TP2 SR01: Programmation Système

A remettre le : 30 décembre 2019

- Ce TP doit se faire en binôme. Les deux étudiants formant un binôme doivent appartenir au même groupe de TD.
- Les différents binômes deveront remettre un seul fichier **Nom1-Nom2.zip** contenant les codes sources ainsi qu'un rapport électronique (pdf) de quelques pages.
- Le dernier délai pour la remise des TP est fixé pour le **30 décembre 2019**.

1 Exercice 1

1.1 Partie 1

Soient les programmes en C suivants :

1.1.1 Prog1:

```c
#include<unistd.h>
int main(){
  (fork() || fork()) && (fork() || fork());
}
```

Listing 1: prog1.c

1.1.2 Prog2:

```c
#include<unistd.h>
#include<stdlib.h>
int main(){
  int i=0;
  fork();
  while(i<4){
    if(getpid()%2==0){
      fork();
    }
    i++;
  }
}
```

Listing 2: prog2.c

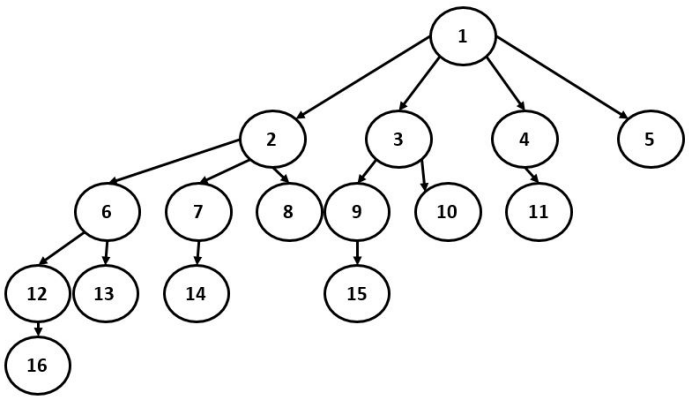
On suppose dans les deux programmes que :

- le PID du shell est 2000,
- le PID du processus correspondant au parent est 2400,
- la numérotation des processus est séquentielle (incrémentation par 1),
- un processus de PID p ne peut être exécuté qu'après la fin de l'exécution du processus de PID $p - 1$.

Question : Donner l'arbre généalogique des processus générés par chaque programme.

1.2 Partie 2

Soit l'arbre généalogique de processus suivant :



Question : En utilisant la fonction *fork()*, proposer un programme C permettant de générer cet arbre de processus. Pour chaque processus, afficher son PID et celui de son père.

2 Exercice 2

Un gestionnaire d'applications est un programme lancé au démarrage du système d'exploitation. Il se charge de lancer et de gérer un ensemble d'applications nécessaires au bon fonctionnement du système (gestion des cartes réseau, gestion des périphériques...). Dans cet exercice, on va programmer un gestionnaire d'applications personnalisé (*Application-Manager*). La liste des applications à lancer est stockée dans le fichier *list_ appli.txt*. Vous disposez de quelques exemples d'applications (*power_ manager.c*, *network_ manager.c*, *get_ time.c*).

Exemple de fichier *list_ appli.txt* contenant deux applications:

```
nombre_applications=2
name=Power Manager
path=./power_manager
nombre_arguments=2
arguments=
./mise_en_veille.txt
4

name=Get Time
path =./get_time
nombre_arguments=0
arguments=
```

1. Écrire un programme *ApplicationManager.c* qui doit:
   - créer un ensemble de processus fils chacun est responsable à l'exécution d'une application.
   - lors de l'arrêt d'une application, informer l'utilisateur en lui affichant le nom de l'application terminée.
   - s'arrêter après avoir fermer toutes les applications en cours d'exécution lors de la réception d'un ordre de mise en veille de la part de *power_ manager* (signal **SIGUSR1**).
2. Modifier le programme *power_ manager.c* pour envoyer le signal **SIGUSR1** à l'*ApplicationManger* lorsque l'utilisateur tape 1 dans le fichier *mise_ en_ veille.txt*

NB: Lorsque *ApplicationManger* reçoit un signal **SIGUSR1** de la part d'un autre processus, il ne ferme pas les applications.

3 Exercice 3

L'objectif de cet exercice est de paralléliser le calcul de la somme ou le produit de deux matrices. Afin de réaliser une telle tâche, vous allez procéder comme suit :

1. Créez un programme **Somme.c**, qui reçoit comme paramètre, dans la fonction main, deux chemins vers deux fichiers binaires contenant chacun une matrice carrée de la même taille, et le nombre de lignes N d'une des matrices. (Lors de l'implémentation de ce programme, supposez que les matrices sont déjà saisies dans les fichiers). Dans le programme **Somme.c**, on commence par lire les deux matrices. Ensuite, on crée N processus fils. Chaque processus fils i calcule la somme des ième lignes des deux matrices et communique le résultat au processus père en utilisant les pipes.

