

HPC/BigData - Compilation Avancée (CA 2017-2018)



TD1

Rappels sur gcc, Make et gdb Introduction aux passes de compilation

julien.jaeger@cea.fr
patrick.carribault@cea.fr

Installation de GCC 8.1

Durant toute la durée de ce module, nous allons utiliser et étendre GCC 8.1. Pour ce faire, il est possible d'utiliser une version du compilateur déjà installée par la distribution Linux ou alors de le compiler à partir des sources. Pour savoir quelle méthode utilisée, il faut d'abord tester si la version de GCC installée sur le système est bien la bonne version (8.1) et supporte l'ajout de nouveaux plugins. Pour vérifier la version du compilateur, il faut lancer la commande suivante :

```
$ gcc --version  
gcc (GCC) 8.1.0  
...
```

Si la réponse de cette commande est bien GCC 8.1.0 (comme sur l'exemple ci-dessus), il faut alors vérifier que le compilateur a bien été construit avec le support des plugins grâce à la commande suivante :

```
$ gcc -print-file-name=plugin  
/home/login/...
```

Cette commande doit afficher un répertoire. Si cette dernière affiche bien un répertoire existant (et lisible) et si la version de GCC est bien 8.1.0, le TD peut être commencé. Sinon, il est nécessaire d'installer GCC 8.1.0 à partir des sources. En raison du temps requis pour réaliser cette installation, nous allons lancer celle-ci avant de commencer les exercices du TD.

Pour installer les sources de GCC 8.1.0, vous devez avoir accès à un environnement de travail sous LINUX. Cela est possible de deux façons : soit vous avez la possibilité d'installer une partition LINUX sur votre machine, soit vous devez en installer une sur une machine virtuelle (par exemple VirtualBox).

Avant d'installer la version 8.1.0 de GCC, vérifiez que vous avez accès à une version antérieure sur votre machine en tapant la commande `gcc -v`. Une version de GCC sera nécessaire pour les premiers exercices de ce TP.

Pour procéder à l'installation de GCC 8.1.0, il vous faut récupérer les sources. Le lien pour récupérer les sources est disponible sur [. Si vous avez des difficultés à accéder aux sources, une clé USB avec les sources sera mis à disposition.](#)

Avant de compiler GCC 8.1.0, certains packages sont nécessaires. Si ce n'est pas déjà fait, installer les packages *libgmp-dev*, *libmpc-dev* et *libmpfr-dev*.

Une fois que vous avez récupéré l'archive, décompressez-la. Dans le répertoire des sources, créez deux dossiers MYBUILD et MYINSTALL. Allez dans le répertoire MYBUILD, puis tapez la commande suivante :

```
../configure --prefix=/home/login/Bureau/GCC/gcc-8.1.0/MYINSTALL
--enable-languages=c,c++,fortran --enable-plugin --disable-bootstrap
--disable-multilib
```

GCC 8.1.0 devrait s'installer. Pendant l'installation, nous allons faire les premières parties du TP.

I Compiler un programme avec GCC

Q.1: Compilez sans optimisation (sans flag `-O` ou avec `-O0`) le programme situé dans le répertoire *VECTOR*. Exécutez le programme. Que constatez-vous sur l'affichage des valeurs du vecteur *V3* ?

Correction

```
gcc main.c vector.c compute.c -o vector
./vector
*****
HPC/BigData - module CA - TD1 : part1 gcc
Add two vectors and store values in a third one
Vectors size : 32
Repeat 30000000 times
*****
V1 + V2 = V3
-----
1 + 2 = 1.77864e-322
1 + 2 = 1.77864e-322
1 + 2 = 1.77864e-322
1 + 2 = 1.77864e-322
...
Erreur d'affichage de V3
```

Q.2: Rajoutez le flag `-Wall` et corrigez tous les avertissements émis par le compilateur (warnings).

