PHAM Nu-Huyen-Trang et NGUYEN Van-Triet
Université de Technologie de Compiègne
9 décembre 2018

Devoir 2 - SR01

Programmation Système

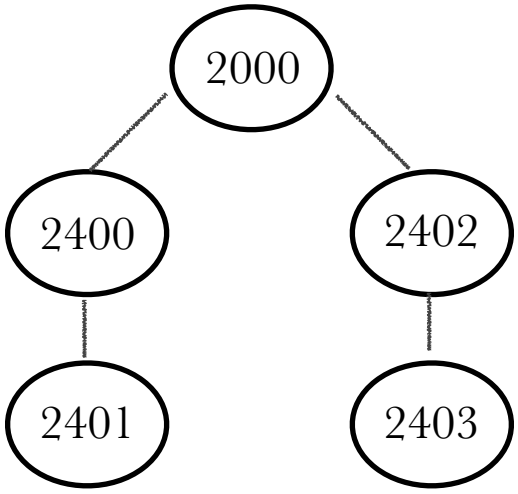
Exercice 1:

1.1 Partie 1:

a. Programme 1:

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

Le processus courant (appelons-le le père) engendre dans l'ensemble 4 autres processus. En effet, comme dans une instruction a && b, b n'est pas évaluée si l'évaluation de a donne 0, de même, dans dans une instruction a || b, b n'est pas évaluée si l'évaluation de a ne donne pas 0. Donc, nous avons trouvé l'arbre généalogique des processus comme ci- dessous:



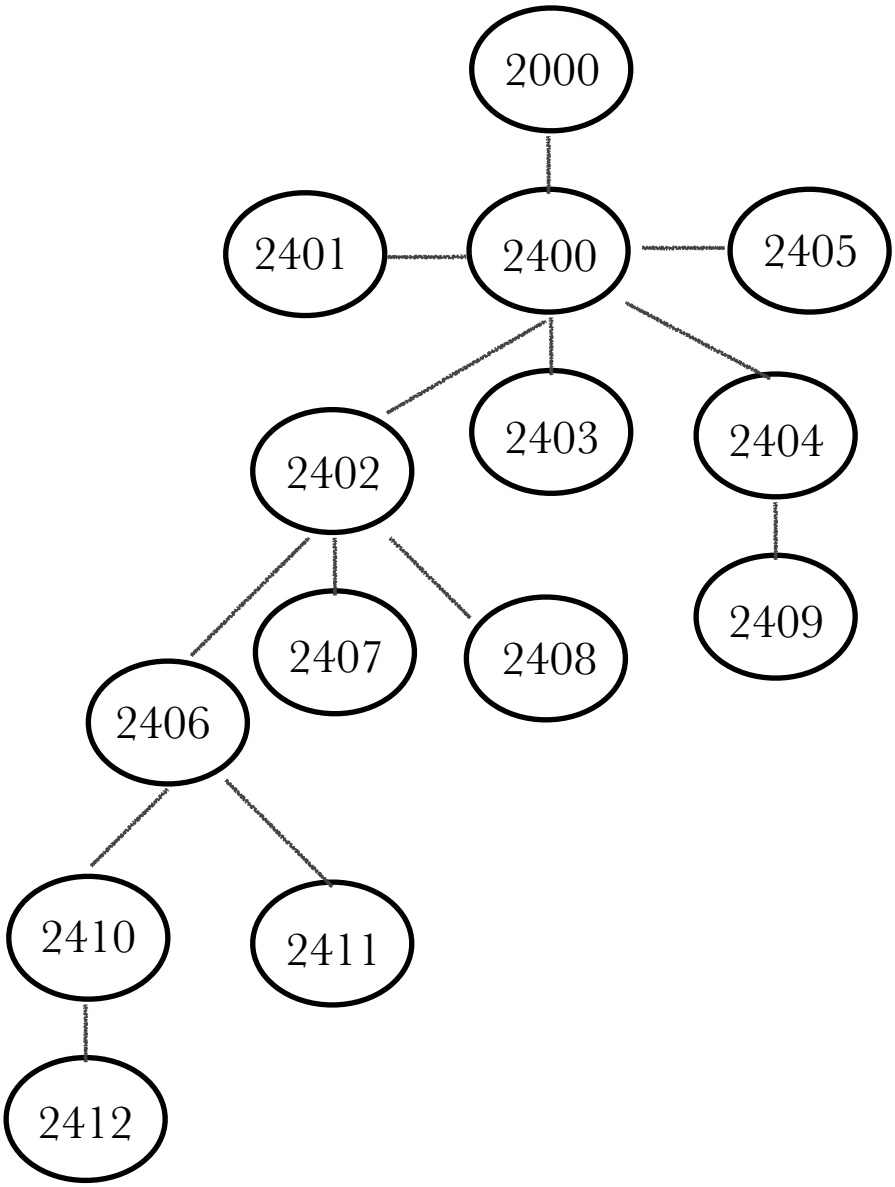
Pour s'en convaincre, nous avons exécuter le programme 1 sous linux et trouve le résultat comme l'illustration ci-dessous:

```
macbook-air-de-nu:ExoC bipham$ gcc -Wall -pedantic test.c -o test
macbook-air-de-nu:ExoC bipham$ test & (sleep 1; ps -ao ppid,pid,comm)
[1] 1101
 PPID   PID COMM
  649   650 login
  650   651 -bash
  651  1102 -bash
 1102  1104 ps
[1]+  Exit 1                  test
```

