TP 1

3 séances de TP en salle machine

Objectifs :

1°) Se familiariser avec les outils de base sous Linux :
   a) Ecrire un fichier **Makefile** et utiliser la commande " **make** ",
   b) Compiler avec le compilateur " **gcc** " (GNU Compiler Collection) en utilisant les options, et enfin exécuter le programme, fournir des résultats et mettre le tout en page sous linux.

2°) Utiliser les appels système sous linux (**open**, **opendir**, **read**, **readdir**, **stat**, **fstat**, …) en utilisant le manuel en ligne : " **xman** ", " **man** " ou " **info** " ou les icônes du bandeau (ces commandes sont auto-documentées : ***man man***) et se familiariser avec le système de fichiers *ext3fs*.

1) Le fichier Makefile et la commande make :

Création du fichier Makefile (sans aucune extension !!) avec votre éditeur préféré : gedit, vim, emacs, ...

La structure de base du Makefile est :
cible : dépendances
[tab] commandes

- **cible** est le nom du fichier créé (par exemple, monProg) ;
- **dépendances** représente la liste des fichiers (ou règles) nécessaires à la construction de la cible ;
- **commandes** représente les commandes à effectuer pour créer la cible.

Notez bien que la cible n'est construite que si le fichier source est plus récent.

La structure de base définie en haut est appelée une règle, et le fichier Makefile n'est rien de plus qu'un ensemble de règles.

Prenons un petit **exemple** :
Nous avons 3 fichiers main.c, afficherMessage.c et afficherMessage.h .

```
/*afficherMessage.c*/
#include <stdio.h>
#include "afficherMessage.h"

void afficherMessage(char * message) {
        printf("\n %s ", message);  }
```

```
/* afficherMessage.h */
#include <stdio.h>
void afficherMessage(char * message);
```

```
/*Programme principal main.c*/
#include "afficherMessage.h"

int main() {
        afficherMessage("Bonjour tout le monde");
  }
```

Les commandes à exécuter pour compiler manuellement seront :
gcc -c afficherMessage.c -o afficherMessage.o
gcc -c main.c -o main.o
gcc -o monProg main.o afficherMessage.o

Le **Makefile** sera alors structuré de la manière suivante :

```
# un exemple de Makefile

monProg : main.o afficherMessage.o
[tab] gcc -o monProg main.o afficherMessage.o

main.o : main.c afficherMessage.c
[tab] gcc -c main.c -o main.o

afficherMessage.o : afficherMessage.c
[tab] gcc -c afficherMessage.c -o afficherMessage.o
```

La commande make exécute les instructions dans le Makefile.
make sans arguments exécute la première règle du Makefile. Dans le cas de cet exemple, elle exécute monProg : …

2) Cette deuxième partie du TP consiste à étudier un programme qui permet de gérer les éléments du système de fichiers ext3fs (linux). En d'autres termes, le programme permet d'utiliser des primitives de gestion des fichiers/répertoires et d'accéder et d'afficher des attributs d'un fichier/un répertoire. En plus, ce programme effectue certaines actions à identifier et bien comprendre.
-- Le but est de modifier ces actions, sous linux, en langage C, au moyen des outils disponibles sous linux, afin de lui faire effectuer des actions complémentaires avec l'aide du chargé de TP.

Vous prendrez donc le programme source en C ***ls-exemple.c*** qui se trouve sous le répertoire ***H:\COMMUN\DUT 2ème année\Systeme\JE\TP***, et vous le copierez dans un répertoire que vous créerez à cet effet dans votre domaine de travail. Ensuite vous en étudierez le texte avant de le compiler et de tester son fonctionnement. Par exemple, pour l'utilisateur Dupont :

```
[Dupont@up5iut2 Dupont]$ gcc ls-exemple.c -o montest # compilation du programme
ls-exemple.c et génération du fichier exécutable montest.

[Dupont@up5iut2 Dupont]$ ./montest # ou export PATH=.:$PATH ; montest
[Dupont@up5iut2 Dupont]$ ./montest montest
[Dupont@up5iut2 Dupont]$ ./montest toto

[Dupont@up5iut2 Dupont]$ > fichier1
[Dupont@up5iut2 Dupont]$ ./montest fichier1

[Dupont@up5iut2 Dupont]$ ./montest ..
[Dupont@up5iut2 Dupont]$ ./montest /bin
```
etc.

Une fois cette prise en main effectuée, vous étudierez en profondeur les fonctions ‘‘***stat***’’ et ‘‘***fstat***’’ ➔ *man stat* et *man fstat*

a) Vous modifierez le programme source (*ls-exemple.c*) pour lui faire afficher la date de dernier accès en clair.
Pour afficher la date en clair, c'est à dire pour convertir en un format à la fois compréhensible et affichable par l'utilisateur, pensez à utiliser la commande ‘‘apropos’’ avec le mot clef ***date*** de la manière suivante :
[Dupont@up5iut2 Dupont]$ apropos date | grep " (3) " # (3) pour se limiter aux fonctions de la librairie standard.

b) Vous modifierez *ls-exemple.c* pour pouvoir afficher le nombre de références existant sur l'i-node du programme ; pour cela regardez la structure ***stat*** présentée dans le document PDF ‘‘ Structure_STAT.pdf ’’.

c) Vous modifierez *ls-exemple.c* pour pouvoir afficher le **nom du fichier** dont vous avez le **numéro d'inode**.

Il va sans dire, mais encore mieux en le disant, que vous devez, au bout de ces séances, être capable de compiler par la ligne de commande, créer un Makefile avec un éditeur de texte, créer ou modifier un programme avec un éditeur de texte sous linux, utiliser le manuel, rechercher la commande ou la fonction le mieux adaptée à vos besoins, mettre au propre le source du programme commenté et ses résultats.