Correction

<i>timer.h</i>		
timer.h :15 : warning : implicit declaration of function 'gettimeofday'		
	Avec warnings	Sans warning
1 : 4	#include <time.h>	#include <sys/time.h>
<i>compute.c</i>		
compute.c :31 : warning : format '%g' expects type 'double', but argument 4 has type 'unsigned int'		
	Avec warnings	Sans warning
1 : 30	printf("\t%d\t\t%d\t\t%g\n",	printf("\t%d\t\t%d\t\t%d\n",

Correction

<i>main.c</i>		
main.c :31 : warning : unused variable 'useless'		
main.c :33 : warning : control reaches end of non-void function		
	Avec warnings	Sans warning
1 : 31	float useless = 0;	
1 : 32		return 0;

Q.3: Exécutez le programme corrigé et relevez le temps affiché.

Correction

```
gcc main.c vector.c compute.c -o vector
./vector
...
TIME ELAPSED : 4.57 s
(Intel Core2 Duo E6750 @ 2.66GHz, gcc 4.4.5)
```

Q.4: Compilez avec le flag -O3, exécutez et relevez à nouveau le temps. Que constatez-vous ?

Correction

```
gcc -O3 main.c vector.c compute.c -o vector
./vector
...
TIME ELAPSED : 0.54 s
(Intel Core2 Duo E6750 @ 2.66GHz, gcc 4.4.5)
```

Q.5: On souhaite maintenant créer une bibliothèque dynamique avec les fonctions contenues dans le fichier *vector.c*. Pour ce faire, utilisez les flags *-shared* et *-fPIC* pour créer *libvector.so*.

Correction

```
gcc -O3 -shared -fPIC vector.c -o libvector.so
```

Q.6: Compilez le programme en le liant à la bibliothèque et exécutez le programme.

Correction

```
gcc -O3 main.c compute.c -L. -lvector
```

Rappels :

- Ne pas oublier de mettre à jour la variable d'environnement `LD_LIBRARY_PATH` depuis le fichier `.bashrc` ou en utilisant la commande :
`export LD_LIBRARY_PATH=chemin_de_la_lib:$LD_LIBRARY_PATH`
(Attention, ces manipulations peuvent être différentes en fonction du type de shell)
- Vous pouvez vérifier la liste des bibliothèques dynamiques liées à un exécutable en utilisant la commande suivante :
`ldd nom_de_l_executable`

II Utilisation de Make

Q.1: Écrivez un Makefile pour le code de la section précédente (avec la bibliothèque dynamique), en écrivant toutes les règles explicitement (sans utiliser de variable personnalisée ni de variable propre à Make).

Correction

```
all: vector

vector: compute.o main.o
    gcc -O3 -Wall -L. -lvector main.o compute.o -o vector

main.o: main.c
    gcc -O3 -Wall -c main.c -o main.o

compute.o: compute.c
    gcc -O3 -Wall -c compute.c -o compute.o

clean:
    rm -rf *.o
```

Q.2: Améliorez le Makefile en employant cette fois-ci des variables personnalisées (`CC`, `CFLAGS`, `LDFLAGS`, `EXEC`) et variables internes à Make (`%`, `^`, `@`, `<`).

Correction

```
CC=gcc
CFLAGS=-O3 -Wall
LDFLAGS=-L. -lvector
EXEC=vector

all: $(EXEC)

$(EXEC): compute.o main.o
    $(CC) $(CFLAGS) $(LDFLAGS) ^ -o $@

%.o: %.c
    $(CC) $(CFLAGS) -c $< -o $@

clean:
    rm -rf *.o
```

III Prise en main de gdb

Étudiez le programme dans le répertoire *BUGS*, puis compilez le avec le flag *-g*. Exécutez le programme compilé avec gdb :

```
gdb nom_de_l_executable
```

gdb dispose d'une aide interactive. Commencez par parcourir le menu d'aide en saisissant d'abord *help* pour avoir la liste des commandes. Puis, *help* suivi d'une commande pour obtenir des informations sur celle-ci.

Q.1: Afin de repérer la source du premier bug, tapez *run* sous gdb. Quittez gdb (*quit*), corrigez l'erreur et recompilez le programme.

Correction

```
Arithmetic exception(division par 0)
Supprimez memset(&nb,0,sizeof(unsigned int));
```

Q.2: Procédez de la même manière pour corriger le bug suivant (i.e. *gdb executable*, puis *run*). Utilisez la commande *backtrace* (ou *bt*) pour afficher la pile des appels de fonctions et obtenir plus d'informations sur la source de l'erreur.