b. **Programme 2:**

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

Après le shell, vu fork() après l'instruction int i=0, le processus 2400 a été créé. Puis, vu que 2400 est pair, le processus 2401 a été créé quand i=0, et il arrête car 2401 est impair. Puis, on retourne vers le père 2400 pour créer les autres fils. On continue à dessiner l'arbre par exécuter et finir un branche d'arbre puis retourne vers le père et nous avons trouvé l'arbre comme ci-dessous:


2000
2401
2400
2405
2402
2403
2404
2409
2407
2408
2406
2410
2411
2412

Pour s'en convaincre, nous avons exécuter le programme 2 sous linux et trouve le résultat comme l'illustration ci-dessous:

```
macbook-air-de-nu:ExoC bipham$ gcc -Wall -pedantic test.c -o test
macbook-air-de-nu:ExoC bipham$ test & (sleep 1; ps -ao ppid,pid,comm)
[1] 1167
 PPID   PID COMM
  649   650 login
  650   651 -bash
  651  1168 -bash
 1168  1170 ps
[1]+  Exit 1                  test
```

1.2 Partie 2:

Pour créer l'arbre comme le sujet donné, j'ai utilisé fork et boucle while pour créer les nouveaux processus. Vu qu'il faut terminer tous les processus fils avant retourner vers le père donc j'ai mis le boucle while pour que la création de fils soit fermée et limitée comme l'arbre donné.

Exercice 2:

Tout d'abord, j'ai fait un procédure void setInfo pour prendre tous les informations nécessaires pour l'exécution les applications. J'ai créé un structure qui contient les informations: nom, path, nombre d'arguments et les arguments. Vu qu'il y a quelques difficultés dans la lecture du fichier list_appli.txt, j'ai utilisé le format « name=%s », « path=%s » par exemple pour récupérer les informations.

Ensuite, pour exécuter les applications dans un processus fils, il faut créer un tableau de pid sous format pid_t et nous avons utilisé le boucle for entre 1 et nombre d'applications. Nous avons pris la fonction execv (path, tableau d'arguments) avec les informations que j'ai récupérés dans le fichier. Dans le fichier list_applis,txt, il y a 3 applications mais le nombre d'applis est 2 donc nous avons pensé qu'il y a une erreur donc nous avons le corrigé en 3. De plus, d'après nous, il est plus logique que les applications network_manager et get_time sont exécutés en premiers puis terminer par l'application power_manager (voir les informations puis envoyer le signal pour terminer tous) alors nous avons inversé l'ordre de ces applications, network_manager et get_time sont en premiers, power_manager est le dernier.

Enfin, en expliquant un peu sur le principe du signal, nous avons créé un structure signalisation, vidé les masques et aussi un procédure pour récupérer le signal quand l'application power_manager envoie « kill ».

Exercice 3:

Dans cette exercice, nous avons fait une presque longue chemin: créer un bibliothèque fonction.h avec la gestion des fonctions calculer somme et produit de 2 matrices. Dans chacune de ces fonctions, nous avons utilisés le principe pipe, c'est à dire on calcule la somme de chaque ligne de matrice puis envoyer le résultat par le tube[0], tous les résultats fils seront après envoyés par le tube[1] au tableau père.

Concernant le programme principal, j'ai inclus le bibliothèque fonction ainsi le choix de calculer la somme ou le produit de 2 matrices. Puis, nous avons utilisé la fonction exec pour exécuter le programme somme ou le programme produit selon le choix d'utilisateur.