TP N°02 : Mécanismes de Base d'Exécution des Programmes

L'Objectif de ce TP est d'observer les processus Linux dans tous leurs états et de créer des processus simples.

Remarque : Les programmes des exercices 2.5, 2.6 et 2.7 sont implémentés et exécutés dans un environnement Windows et à l'aide du compilateur Turbo C.

Exercice 2.1 (Gestion des utilisateurs)

Q1) Créez deux groupes group1 et group2 ?
Q2) Créez quatre utilisateurs user1, user2, user3 et user4 ?
Q3) Rendez les utilisateurs dans les groupes :
- Les premier et deuxième utilisateurs sont membres du premier groupe.
- Les troisième et quatrième utilisateurs sont membres du second groupe.
- Le deuxième utilisateur est aussi membre du second groupe.
- Le quatrième utilisateur est aussi membre du premier groupe.

Q4) Vérifier les membres du groupe group2 ?
Q5) Créer deux répertoires rep1, rep2 et rep3 en seul ligne ?
Q6) Créer dans rep1 un fichier nommé fich11 et dans rep2 un répertoire nommé rep21
Q7) Déplacez-vous au répertoire rep21
Q8) Copiez le rep1 et son contenu dans le répertoire courant?
Q9) Copiez l'arbre rep2 dans le répertoire rp3 ?
Q10) Visualisez le contenu de rep3 de façon détaillée ?
Q11) Supprimez l'arbre rep3 ?

ex01.c

```
#include <stdio.h>
int main (void) {
printf ("Hello world !");
return 0;
}
```

Exercice 2.2 (Environnement de compilation : gcc)

Q1) Ecrivez le programme **ex01.c** en utilisant l'éditeur **gedit**.
Q2) Compilez le programme en utilisant la commande : **make ex01**
Quelle est votre remarque ?
Q3) Utilisez le compilateur **gcc** avec la commande : **gcc ex01.c -o ex01**
Quelle est votre remarque ?
Q4) Exécutez votre programme en tapant : **./ex01**
A. Supprimer tous les fichiers résolut dans la partie précédente
Q5) Compilez le module source en appelant le compilateur C par : **gcc –c ex01.c**
Q6) Quel est le nom du module objet obtenu et sa taille ?
Q7) Faites l'édition de liens du module objet (les bibliothèques utiles pour cet exemple sont ajoutées automatiquement) : **gcc nom_du_module_objet**
Q8) Quel est le nom du programme exécutable et sa taille?
Q9) Exécutez le module exécutable ? **./nom_du_module_exécutable**
B. Supprimer tous les fichiers résolut dans la partie précédente (il est possible de choisir le nom des modules.)
Q10) Compilez **ex01.c** et créez le fichier objet **ex01.o** : **gcc -o ex01.o -c ex01.c**
Q11) Construisez le programme exécutable **ex01** avec le fichier objet **ex01.o** :
gcc -o ex01 ex01.o
Q12) exécutez votre programme en tapant : **./ex01**

Exercice 2.3 (Compilation & édition de liens)

Supposons que le fichier « cal_moy.c » contient un programme principal écrit en langage « C » (un programme qui contient la fonction « **main()** »). Il consiste à calculer et afficher la moyenne des nombres réels lus au clavier. Les calculs des moyennes s'effectuent à l'aide des fonctions suivantes :

- La fonction « moy2(arg1,arg2) » calcule et retourne la moyenne des réels « arg1 » et « arg2 ».
- La fonction « moy3(arg1, arg2, arg3) » calcule et retourne la moyenne des réels « arg1 », « arg2 » et « arg3 ».

Q1) On suppose que les fonctions « moy2() » et « moy3() » sont définies dans le fichier « cal_moy.c » et qu'elles sont écrites après la fonction « main() ».

a) Ecrire le programme « cal_moy.c » et ensuite le compiler et l'exécuter.

b) Afficher la taille du fichier exécutable.

c) Recompiler avec l'option « -static ». Afficher la taille du fichier exécutable.

d) Conclure.

Q2) Maintenant, on suppose que les fonctions « moy2() » et « moy3() » sont définies dans des fichiers séparés : la fonction « moy2() » est définie dans un fichier nommé « moyenne2.c » et « moy3() » est définie dans un autre fichier nommé « moyenne3.c ».

a) Modifier le programme « cal_moy.c » ensuite le compiler et l'exécuter.

