Université Hassan II DE Casablanca
Faculté des sciences AIN Chock
Département des mathématiques et informatique

Module : Système d'exploitation II

TP 1 : Les processus

Manipulation des commandes Linux (Rappels) :

Dans cette partie nous proposons la manipulation des commandes les plus utilisées pour la gestion de fichier et le parcours des répertoires sous linux.

Question 1 : Utilisation du manuel des commandes.

man commande : exemple man ls

La commande man permet de lister l'aide pour une commande on peut aussi utiliser la commande :

ls - -help ou man ls

Question 2 : Accéder à l'éditeur de commande et placer vous à la racine.

a) Afficher l'arborescence.

cd /
ls -l

b) Citer le contenu de chaque dossier sous la racine « / ».

La racine inclut des répertoires prédéfinis de linux qui décrit l'organisation des données dans le système de fichier tel que :
Home : les dossiers personnels des utilisateurs
Etc : les fichiers de configuration
Bin : les exécutables
Dev : Contient des fichiers de périphériques qui représentent et permettent d'accéder au matériel et aux périphériques du système.

Question 3 : Création des répertoires et des fichiers

a) Accéder le répertoire home et créer le dossier TP1.

cd /home
mkdir TP1 ➔ *permission denied*
➔ *il va vous afficher permissions non accordés car le home est le répertoire personnel de tout les utilisateurs. Vous avez le droit de créer des fichiers dans votre répertoire personnel, généralement le répertoire personnel a le même nom de l'utilisateur connecté.*

cd /home/user
mkdir TP1

b) Sous ce dossier créer deux dossiers dont les noms sont « CODE » et « EXE ».

cd TP1

mkdir CODE
mkdir EXE

c) Afficher le contenu du dossier TP1. Mettez le résultat de la commande dans un fichier intitulé « fich1 » sous le répertoire TP1.

ls /home/user/TP1 > fich1

d) Afficher le contenu du répertoire home avec les informations de chaque fichier. Mettez le résultat « fich1 » sans écraser le contenu.

ls –l /home >> fich1

e) Afficher le contenu de « fich1 » via la commande « cat ». Comment éditer un fichier ?

cat fich1
gedit fich1

ou :

vim fich1

Partie II : Visualisation de la table des processus :

Dans cet exercice nous voulons visualiser les tables des processus sous linux. Pour cela on commence par la création des processus par un programme dédié écris en langage C.

a) Créer un répertoire dans le répertoire personnel, intitulé « TP ». Accéder à TP.

cd
mkdir TP
cd TP

b) Créer un fichier « prog.c ». Editer « prog.c » pour taper le programme suivant :

```c
#include <unistd.h>
int main(){
    printf ("Je suis le processus père\n") ;
    printf ("Création du processus fils\n") ;
    int p = fork() ;
    if(p==0) {
        int f=getpid() ;
        printf ("je suis le processus fils\n") ;
        printf ("le PID  du fils est =%d \n",f) ;
    }
    else {
        if (p<0) {
            printf ("erreur de création \n") ;
        } else  {
            printf ("je suis le processus père\n") ;
        }
    }
    return 0;
}
```

vim prog.c

ou :

gedit prog.c

c) Afficher la table des processus. Expliquer chaque attribut de la table. Chercher les attributs du processus « init ».

pf -ef

La table comporte plusieurs informations tel que :

UID : User ID l'identifiant de l'utilisateur propriétaire du processus
GID : Group ID l'identifiant du groupe du propriétaire du processus.
PID : Process ID un numéro unique qui identifie chaque processus.
PPID : Process ID du processus père du processus courant.
CMD : La commande qui a lancé le processus

d) Exécuter et compiler le programme prog.c

gcc prog.c -o prog
./prog

Exercice 3 : Attributs des Processus :

Dans cette partie nous traiterons les attributs des processus sous linux. Pour cela nous utiliserons les primitifs systèmes de récupérations des attributs :

getpid ()	identité du processus en cours
getppid ()	identité du processus parent de celui en cours
getuid ()	propriétaire réel : en général celui du login
geteuid ()	propriétaire effectif : propriétaire du processus pour lequel le bit set-uid a été modifié pour permettre à un exécutable d'être exécuter par un autre utilisateur. set-uid est le bit à changer pour changer d'utilisateur en utilisant la fonction setuid
getgid ()	groupe réel
getegid ()	groupe propriétaire effectif
getpwd ()	répertoire de travail

- Sous le dossier home associé à la session courante, créez un répertoire intitulé TP2.

cd ~
mkdir TP2

- Ecrire un programme C « prog1.c » sous TP2, qui permet de créer un processus fils.

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

int main(){

  printf ("Je suis le processus père\n") ;
  printf ("Création du processus fils\n") ;
  int p = fork() ;
  if(p==0) {
    int f=getpid() ;
    printf ("je suis le processus fils\n") ;
    printf ("le PID  du fils est =%d \n",f) ;
  } else {
    if (p<0) {
      printf ("erreur de création \n") ;
    }
    else {
      printf ("je suis le processus père\n") ;
    }
  }

