TP 0 : Préparation de l'environnement de programmation système

1-1- Outils de développement

La programmation système en C sous Linux principalement ces utilitaires :

- ***L'éditeur de texte*****,** qui permet de créer et de modifier le fichier source.
  Vi,Vim, Emacs…..
- ***Le compilateur***, gcc (Gnu C Compiler)

GCC est le nom générique de la suite d'outils de développement contenant, entre autres, le compilateur C/C++ GNU développé par la Free Software Foundation. C'est en fait un front-end qui permet d'utiliser les programmes de développement. Ce front-end est très souple et peut être utilisé pour les différentes phases de production des binaires : ***preprocessing des fichiers sources, compilation, assemblage et édition de liens. Il dispose d'un grand nombre d'options, et peut être paramétré à l'aide d'un fichier de configuration***

- ***Le débogueur*** gdb (Gnu Debugger)

GDB permet l'exécution pas à pas du code, l'examen des variables internes, etc. Pour cela, il a besoin du fichier exécutable et du code source.

1-2- Installation et utilisation

- **Installation**

La plupart de ces outils sont installés par défaut sur votre distribution linux. si ce n'est pas le ou si vous voulez changer de version , il faut procéder à leur installation par le fameux : ***apt-get install ….***
Il est conseillé d'installer le paquet ***build-essential***, car il contient la plupart de ces utilitaires (compilations, éditions de liens, débogage….).

- **Compilation**

Dans un terminal taper la commande suivante

```
gcc fichier.c -o binaire
```

En remplaçant bien sûr **fichier.c** par le fichier source d'entrée et **binaire** par le fichier de sortie souhaité. À défaut de paramètre *output file* (-o binaire), gcc créera **a.out.**

Remarque *: À noter qu'il faut se trouver dans le répertoire où se situe le* ***fichier.c***

- **Exécution**

Pour exécuter le fichier binaire généré par le GCC, on exécute dans le terminal.

```
./binaire
```

Si vous avez des erreurs essayez de modifier le "droit d'accès" du ficher avec cette commande :

```
sudo chmod 770 binaire
```

1-4- les options de gcc

Option	Signification
--help	Affiche l'aide de GCC.
--version	Donne la version de GCC.
-E	Appelle le préprocesseur. N'effectue pas la compilation.
-S	Appelle le préprocesseur et effectue la compilation. N'effectue pas l'assemblage ni l'édition de lien. Seuls les fichiers assembleur (« .S ») sont générés.
-c	Appelle le préprocesseur, effectue la compilation et l'assemblage, mais ne fait pas l'édition de lien. Seuls les fichiers objets (« .o ») sont générés.
-o nom	Fixe le nom du fichier objet généré lors de la compilation d'un fichier source.
-g	Génère les informations symboliques de débogage.
-fexceptions	Active la gestion des exceptions C++.
-fpic	Génère du code relogeable. Cette option est nécessaire pour la compilation des fichiers utilisés dans une DLL ou un fichier chargeable dynamiquement.
-On	Indique le niveau d'optimisation (n peut prendre les valeurs allant de 0 à 3, ou « s » pour optimiser la taille des binaires).
-mcpu=cpu	Indique le type de processeur pour lequel le code doit être optimisé. Le code fonctionnera sur tous les processeurs de la famille de ce processeur.
-march=cpu	Indique le type de processeur pour lequel le code doit être généré. Le code généré sera spécifique à ce processeur, et ne fonctionnera peut-être pas sur un autre modèle de la même famille. Cette option active automatiquement l'option -mcpu avec le même processeur.
-pipe	Utilise les pipes systèmes au lieu des fichiers temporaires pour les communications entre le préprocesseur, le compilateur et l'assembleur.
-w	Supprime tous les warnings.
-W	Active les warnings supplémentaires.
-Wall	Active tous les warnings possibles.
-mwindows	Crée un exécutable GUI Windows.
-mdll	Crée une DLL Windows.
-fvtable-thunks	Utilise le mécanisme des tables de fonctions virtuelles. Cette option est nécessaire pour utiliser les interfaces COM sous Windows.

1-5- Les commandes de l'éditeur vi , vim

voir le lien

http://wiki.linux-france.org/wiki/Utilisation_de_vi

TP 1 : Argument d'un programme et variable d'environnement

Nous allons voir dans ce TP les passages d'arguments et variables d'environnement qui permettent à un shell de transmettre des informations à un programme qu'il lance. Plus généralement, ces techniques permettent à un programme de transmettre des informations aux programmes qu'il lance (processus fils ou descendants).

1-1- atoi , sprinf, sscanf

Parfois, un nombre nous est donné sous forme de chaîne de caractère dont les caractères sont des chiffres. Dans ce cas, la fonction **atoi** permet de réaliser la conversion d'une **chaîne** vers un **int**.

```
#include <stdio.h>

int main()
{
   int a;
   char s[50];

   printf("Saisissez des chiffres : ");
   scanf("%s", s);  /*saisie d'une chaîne de caractères */
   a = atoi(s);  /* conversion en entier */
   printf("Vous avez saisi : %d\n", a);
   return 0;
}
```

Plus généralement, la fonction **sscanf** permet de lire des données formatées dans une chaîne de caractère (de même que scanf permet de lire des données formatées au clavier ou **fscanf** dans un fichier texte).

```
#include <stdio.h>

int main()
{
   float x;
   char s[50];

   printf("Saisissez des chiffres (avec un point au milieu) : ");
   scanf("%s", s);  /* saisie d'une chaîne de caractères */
   sscanf(s, "%f", &x);  /* lecture dans la chaîne */
   printf("Vous avez saisi : %f\n", x);
   return 0;
}
```

