TD n° 1 Initiation aux outils de développement C sous Linux

Objectifs

Le but de ce TD est de :

- S'initier à l'utilisation de quelques outils de développement avec le langage C sous Linux.
- Développer en C sous Linux à partir d'un terminal (en mode console).

NB: Il est à noter de même que l'utilisation de tout IDE tel que codeblocks par exemple est strictement interdite.

Prérequis

Vous devriez disposer des machines virtuelles Linux

Intervenants

Enseignante: Amamou Lamis

Outils de développement

gcc: (GNU Compiler Collection) est une suite de logiciels libres de compilation.
Utilisé dans le monde Linux lorsqu'on veut transcrire du code source en langage machine, c'est le plus répandu des compilateurs.
La suite gère le C et ses dérivés mais aussi le Java ou encore le Fortran.

make: (Le gestionnaire de compilation make) permet d'automatiser la phase de compilation, celle de votre programme mais aussi celle de la documentation associée, et permet aussi d'automatiser l'installation, la mise en paquetage pour distribuer plus facilement votre application, l'élimination de fichiers temporaires créés par votre éditeur de texte, le compilateur, etc...

gdb: GDB est l'acronyme de Gnu DeBugger. C'est un debugger puissant dont l'interface est totalement en ligne de commande, c'est à dire avec une invite en texte. GDB est tellement apprécié qu'on le trouve aussi encapsulé dans des interfaces graphiques, comme XXGDB ou DDD. GDB est publié sous la licence GNU et gratuit.

Préparation de l'environnement

Lancez la machine virtuelle Linux.

1. Lancez un terminal (combinaison de touches: **ctrl** + **alt** + **t**)
2. Vérifiez l'existence des outils gcc, make et gdb dans votre système d'exploitation:

gcc - - version
make - - version
gdb - - version

3. S'ils n'existent pas : Installez les outils gcc, make et gdb en tapant les commandes suivantes:

sudo apt-get update && sudo apt-get install build-essential gdb

En cas de succès d'installation, les commandes afficheront les versions des outils gcc, make et gdb installés dans votre système d'exploitation Linux.

Développement C Sous Linux

Le développement de programmes C sous linux à partir d'un terminal passe par des étapes:

1. phase d'édition ou d'écriture du code
2. phase de compilation du code afin de créer un binaire (exécutable sur la machine)
3. phase d'exécution du binaire et débogage en cas d'erreur de segmentation.

Phase d'Édition de code source

La première phase de développement avec n'importe quel langage est l'édition du code ou rédaction du code moyennant un éditeur de texte.
Dans un terminal (en mode console) une panoplie d'éditeurs existe. L'éditeur le plus facile à utiliser est **nano**.
En mode graphique, il existe un éditeur préinstallé dont le nom est **gedit.**
NB: si l'éditeur nano n'est pas installé, tapez dans le terminal la commande suivante: **sudo apt-get**

install nano

Dans un terminal: Créez un répertoire ***tdcompilation*** dans lequel seront créés tous les fichiers.

Éditez un fichier main.c avec la commande suivante: ***nano main.c***

Ensuite, écrivez le code suivant:

```
#include<stdio.h>
int main() {
        int a;
        int b;

        printf("\nSaisie du premier Nombre :");
        scanf("%d", &a);
        printf("\nsaisie du second Nombre:");
        scanf("%d", &b);

        printf("\n%d + %d = %d", a, b, a+b);
        printf("\n%d - %d = %d", a, b, a-b);
        printf("\n%d / %d = %f\n", a, b, (float)a/b);
        return (o);
}
```

a) Phase de Compilation du code source

Dans le même terminal, tapez la commande qui permet de compiler le fichier main.c:

gcc main.c -o programme

main.c : le fichier source

programme : le binaire (exécutable) résultant de l'opération de compilation.

b) Phase d'Exécution du binaire

Afin de lancer l'exécution du binaire issu de la compilation du code source main.c, juste tapez dans le même terminal la commande**: ./programme**

c) Compilation de plusieurs fichiers sources

Dans le même terminal actif, vous disposez à présent du fichier main.c.

