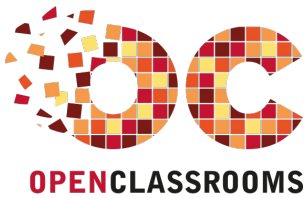
Compilation à la volée avec libtcc

Par Grota



www.openclassrooms.com

Dernière mise à jour le 7/04/2011

Sommaire

Compilation à la volée avec libtcc

Par  Grota

Mise à jour : 07/04/2011
Difficulté : Facile Durée d'étude : 20 minutes

Vous avez toujours voulu avoir **un compilateur dans votre programme** ? Vous voulez laisser à vos utilisateurs la possibilité de coder en C dans votre application, en exécutant le code **à chaud** ? La libtcc est faite pour vous ! Et en plus, ce n'est qu'une question de minute tellement c'est simple !

LibTCC n'est **pas** entièrement portable, mais est compatible posix (vous pourrez donc l'utiliser si vous disposez par exemple du compilateur GCC, ou mingw, qui sont fournis, par exemple, avec code::block). Cependant Visual studio permet de compiler la libtcc.

Le code présenté dans ce tutoriel n'est pas vraiment robuste : pour des raisons de clarté, certaines vérifications ont été omises (notamment, les retours de malloc)

Sommaire du tutoriel :

TCC et libtcc

Un peu d'histoire

Tout d'abord, laissez-moi présenter ce qui est à l'origine de libtcc, le compilateur TCC, écrit par Fabrice Bellard (Qui a notamment créé QEMU et qui est recordman du nombre de décimales de pi...). TCC, c'est un compilateur extrêmement léger (environ 100ko), qui gère le C sauf certaines fonctionnalités avancées (*trigraphe* et *proper type* notamment). Jusque-là, rien de bien enchanteur. Sauf que ce logiciel compile neuf fois plus vite que GCC. Oui, vous avez bien lu : **neuf fois** !

De ce logiciel est né libtcc qui permet de compiler du code C directement dans une application et l'exécuter sans redémarrer le programme (à chaud).

Pour l'anecdote, tcc peut être utilisé comme interpréteur C, qui fonctionne de la même manière que les scripts sh : mettez la ligne `#!/bin/tcc -run` en début de fichier, puis placez votre programme C, rendez le fichier exécutable et il sera compilé puis exécuté comme un script shell normal.

Installation de TCC et libtcc

Sous les UNIX-like

Pour commencer, il vous faut les sources de tcc, disponible sur le site officiel, section **download**. Il faut bien choisir le code source et pas la version exécutable.
Extrayez l'archive dans un répertoire, puis, dans un terminal, déplacez-vous dans celui-ci. Compilez et installez ensuite le

programme. Ainsi, vous aurez à exécuter les commandes suivantes :

Code : Console

```
tar xjf tcc-0.9.25.tar.bz2
./configure
make
make install # avec les droits root
```

Sous windows

Pour commencer, il vous faut les sources de tcc, disponible sur le site officiel, section **download**. Il faut bien choisir le code source et pas la version exécutable.

Il existe 2 méthodes pour utiliser libtcc sous windows : utiliser visual studio ou mingw.

Avec visual studio : Compilez le fichier libtcc.c, disponible à la racine des sources, en une bibliothèque .lib. Vous pourrez l'utiliser ultérieurement en lieu et place de libtcc.a dans visual studio

Avec mingw : Dans le dossier "win32" de l'archive, vous trouverez un script batch "build-tcc.bat" qu'il suffit d'exécuter une fois (double clic) pour compiler les exécutables ainsi que la librairie elle-même (un nouveau dossier "libtcc" est créé, et il contient les fichiers "libtcc.h" et "libtcc.a"). Notez que la compilation avec MinGW sous windows peut causer des problème lors de l'exécution (Je recommande donc plutôt d'utiliser la méthode précédente).

Configurer votre projet pour utiliser libtcc

Si vous utilisez directement GCC, vous n'aurez qu'à utiliser la ligne `gcc votrefichiersource.c -ldl -ltcc` : en plus de lier avec libtcc (libtcc.a), il est nécessaire de lier avec libdl (Librairie pour charger dynamiquement du code exécutable, utilisée par exemple lors de l'utilisation de bibliothèque dynamique comme les .dll ou .so).

