Correction du contrôle de système d'exploitation FST Settat 2015-2016

La solution a été proposé par un étudiant, si vous avez trouvé des fautes ou si vous avez des remarques merci de me contacter.

Exercice 1 :

a) Pour créer un « pipe » on crée une table de deux cases et on la passe à la fonction **pipe(int tab[2])**
b) Il suffit d'appeler la fonction fork() pour créer un nouveau processus, il est préférable de stocker le retour de la fonction dans une variable afin de pouvoir distinguer le processus fils du processus père (sinon on peut aussi utiliser getpid() et getppid() pour faire cela)
c) Ce qu'on doit faire ici c'est de remplacer le stdout(Standard output, c.-à-d. le flux de sortie standard) par l'entrée de notre pipe, c'est-à-dire au lieu d'écrire à l'écran on va écrire dans notre pipe, pour que l'autre processus puisse le lire, pour ce faire on va utiliser la fonction dup2(df1,df) (df veut dire descripteur de fichier) comme il est demandé.
d) Ici, le deuxième processus doit récupérer ce que l'autre processus a écrit dans le pipe, pour ce faire, on aura besoin de remplacer le stdin (standard input, le flux d'entrée standard) par la sortie de notre pipe.

Le code final :

```
#include  <unistd.h>
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int p[2];
    pipe(p);
    if (fork() == 0) {
        close(p[0]);/* la bonne pratique de fermer la sortie de la pipe avant d'utiliser
son entrée et vice versa */
        dup2(p[1],STDOUT_FILENO); /* stdin devient entrée tube */
        execl("/bin/ps","ps",NULL);
    } else {
        close(p[1]);
        dup2(p[0],STDIN_FILENO); /* stdout devient sortie tube */
        execl("/usr/bin/wc","wc", "-l", NULL);
    }
}
```

Remarques :

-La fonction execl reçoit en paramètre le chemin vers la commande (utiliser which nom_commande pour connaitre l'emplacement du script d'une commande), puis en deuxième paramètre le nom de la commande, et puis les arguments s'il y en et enfin la valeur NULL.

-La fermeture de la pipe n'est pas obligatoire mais c'est une bonne pratique.

Exercice 2 –

1-

- Oui
- Les opérations se font dans l'ordre suivant : b puis e puis h
    - b : x=3+1 = 4
    - e : x=4*2 = 8
    - f : x=8-4 = 4

2- Non, car P2 ne peut pas exécuter l'instruction (e) avant que P1 libère le mutex (instruction c) parce que c'est ce dernier (P1) qui l'a verrouillé.

3- il y a 3 ! = 6 valeur finale possible (pas forcement différent), un peu de probabilité …

Dénombrons-les :

P1->P2->P3 - P1->P3->P2
P2->P1->P3 - P2->P3->P1

P3->P1->P2 - P3->P2->P1

A vous de faire le calcul … Vous trouverez que les valeurs finales possibles sont (3,4,0,-1)

4-En faisant le calcul, on trouvera que les deux scénarios :

(1) P2->P1->P3
(2) P2->P3->P1

Donnent le résultat voulu, donc on va modifier les mutex pour obtenir l'un de ces deux scénarios :

On choisit le scénario 1 : P2 s'exécute en premier, puis P1 et enfin P3 (donc on choisit le scénario (1)) :

Soit 3 mutex :

m_p2 <-1

m_p1 <-0

m_p3<-0

P2 :	V(m_p1)	(a)
P(m_p2)		(b)
(d)		(c)
(e)	P1 :	V(m_p3)
(f)	P(m_p1)	

P3 :

P(m_p3)

(g)

(h)

(i)

V(m_p2)

Exercice 3 :

a) La taille de l'adresse virtuelle est de 32 bits, avec ce nombre de bits on peut adresser 2^32 adresses différentes.
b) google it.
c) On sait que le taille de page de système est 4Ko = 4*2^10 = 2^12 octets, donc pour adresser tous les octets d'une page (l'offset ou le déplacement) on a besoin d'une adresse de 12bits, et on sait que les adresses virtuelles sont codées sur 32 bits, donc il reste 20 bits pour adresser les pages (numéro de page), ces 20 bits peuvent adresser 2^20 adresse différentes, si chacune de ces adresses fait 4 octets, la table prendra une taille de 4*2^20 = 2^22 octets.