Inversement, la fonction **sprintf** permet d'écrire des données formatées dans une chaîne de caractères (de même que **printf** permet d'écrire dans la console ou **fprintf** dans un fichier texte).

```
#include <stdio.h>

void AfficheMessage(char *message)
{
    puts(message);
}

int main ()
{
    float x;
    int a;

    printf("Saisissez un entier et un réel : ");
    scanf("%d %f", &a, &x) ;
    sprintf(s, "Vous avez tapé : a = %d x = %f", a, x) ;
    AfficheMessage(s);
    return 0;
}
```

1-2- Arguments du main

La fonction **main** d'un programme peut prendre des arguments en ligne de commande. Par exemple, si un fichier **monprog.c** a permis de générer un exécutable **monprog** à la compilation,

```
gcc monprog.c -o monprog
```

On peut invoquer le programme monprog avec des arguments

```
./monprog argument1 argment2 argument3
```

Pour récupérer les arguments dans le programme *C*, on utilise les paramètres **argc** et **argv** du main.

- L'entier **argc** donne le nombre d'arguments rentrés dans la ligne de commande **plus 1**
- le paramètre argv est un tableau de chaînes de caractères qui contient comme éléments :
    - Le premier élément **argv[0]** est une chaîne qui contient le nom du fichier exécutable du programme ;

- Les éléments suivants **argv[1], argv[2],** etc... sont des chaînes de caractères qui contiennent les arguments passés en ligne de commande.

Le prototype de la fonction mainest donc :

```
int main(int argc, char**argv);
```

Exemple. *Voici un programme longueurs, qui prend en argument des mots, et a che la longueur de ces mots.*

```
#include <stdio.h>
#include <string.h>

int main(int argc, char**argv)
{
   int i;
   printf("Vous avez entré %d mots\n", argc-1);
   puts("Leurs longueurs sont :");
   for (i=1 ; i<argc ; i++)
      {
         printf("%s : %d\n", argv[i], strlen(argv[i]));
      }
   return 0;
}
```

Voici un exemple de trace :

```
$ gcc longueur.c -o longueur
$ ./longueur toto blabla
 Vous avez entré 2 mots
Leurs longueurs sont :
toto : 4
blabla : 6
```

1-3- Les variables d'environnement

Les variables d'environnement sont des affectations de la forme

NOM=VALEUR
qui sont disponibles pour tous les processus du système, y compris les shells. Dans un shell, on peut avoir la liste des variables d'environnement par la commande **env**. Par exemple, pour la variable d'environnement **PATH** qui contient la liste des répertoires où le shell va chercher les commandes exécutables :

```
$ echo $PATH
/usr/local/bin:/usr/bin:/bin:/usr/b
in/X11
$ PATH=PATH:.
$ export PATH
$ env | grep PATH
PATH=/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:.
```

La commande export permet de transmettre la valeur d'une variable d'environnement aux descendants d'un shell (programmes lancés à partir du shell).

1-3-1- Accéder aux variables d'environnement en C

Dans un programme C, on peut accéder à la liste des variables d'environnement dans la variable **environ**, qui est un tableau de chaînes de caractères (terminé par un pointeur NULL pour marquer la fin de la liste).

```
#include<stdio.h>

extern char **environ;

int main(void)
{
   int i;
   for (i=0 ; environ[i]!=NULL ; i++)
      puts(environ[i]);
   return 0;
}
```

Pour accéder à une variable d'environnement particulière à partir de son nom, on utilise la fonction **getenv**, qui prend en paramètre le nom de la variable et qui retourne sa valeur sous forme de chaîne de caractère.

```
#include <stdio.h>
#include <stdlib.h> /* pour utiliser getenv */

int main(void)
{
   char *valeur;
   valeur = getenv("PATH");
   if (valeur != NULL)
      printf("Le PATH vaut : %s\n", valeur);
   valeur = getenv("HOME");
   if (valeur != NULL)

      printf("Le home directory est dans %s\n", valeur);
   return 0;
}
```

Pour assigner une variable d'environnement, on utilise la fonction putenv, qui prend en paramètre une chaîne de caractère. Notons que la modification de la variable ne vaut que pour le programme lui-même et ses descendants (autres programmes lancés par le programme), et ne se transmet pas au shell (ou autre) qui a lancé le programme en cours.

```
#include <stdio.h>
#include <stdlib.h> /* pour utiliser getenv */

int main(void)
{
   char *path, *home, *nouveaupath;
   char assignation[150];
   path = getenv("PATH");
   home = getenv("HOME");
   printf("ancien PATH : %s\n et HOME : %s\n", path, home);
   sprintf(assignation, "PATH=%s:%s/bin", path, home);
   putenv(assignation);
   nouveaupath = getenv("PATH");
   printf("nouveau PATH : \n%s\n", nouveaupath);
   return 0;
}
```

Exemple de trace :

```
$ gcc putenv.c -o putenv
$ echo $PATH
$ /usr/local/bin:/usr/bin:/bin:/usr/bin/X11
$ ./putenv
ancien PATH :
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11 et HOME
: /home/remy
nouveau PATH :
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11:/home/re
 my/bin echo PATH
/usr/local/bin:/usr/bin:/bin:/usr/bin/X11
```