Avec un IDE, spécifiez que l'éditeur de lien doit utiliser votre fichier libtcc.a, et incluez libtcc.h dans votre projet.

Les utilisateurs de Visual Studio utiliseront la librairie qu'ils ont compilé à la place de libtcc.a .

Compilons !

Je vais donner un exemple tout fait que je commenterais petit à petit. Il s'agit d'un programme qui affichera « Hello World ! (32)» (Comme c'est original…), mais avec quelques subtilités.

Code : C

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

#include <libtcc.h>

char my_program[] =
        "int fonction(int n) "
        "{"
        " printf(\"Hello World! (%d)\\n\",n);"
        " return 0; "
        "}";

int main(int argc, char **argv)
{
        TCCState *s;

        int (*entry)(int);
        void *mem;
        int size;
```

```
        s = tcc_new();
        if (!s) {
                fprintf(stderr, "Impossible de creer un contexte
TCC\n");
                exit(1);
        }

        tcc_set_output_type(s, TCC_OUTPUT_MEMORY);

        if (tcc_compile_string(s, my_program) != 0) {
                printf("Erreur de compilation !\n");
                return 1;
        }

        size = tcc_relocate(s, NULL);
        if (size == -1)
                return 1;

        mem = malloc(size);
        tcc_relocate(s, mem);

        entry = tcc_get_symbol(s, "fonction");

        tcc_delete(s);

        entry(32);

        free(mem);
        return 0;
}
```

Commentaires:

Code : C

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

#include <libtcc.h>
```

Très classique, à ceci près que l'on inclut *libtcc.h*

Code : C

```
char my_program[] =
        "int fonction(int n)"
        "{"
        " printf(\"Hello World! (%d)\\n\",n);"
        " return 0;"
        "}";
```

Dans ce tableau de char est contenue une fonction qui sera compilée et exécutée. Bien sûr, vous n'êtes pas obligé de faire ainsi, vous pouvez récupérer directement l'entrée utilisateur, ou un champ texte de votre application graphique. L'important est de l'avoir disponible sous la forme d'un tableau de char.
Notez cependant ici que j'ai dû échapper (placer un antislash devant) les double quote (\") de façon à éviter de fermer malencontreusement la chaine de caractère. De la même manière, j'ai échappé le \n en \\n pour qu'il ne soit pas interprété en un retour à la ligne lors de la compilation.

Code : C

```
TCCState *s;
/// … ///
s = tcc_new();
if (!s)
{
        fprintf(stderr, "Impossible de creer un context TCC\n");
        exit(1);
}
```

Création d'un contexte TCC, ce qui se fait par création d'un pointeur, puis acquisition du contexte via un appel à **tcc_new** Ce contexte identifie notre compilation. Ainsi, en faisant plusieurs contextes, on peut compiler plusieurs choses en même temps. Sachez aussi que quasiment toutes les fonctions de libtcc prennent pour premier argument un contexte TCC.

Un contexte TCC est une sorte de variable, dont on ne connait pas le contenu et qui stocke toutes les informations nécessaires à une compilation avec libtcc.

Code : C

```
tcc_set_output_type(s, TCC_OUTPUT_MEMORY);
```

Cette ligne est importante. Elle indique à libtcc sous quelle forme nous allons récupérer la sortie de la compilation. Dans notre cas, nous allons l'exécuter, il est donc nécessaire de stocker le code exécutable en mémoire (TCC_OUTPUT_MEMORY). Remarquez que l'on passe comme premier argument le contexte TCC créé précédemment.

Code : C

```
if (tcc_compile_string(s, my_program) != 0)
{
        printf("Erreur de compilation !\n");
        return 1;
}
```

On compile le programme avec l'instruction `tcc_compile_string(s, my_program)` , puis, si la fonction renvoie autre chose que 0, c'est qu'il y a eu une erreur de compilation. Dommage  . Les arguments sont : le contexte TCC et le tableau de char contenant le programme.

