «`toto`», en lecture et écriture, dont le contenu est la suite `123456789`. Le programme fait ensuite un `fork()`; le fils écrit `ab` dans le fichier, ensuite il s'endort et après il lit 2 caractères; le père s'endort, ensuit il lit 2 caractères et après il écrit `AB` dans le fichier. □