Correction

```
#0 0x0000000004005ba in factorial (val=Cannot access memory at address 0x7ffff3feffc) at main.c :32
#1 0x0000000004005de in factorial (val=0) at main.c :37
Dans le if, remplacez val = 0 par val == 0
```

Q.3: Identifiez la prochaine erreur après avoir recompilez le programme. Quel est le problème et peut on le corriger ?

Correction

```
#0 0x0000000004005ba in factorial (val=Cannot access memory at address 0x7ffff3feffc) at main.c :32
#1 0x0000000004005d7 in factorial (val=4294574295) at main.c :37
Appels récursifs infinis, la pile explose. Il faut empêcher de passer un nombre négatif à la fonction factorial.
```

Q.4: Nous allons maintenant résoudre le dernier bug avec d'autres fonctionnalités élémentaires de gdb. Démarrez gdb sans utiliser la commande *run* pour le moment. Fixez un point d'arrêt sur la fonction *launch_fibonacci* (*breakpoint launch_fibonacci* ou *b launch_fibonacci*). Utilisez ensuite la commande *run*. Le programme va s'arrêter lors de la première entrée dans la fonction *launch_fibonacci*. Essayez maintenant d'accéder à la valeur *fibonacci_values* → *max* avec la commande *print fibonacci_values* → *max* que constatez-vous ? Saisissez maintenant *up* pour se placer avant l'appel de la fonction puis *list* pour afficher les lignes de code autour du point d'arrêt. Entrez maintenant la commande *print fibonacci_values*. Corrigez le problème et recompilez le programme.

Correction

```
Cannot access memory at address 0xc,  
$1 = (Fibo *) 0x0  
Il faut allouer la structure de la variable fibo_values.  
Fibo *fibo_values = malloc(sizeof(Fibo));
```

Q.5: La suite est incorrecte. Nous devrions obtenir les nombres suivants :
 $\mathcal{F}_0 = 0, \mathcal{F}_1 = 1, \mathcal{F}_2 = 1, \mathcal{F}_3 = 2, \mathcal{F}_4 = 3, \mathcal{F}_5 = 5, \mathcal{F}_6 = 8, \dots$
Pour repérer d'où vient l'erreur, nous allons afficher pas par pas les valeurs de la suite en surveillant les modifications de la variables *fibonacci_values* → *result*.
Démarrez gdb et saisissez les commandes suivantes :

```
b main  
run  
watch fibonacci_values->result
```

Entrez ensuite *continue* (ou *c*) pour avancer pas à pas à chaque modification de *fibonacci_values* → *result*. Trouvez où l'erreur se situe.

Correction

```
La première valeur est fausse.  
Remplacez fibonacci_values->n_minus_2=1; par  
fibonacci_values->n_minus_2=0;
```

IV Introduction au projet : les passes de compilation de gcc

Q.1: 'Etudiez le programme dans le répertoire *SAXPY*, puis compilez-le avec l'option *-fdump-tree-all*. Utilisez la commande *ls*. Qu'observez-vous ?

Correction

```
Des fichiers sont apparus.
```

Q.2: A quoi cela correspond-t-il ? Combien en observez-vous ?

Correction

```
Ces fichiers correspondent à chaque passe effectuée par le compilateur lors de la compilation du fichier main.c. 17 fichiers ont été créés.
```

Q.3: Effacez les fichiers qui sont apparus, et compilez à nouveau le fichier, en ajoutant le flag *-OI*. Qu'observez-vous ? Y a-t-il des fichiers manquants par rapport à la précédente compilation ?

Correction

```
Beaucoup plus de fichiers ont été créés. Ils sont au nombre de 84 désormais. Deux fichiers sont absents de cette compilation : veclower et cplxlower0.
```

Q.4: Dans les sources de gcc, trouvez le fichier *passes.def*. Que contient-il ?

Correction

```
Le fichier passes.def se situe dans le répertoire $(GCC491_HOME)/gcc.
```

Q.5: Trouvez la définition des passes *lower_complex_O0* et *lower_vector*.

Correction

Les fichiers **tree-complex.c :1722** (voir aussi l.1616) et **tree-vect-generic.c :1672** (voir aussi l.1628).