Code : C

```
size = tcc_relocate(s, NULL);
if (size == -1)
        return 1;
```

Cette instruction est plus compliquée. tcc_relocate sert à copier notre résultat de compilation depuis le contexte (virtuellement opaque à nos yeux) vers un endroit dans la mémoire que l'on contrôle ; malheureusement, nous ne savons pas combien ce code exécutable prend comme place, nous allons donc le copier une fois à vide (on l'envoie vers NULL, qui n'est pas valable), mais on obtient en retour le nombre d'octet copié, c'est-à-dire la taille du code compilé.

Code : C

```
mem = malloc(size);
tcc_relocate(s, mem);
```

Cette étape est très logiquement l'allocation de la mémoire disponible (déterminée à l'étape précédente et via *malloc*), et la copie réelle du code exécutable vers cette adresse (via *tcc_relocate*).

Code : C

```
entry = tcc_get_symbol(s, "fonction");
```

Comme vous avez pu le constater, dans notre code compilé n'était pas présent de fonction main. Et pour cause, elle n'est pas nécessaire ! Nous allons obtenir l'adresse de notre fonction via *tcc_get_symbol*. Comme vous l'avez deviné, il prend comme argument un contexte TCC et un nom de symbole (un symbole est une variable, une fonction,...). Dans notre cas, nous voulons l'adresse de la fonction **fonction**. Notez tout de même que nous aurions pu appeler cette fonction comme nous le voulions, par exemple **leGateauEstUnMensoge**, **trucbidule**, ou même **main** (ce n'est pas interdit).

Qu'est-ce que **entry** ? Ce truc mystérieux que vous avez vu en début de main() est un pointeur sur fonction, en l'occurrence un pointeur sur une fonction qui prend en argument un int et renvois un int. Pour vous faire mieux comprendre: voici un pointeur sur une fonction qui prend en argument 2 int et un char, et renvois un int: `int (*pointeur)(int, int, char);` . Les arguments des pointeurs doivent être les mêmes que la fonction ciblée ! Dans notre exemple, ils le sont, nous pouvons donc l'utiliser

Code : C

```
tcc_delete(s);
```

Ceci est la suppression du contexte TCC. Cela libère de la mémoire.

Code : C

```
entry(32);
```

Et le moment de vérité : exécution de la fonction ! Votre programma va afficher "Hello World ! (32)". Si vous aviez appelé la fonction avec 23 comme argument (`entry(23) ;`), elle aurait affiché "Hello world ! (23)". Les arguments passés au pointeur sont passés à la fonction.

Code : C

```
free(mem);
return 0;
```

Libération de la mémoire et fin du programme.

Exécution du programme

Lorsque vous exécuterez ce programme, vous obtiendrez la sortie suivante :

Code : Console

```
Hello world! (32)
```

Décortiquons le comportement du programme, en entrant dans les détails :

- Lancement du programme
- Acquisition d'un contexte TCC (les opérations réalisées ici nous sont inconnues)
- Compilation du programme contenu dans le tableau **my_program**
- Récupération de l'adresse de la fonction **fonction(int n)** dans le pointeur **entry**
- Exécution de cette fonction via le pointeur de fonction **entry**

Manipulation de symbole

Jusque-là, vous avez appris à faire compiler un morceau de code. Mais vous n'avez pas pu intervenir dessus. C'est là que vient la manipulation de symbole : vous pouvez rendre une fonction de votre programme accessible dans le morceau de code qui sera compilé et l'inverse sera aussi possible. Voyons ensemble les possibilités de la manipulation de symbole.

Dans ce chapitre, je ne donnerais plus de code complet, mais des exemples

tcc_add_symbol()

Soit l'instruction :

Code : C

```
tcc_add_symbol(s, "addition", add);
```

Cette instruction prend trois arguments : un contexte TCC, un nom de symbole et un symbole existant. Cela signifie que, dans le contexte de compilation **s**, la fonction **add** sera disponible sous le nom **addition**.

Vous pourrez, dans le code que vous compilerez, faire appel à la fonction **addition**, en utilisant le même prototype que la fonction **add**.
Si la fonction **add** est codée ainsi:

Code : C

```
int add(int a, int b){return a+b;}
```

Alors on pourra y faire appel depuis le programme allant être compilé via

Code : C

```
addition(1,2); // résultat 3
```

C'est assez utile si vous voulez créer une sorte de binding C avec votre programme.

tcc_get_symbol()

Cette instruction fait exactement l'inverse de la fonction précédente. Nous l'avons brièvement décrite durant le chapitre précédent.