  return 0;
}
```

- Modifier « prog1.c » afin d'afficher les attributs de chaque processus (père et fils), le **pid**, **ppid**, le **uid, guid** et le répertoire de travail. Pour la récupération du répertoire en utilise la syntaxe suivante : printf(" répertoire de travail : %s\n ",getcwd(buf,1024)), avec buf est une variable de type chaine de caractère.

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

int main(){

  char buf[1024];
  // Informations sur le processus pere
  printf ("le PID  du pere est =%d \n",getpid());
  printf ("le PPID  du pere est =%d \n",getppid()) ;
  printf ("le UID  du pere est =%d \n",getuid());
  printf ("le GUID  du pere est =%d \n",getgid());
  getcwd(buf, 1024);
  printf ("le rep de travail du pere est =%s \n", buf);
```

```c
  int p = fork() ;
  if(p==0) {
    int f=getpid() ;
    printf ("je suis le processus fils\n") ;
    // Informations sur le processus fils
    printf ("le PID  du fils est =%d \n",getpid());
    printf ("le PPID  du fils est =%d \n",getppid()) ;
    printf ("le UID  du fils est =%d \n",getuid());
    printf ("le GUID  du fils est =%d \n",getgid());
    getcwd(buf, 1024);
    printf ("le rep de travail du fils est =%s \n", buf);
  } else {
    if (p<0) {
      printf ("erreur de création \n") ;
    }
    else {
      printf ("je suis le processus pere\n") ;
    }
  }
  return 0;
}
```

```
le PID  du pere est =5312
le PPID  du pere est =26839
le UID  du pere est =1059
le GUID  du pere est =1060
le rep de travail du pere est =/home/joksolutions/TP1
je suis le processus pere
je suis le processus fils
le PID  du fils est =5313
le PPID  du fils est =5312
le UID  du fils est =1059
le GUID  du fils est =1060
le rep de travail du fils est =/home/joksolutions/TP1
```

- Quelles sont les attributs différents entre les deux processus. Expliquer.

Le processus fils et père ont des PID et PPID différents.

PID (Process Identifier) : Chaque processus a un identifiant unique. Le processus fils reçoit un PID différent de celui de tout autre processus sur le système, y compris le processus père.

PPID (Parent Process Identifier) : Le PPID du processus fils est le PID du processus père. Le PPID du processus père n'est pas affecté par la création du fils.

Exercice 4 :

Qu'affiche l'exécution du programme suivant :

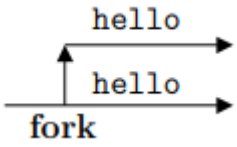
```
int main() {
        int pid;
        int x = 1;
        pid = fork();
        if (pid == 0) {
                printf("Dans fils : x=%d\n", ++x);
                exit(0);
        }
        printf("Dans père : x=%d\n", --x);
        return 0 ;
}
```

Exercice 5 :

Donner le schéma d'exécution des programmes suivant :

Exemple du schéma d'exécution :

```
int main() {
        fork();
        printf("hello!\n");
        exit(0);
}
```



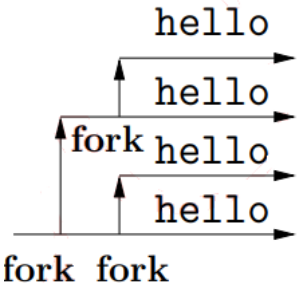
Programme 1:

```
int main() {
        fork();
        fork();
        printf("hello!\n");
        exit(0);
}
```

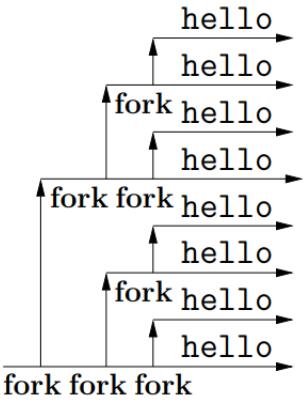
Programme 2:

```
int main() {
        fork();
        fork();
        fork();
        printf("hello!\n");
        exit(0);
}
```

Le schéma du programme 1 :



Le schéma du programme 1 :



Exercice 6 :

Combien de message « hello ! » affichera chaque programme ?

PROGRAMME 1:

```c
#include <unistd.h>
#include <stdio.h>
int main(){
    int i;
    for(i=0; i<2; i++)
        fork();
    printf("Hello\n");
    return 0;
}
```

PROGRAMME 2:

```c
#include <unistd.h>
#include <stdio.h>
void doit(){
    fork();
    fork();
    printf("Hello\n");
}
int main(){
    doit();
    printf("Hello\n");
    return 0;
}
```

PROGRAMME 3:

```c
#include <unistd.h>
#include <stdio.h>
int main(){
    if(fork())
        fork();
    printf("Hello\n");
    return 0;
}
```

PROGRAMME 4:

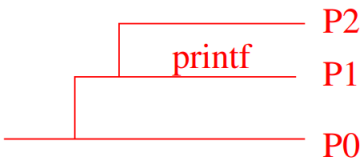
```c
#include <unistd.h>
#include <stdio.h>
int main(){
    if(fork()==0) {
        if(fork()){
            printf("Hello\n");
        }
    }
    return 0;
}
```

Réponse 1 : 4 fois

Réponse 2 : 8 fois. On crée 4 processus, et chacun écrit deux fois hello (une fois dans doit(), une fois dans main())

Réponse 3 : 3 fois. Car fork() renvoi non-nul (ie, vrai) dans le père seulement. Le père se clone donc à nouveau, tandis que les fils non.

Réponse 4 : Une seule fois. Car le père lance un fils (P1) et c'est tout (car fork ligne 2 renvoie != dans le père). Son fils P0 fait un fork supplémentaire ligne 3 qui crée P2. Ce fork renvoie == 0 dans P3 et != 0 dans P2. Donc, seul P2 entre dans le corps du if et exécute le printf. En résumé, ça crée trois processus, mais ça n'affiche qu'une seule fois la ligne.



Annexe : Tableau des commandes

Commande	Description
cd	Accéder à un dossier
ls	Afficher le contenu d'un répertoire
man	Afficher l'aide pour une commande ou l'ensemble des commandes
touch	Création d'un fichier
mkdir	Création d'un répertoire
gcc	Compilation d'un programme C

cat	Visualiser un fichier
diff	afficher les différences entre 2 fichiers.
ps –ef	Afficher la table des processus