Q3) Ecrire un fichier nommé « utile.h » qui contient les prototypes des fonction « moy2() » et « moy3() » ensuite le sauvegarder dans le répertoire courant. Inclure ce fichier dans le programme principal et recompiler.

Q4) Recompiler le programme en utilisant la commande « make ».

Q5) Créer le répertoire « ~/include » et déplacer le fichier « utile.h » dans ce répertoire. Ensuite recompiler et exécuter le programme principal.

Q6) Créer une bibliothèque statique nommée « liboutils.a » et une bibliothèque dynamique nommée « liboutils.so » qui contiennent les fichiers « moyenne2.o » et « moyenne3.o ». Sauvegarder ces bibliothèques dans le répertoire « ~/lib »
Compiler et exécuter le programme « cal_moy.c » en utilisant ces bibliothèques.

Exercice 2.4 (Visualisation des processus)

Pour voir quels processus tournent sur une machine à un moment donné, il faut utiliser la commande ps.

Q1) Ouvrir deux terminaux. Dans le premier terminal, lancer 2 applications, par exemple **firefox** et **gedit** à l'aide des commandes **firefox &** et **xemacs &.**
Dans le deuxième terminal, tapez la commande **ps**.

a) Que se passe-t-il ? Pourquoi firefox et gedit n'apparaissent-ils pas dans la liste ?

b) Quelle option utiliser avec **ps** pour les voir ?

Q2) Utilisez la commande **ps** pour déterminer le **PID** (Process ID) du **firefox** que vous avez lancé.
Tapez **kill -9 lepiddefirefox**.

a) Que se passe-t-il ?

b) Déterminez le PID d'une des commandes bash et arrêtez-la à l'aide de la commande kill -9. Pourquoi la fenêtre du terminal disparaît-elle ?

Q3) Tapez **firefox** dans le premier terminal.

a) Pouvez-vous exécuter d'autres commandes dans ce terminal ? Pourquoi ? Faites un Ctrl-C. Quel processus a été tué ?

Rappel (Interruptions)

- La déclaration d'une interruption en langage C se fait comme suit :
  void interrupt sous_programme(){…}
- L'utilisation du mot clé **« interrupt »** fait que cette fonction C se terminera par l'instruction **IRET**.
- La déclaration d'un pointeur vers une routine d'interruption est
  void interrupt (*pointeur)();
- « **getvect** » permet de retourner l'adresse du vecteur d'interruption spécifié et utilise le numéro de vecteur d'interruption comme paramètre.
- « **setvect** » permet de fixer l'adresse du vecteur d'interruption spécifié et passe deux paramètres, le numéro de vecteur d'interruption et la fonction correspondant au sous-programme de traitement de cette interruption.
- Toutes les fonctions C utilisées ici sont déclarées dans la bibliothèque **<dos.h>**

Exercice 2.5 (Table des vecteurs d'interruption)

A l'aide de la fonction **getvect(),** écrire un court programme C qui affiche la table des vecteurs d'interruption

N.B : Pour afficher une adresse en hexadécimal, nous utiliserons **printf()** de la façon suivante : **printf("%lxH", (long)adr);**

Le tableau sera présente comme suit :

Numero INT	Adresse traitant
0	A04F7001
1	….
….	….

Exercice 2.6 (Détournement d'une interruption)

L'interruption numéro 0 (zéro) est appelée automatiquement par le processeur lui-même lorsque l'on tente d'effectuer la division d'un nombre entier par 0 (instruction IDIV, **opération /** en langage C).

Modifier le traitant associé afin d'afficher un message d'erreur de votre choix. On veillera à appeler le traitant existant après l'affichage du message. D'autre part, on restaurera l'ancien vecteur à la fin du programme.

Exercice 2.7 (Interruption d'horloge)

Ecrire un programme en langage C qui affiche l'horloge.

En utilisant la notion d'interruption d'horloge et l'instruction **outportb(0x20, 0x20).**

N.B :

- L'interruption d'horloge est déclenchée 18 fois par seconde.
- L'instruction **outportb(0x20, 0x20)** est utilisée pour réactiver le microcontrôleur Contrôleur d'interface périphérique (PIC : Peripheral Interface Controller)