Soit l'instruction :

Code : C

```
void (*func)(int entier);
func = tcc_get_symbol(s, "ma_fonction")
```

Dans le contexte **s**, l'adresse du symbole **ma_fonction**, qui doit être présent dans le code qui a été compilé (dans le code que l'on a fournit à **tcc_compile_string**, il doit y avoir une fonction nommée **ma_fonction**, qui accepte comme argument un entier) est récupérée dans le pointeur sur fonction **func**. Nous pouvons ensuite exécuter la fonction par un appel au pointeur de fonction, ainsi : `func(42);` , comme un appel de fonction normal.

Exemple

Reprenons le premier exemple, avec quelques ajouts :

Code : C

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

#include <libtcc.h>

char ma_fonction[] =
        "int soustraction(int a,int b) "
        "{"
        " return difference(a,b); "
        "}";


int soustraire(int a, int b)
{
        return a-b;
}

int main(int argc, char **argv)
{
        TCCState *s;

        int (*entry)(int,int);
        void *mem;
        int size;

        s = tcc_new();
        if (!s) {
                fprintf(stderr, "Impossible de creer un contexte
TCC\n");
                exit(1);
        }

tcc_add_symbol(s,"difference", soustraire);

        tcc_set_output_type(s, TCC_OUTPUT_MEMORY);

        if (tcc_compile_string(s, ma_fonction) != 0) {
                printf("Erreur de compilation !\n");
                return 1;
        }

        size = tcc_relocate(s, NULL);
        if (size == -1)
                return 1;

        mem = malloc(size);
        tcc_relocate(s, mem);

entry = tcc_get_symbol(s, "soustraction");

        tcc_delete(s);

printf("%d \n", entry(10, 3) );

        free(mem);
```

```
            return 0;
    }
```

A la ligne 33, nous exportons la fonction **soustraire**, sous le nom de **difference**, dans le contexte TCC **s**. Nous avons bien pris soin de placer cette ligne **avant** la compilation, pour éviter une erreur (si nous l'avions défini après, la compilation aurait échoué, car la fonction différence ne serait pas accessible).

A la ligne 49, nous avons obtenu un pointeur sur la fonction **soustraction** et nous avons affiché le résultat de la soustraction de 10 et 3 à la ligne 53.

Mise en garde

Vous devez toujours garder en tête, lors de la manipulation de symbole, et a fortiori lors de l'exécution du code entré par un utilisateur, que des fonctions peuvent entrer en collision : si votre programme originel contient une fonction nommée **ouvrirFichier** et que le code que vous compilez avec libtcc en contient aussi une, elles vont entrer en collision, et le comportement sera indéfini [1]. De la même manière, réfléchissez à deux fois avant d'utiliser l'entrée utilisateur : celui-ci est sournois (ou idiot) et pourra avec une grande facilité crasher votre application (un appel à exit() est si vite arrivé )

[1] En réalité, il est très répandu que la fonction la plus récente écrase celle d'origine, mais vous ne devez **pas** vous fier à ce comportement qui est possiblement lié à une implémentation de libdl !

[TP] Réalisation d'un compilateur

Vous avez les connaissances nécessaires pour réaliser un petit compilateur. Voici donc un TP.

Cahier des charges

- Compile un fichier C en un exécutable
- Support de l'option -l <nom de la librairie> qui lie le programme avec la librairie
- Support de l'option -L <chemin> qui permet de rechercher les librairies spécifiées avec -l dans le chemin <chemin>
- Un seul fichier à l'entrée (je suis de bonne humeur…)

Indices

Regardez dans libtcc.h pour une liste de tous les prototypes disponibles. Vous aurez notamment besoin de **tcc_add_library** et **tcc_add_library_path**. Vous n'êtes pas obligé d'utiliser **tcc_compile_string** (cela implique de charger le fichier de vos propres moyens), je vous conseille `tcc_add_file(TCCState *s, const char *filename)` qui prend pour paramètre un contexte TCC et une chaine de caractère contenant le chemin du fichier à compiler. Cette fonction peut être utilisée plusieurs fois pour intégrer plusieurs fichiers.

Et si vraiment vous séchez:

Secret (cliquez pour afficher)

Cachez ces petits papiers dans votre trousse

Code : C