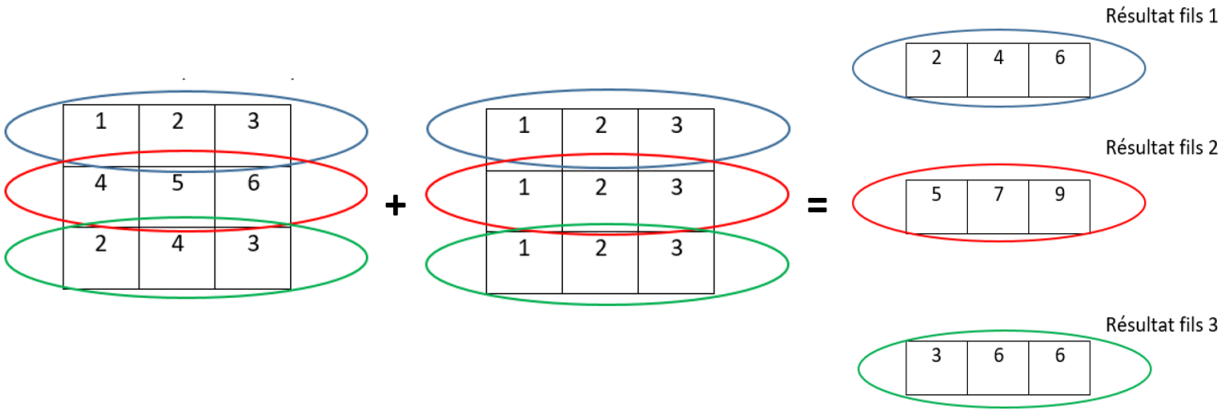


Figure 1: Exemple somme

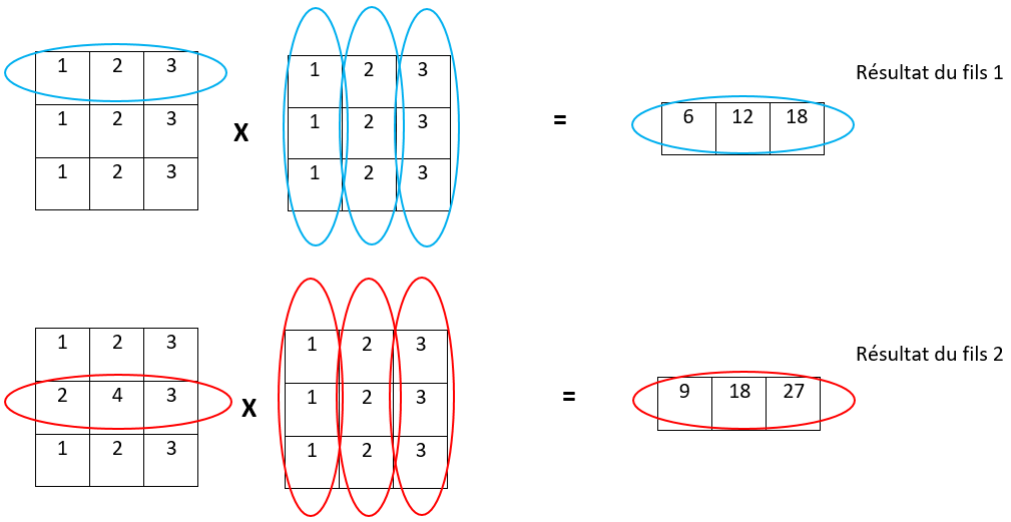


Figure 2: Exemple produit

Une fois que le processus père récupère la somme de chaque ligne, il affiche la matrice résultante.

2. De même, créez un programme **Produit.c** qui reçoit les mêmes paramètres que le programme **Somme.c** mais qui effectue le produit de deux matrices. Dans le programme **Produit.c**, vous devez créer N processus fils (tout comme le programme **Somme.c**). Cependant, chaque processus fils va calculer le produit de la ième ligne de la première matrice avec toutes les colonnes de la deuxième matrice. Ensuite, il communique le résultat au processus père.

3. Créez un programme **ManipMatrice.c** qui permet de saisir des matrices ou de les générer aléatoirement, de les stocker dans des fichiers binaires et de faire appel, en utilisant *execv()*, aux programmes **Somme.c** ou **Produit.c** selon le choix de l'utilisateur.

   Remarques :

   - nous tenons à insister que la somme et le produit doivent se faire en parallèle, donc une approche dans laquelle le processus père attend la fin du traitement réalisé par un fils afin d'en créer un autre ne sera pas acceptée.
   - L'écriture et la lecture des fichiers doivent se faire en utilisant *fread()* et *fwrite()*.