Créez deux nouveaux fichiers nommés fonctions.c et fonctions.h:

fonctions.c :

```
int sommer(int a, int b) {
        return (a + b);
}
int soustraire(int a, int b) {
        return (a - b);
}
float diviser(int a, int b) {
        return ((float) a / b);
}
```

Éditez le fichier ***fonctions.h*** et copiez dedans les prototypes des fonctions implémentées dans fonctions.c

```
#ifndef FONCTIONS_H_
#define FONCTIONS_H_
int sommer(int a, int b);
int soustraire(int a, int b);
float diviser(int a, int b);
#endif /* FONCTIONS_H_ */
```

NB: n'oubliez pas d'ajouter #include "fontions.h" dans main.c et d'évoquer les fonctions de ce module

d) Compilation manuelle sous linux

Compilez chaque module à part :

gcc -c main.c

gcc -c fonctions.c

NB: L'option -c permet de compiler le fichier en question sans passer à la phase d'édition de liens (the -c option says not to run the linker [voir man gcc]).

Tapez la commande «**ls**» afin de lister le contenu du répertoire courant. Vous allez remarquer que deux fichiers fonctions.o et main.o (des objects files), résultat de la phase de compilation, sont créés.

La dernière phase après celle de compilation est de lier les objets files en un seul binaire exécutable (prog) moyennant l'option **-o**

gcc main.o fonctions.o -o prog

Enfin vous pouvez exécuter le binaire résultant, en tapant la commande:

./prog

e) Automatisation de la compilation sous linux

Afin d'automatiser la compilation de plusieurs fichiers sources, nous allons utiliser l'outil **make.**
Make est un outil très général permettant, entre autre, d'automatiser la compilation d'un projet.
Il suffit de décrire ces relations à **Make**, ainsi que les commandes associées pour qu'il puisse produire la séquence correcte de commandes.

f) Création du fichier Makefile

Le fichier **Makefile** est celui dans lequel on met les commandes qui seront exécutées par l'outil **make**. Ce fichier est constitué des descriptions des relations entre les fichiers sources et les fichiers objets ainsi que des relations entre les fichiers objets et l'exécutable (des **règles**).Une règle est constituée des composantes suivantes:

cible: dépendances
<tabulation>commande

Créez un fichier intitulé **Makefile**
Éditez les règles suivantes dans le fichier Makefile

```
prog:fonctions.o main.o
	gcc fonctions.o main.o -o prog
main.o:main.c
	gcc -c main.c
fonctions.o:fonctions.c
	gcc -c fonctions.c
```

Compilez le projet en invoquant l'outil **make** : ***make***
Exécutez la cible **prog** (le binaire résultant)**: ./prog**

g) Exécution et débogage

Éditez à nouveau le fichier **Makefile** contenant les règles de compilation.
Ajoutez l'option **-g** à chaque fois où vous évoquez la commande **gcc**, enregistrez le fichier **Makefile**, ensuite quittez**.**

```
prog:fonctions.o main.o
	gcc fonctions.o main.o -o prog -g
main.o:main.c
	gcc -c main.c -g
fonctions.o:fonctions.c
	gcc -c fonctions.c -g
```

Rééditez le fichier **main.c** et remplacez le code précédent par ce code qui présente un bug lors de l'exécution (Le message: **erreur de segmentation** s'affiche) (On a enlevé le passage par adresse dans scanf)

```
#include<stdio.h>
#include "fonctions.h"
int main() {
	int a;
	int b;

	printf("\n Saisie du premier Nombre :");
	scanf("%d", a);
	printf("\n Saisie du second Nombre:");
	scanf("%d", b);

	printf("\n%d + %d = %d", a, b, sommer(a, b));
	printf("\n%d + %d = %d", a, b, soustraire(a, b));
	printf("\n%d / %d = %f\n", a, b, diviser(a, b));
	return (0);
}
```

Recompilez à nouveau le projet : ***make*** (La compilation va se dérouler sans problèmes)
Exécutez le binaire résultant **prog : ./*prog***
Lors de l'exécution, un message s'affiche: "**Erreur de segmentation**" sans aucune indication sur la nature et l'emplacement de l'erreur, puisque nous sommes en phase d'exécution du binaire **prog.**
La seule façon de trouver les erreurs en cours d'exécution est de déboguer le binaire **prog** grâce au débogueur **gdb**.
Lancez le débogueur gdb: **gdb prog**
Une fois l'interface du **gdb** est lancée, tapez dans le prompt du **gdb** la commande '**run**' ou tapez '**r**'. Des

informations sur la cause de l'erreur s'affichent mais ce n'est pas encore précis, on peut en conclure que l'utilisation de la fonction **scanf** a causé le problème.
On peut mieux faire en localisant la ligne causant l'erreur dans le code. Il suffit de taper la commande **where** dans le prompt du gdb