```
tcc_set_output_type(s, TCC_OUTPUT_EXE);
```

et

Code : C

```
tcc_output_file(s,"mon_executable_qui_sera_genere");
```

Correction

Je vous donne les sources d'une des premières versions de mon compilateur **Irae**. Vous noterez l'utilisation de **getopt** pour gérer les options

Secret (cliquez pour afficher)

Code : C

```
/*
Copyright © 2010 Briand William
This program is free software: you can redistribute it and/or
modify
it under the terms of the GNU General Public License as published
by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public
License
along with this program. If not, see
<http://www.gnu.org/licenses/>.

*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <getopt.h>

#include <libtcc.h>

char copyright[] =  "Irae Copyright (C) 2010 Briand William\n"
                    "This program comes with ABSOLUTELY NO
WARRANTY;\n"
                    "This is free software, and you are welcome to
redistribute it\n"
                    "under certain conditions; You should have
received a"
                    " copy of the GNU General Public License along
with this "
                    "program. If not, see
<http://www.gnu.org/licenses/>.\n"
                    "\n"
                    "irae -o <output> -i <file> : Compile <file>
en <output>\n"
                    "-L <path> : ajoute <path> au chemin de
recherche des librairie\n"
                    "-l <lib> : Lie le programme avec la librairie
<lib>\n";

int main(int argc,char **argv)
{
        char *output;
        output = "a.out";
        char *input;
        input = NULL;
```

```
        TCCState *s;
        s = tcc_new();
        if (!s) {
                fprintf(stderr, "Erreur: impossible de créer un
contexte TCC\n");
                exit(1);
        }

        tcc_set_output_type(s, TCC_OUTPUT_EXE);

        extern char *optarg;
        extern int optind, opterr;
        int c;
        while ((c = getopt (argc, argv, "hi:l:L:o:")) != -1) {
                switch (c) {
                case 'o':
                        output = optarg;
                        break;
                case 'i':
                        input = optarg;
                case 'l':
                        tcc_add_library(s, optarg);
                        break;
                case 'L':
                        tcc_add_library_path(s, optarg);
                        break;
                case 'h':
                        printf(copyright);
                        return 1;
                        break;
                }
        }
        if (input == NULL) {
                printf("Fichier a l'entree manquant (-i
<fichier>)\n") ;
                return 1;
        }
        if ( tcc_add_file(s,input) != 0) {
                printf("Erreur de compilation. \n");
        }

        tcc_output_file(s,output);
        tcc_delete(s);
}
```

Je rappelle que la compilation s'effectue avec les option suivantes:

Code : Console

```
gcc fichierAcompiler.c -ltcc -ldl -o compileur
```

Plus...

Voilà, vous avez la base. Pour mieux connaitre la librairie, lisez **libtcc.h** qui contient tous les prototypes. Ou sinon, revenez sur ce tuto où j'expliquerais peut-être plus de fonction. Une idée d'amélioration pour ce TP serait d'afficher les erreurs de compilations (c'est peut-être plus compliqué que vous ne le pensez.)

Pour vous donner des idées sur comment utiliser libtcc, vous n'avez qu'à songer un peu: cela peut être utilisé dans :

- Un MMORTS de programmation (Je ne vise personne)
- Pour scripter votre programme, en C

- Faire un compilateur à votre sauce
- Faire un programme très modulaire, car qui pourrait compiler ses plugins lors de l'utilisation du programme
- Développez rapidement : au lieu de recompiler tout votre programme, codez un éditeur qui lancera votre programme directement sans passer par un compilateur externe
- ... les possibilités sont infinies !

Voilà, au plus vous avez pris 20 minutes pour lire ce tutoriel et vous êtes paré pour lancer tous vos projets les plus fous ! Moi aussi, quand j'ai découvert TCC, j'ai trouvé que les possibilités étaient gigantesques !

Si vous recherchez le même type de solution appliquée au C++, dirigez vous vers CINT