EPFL CS212 : travail de la semaine 02

J.-C. Chappelier & E. Bugnion EPFL

% Rev. 2021.01.20 / 1

Table des matières

Projet programmation système W02

Introduction

Cette semaine, nous souhaitons vous faire approfondir la compilation séparée (`Makefile`) et utiliser la troisième bibliothèque nécessaire à notre projet (libmongoose). Nous vous proposons pour cela deux exercices :

1. sur la base de `thumbify.c` de la semaine passée, écrire un embryon de webserver d'image(s), créé à partir de 2 fichiers sources plus une bibliothèque locale ;
2. écrire le `Makefile` d'un projet plus ambitieux déjà fourni.

Vous aurez à rendre le(s) code(s) correspondant(s) d'ici au dimanche 14 mars 23:59.

Matériel fourni

Pour récupérez les fichiers fournis pour cette semaine, il vous faut rejoindre le second devoir de ce cours en suivant ce lien : https://classroom.github.com/a/NQfwkbq4.

Vous devriez alors avoir accès à un nouveau dépôt du genre `pps21-week02-YOURGITHUBID.git`, qu'il vous faudra donc cloner sur votre machine :

```
git clone git@github.com:projprogsys-epfl/pps21-week02-YOURGITHUBID.git
```

Ce dépôt contient deux répertoires :

- un répertoire `done/myprj` qui sera le répertoire pour votre propre micro-projet de webserver (détails ci-dessous) ;
- un répertoire `done/bigprj` qui contient déjà un « gros » projet complet, dont vous n'aurez à écrire que le `Makefile`. Mais commencez au préalable par votre propre micro-projet, plus simple.

Pour tout ce qui concerne `git` et de son fonctionnement, n'hésitez pas à revoir cette page complémentaire (semaine passée).
Et pour tout ce qui relève de `make` et des `Makefile`, revoyez aussi cette documentation déjà fournie la semaine passée.
Nous insistons sur ce point (bien prendre le temps d'approfondir les `Makefiles`) car l'an passé plusieurs n'ont pas fait cette étape assez sérieusement et en ont pâti par la suite.

I. Votre propre projet

I.1 Réorganisation du code de la semaine passée

Commencez par aller dans le répertoire `done/myprj` et copiez y votre fichier `thumbify.c` :

```
cp ../../../pps21-week01-YOURID/done/thumbify.c .
```

Editez ensuite ce fichier et `webserver.c` pour reporter la partie pertinente du `main()` de `thumbify.c` vers celui de `webserver.c` à l'endroit indiqué, puis supprimer totalement ce `main()` (de `thumbify.c`).

Comme `thumbify` la semaine passée, `webserver` prendra une image comme argument. Dans le `main()`, `webserver.c` doit donc appeler `create_thumb()` (fourni par `thumbify.h` comme expliqué ci-dessous) pour créer la version réduite (« *thumbnail* ») de l'image reçue en argument.

Créer ensuite un fichier `thumbify.h` permettant d'exporter les deux fonctions `generate_thumb_name()` et `create_thumb()`.

Complétez enfin le `Makefile` pour pouvoir compiler et créer `webserver` à partir de `thumbify.o`, `webserver.o` et les bibliothèques nécessaires, dont la bibliothèque `libmongoose` fournie en local.
A noter : 1. que la compilation de la bibliothèque locale `libmongoose` (c.-à.-d. la création du fichier `libmongoose/libmongoose.so`) est déjà fournie (dans le `Makefile`); 2. cette partie (« *Complétez le `Makefile`...* » est courte à décrire, mais nécessite un vrai travail de votre part ; il est *normal* que vous ne sachiez pas faire cela « en 5 minutes » et deviez lire la documentation et chercher par vous-même ; c'est bien *tout le but* de cette partie (vous enseigner les `Makefile` par la pratique). 3. `webserver.c` n'est pas terminé et fait l'objet de la partie

ci-dessous; mais il devrait déjà compiler si vous lui fournissez les bonnes bibliothèques ; 4. lors de l'exécution d'un programme utilisant des bibliothèques non standard (comme p.ex. `webserver` utilisant la bibliothèque locale `libmongoose`), il faut *au préalable* indiquer à l'aide de la variable `LD_LIBRARY_PATH`, les endroits où se trouvent ces bibliothèques; par exemple (à ne faire qu'une seule fois par Shell) :

```
export LD_LIBRARY_PATH="${PWD}"/libmongoose
```

I.2 Complétion de la partie « webserver »

La bibliothèque fournie `libmongoose` est une bibliothèque très simple à utiliser qui implémente le protocole HTTP (comme par exemple `apache` ou `nginx`, qui sont des serveurs web plus robustes et avec plus de fonctionnalités, mais également beaucoup plus difficile à intégrer).
Sa documentation se trouve ici : https://github.com/cesanta/mongoose/tree/master/docs. Le but cette semaine n'est pas maintenant de tout lire/comprendre, mais simplement s'assurer que vous arrivez à l'utiliser.
Pour cela, vous n'aurez qu'à utilise la fonction `mg_http_serve_file()` dont la documentation se trouve ici.

Noter que vous n'avez pas à passer de « `extra_headers` », passez simplement `NULL` à la place. Et le type que nous utilisons est « `image/jpg` ».

Complétez `webserver.c` aux deux endroits demandés pour « servir » les images demandées.

Pour tester votre programme, lancez le comme suit :

```
./webserver papillon.jpg
```

(en ayant recopié l'image `papillon.jpg` depuis la semaine passée, ou en indiquant son chemin.)
Si un message du genre :

```
2021-02-02 21:17:37  I mongoose.c:2925:mg_listen 1 accepting on http://localhost:8000
```

s'affiche, c'est que le server Web tourne et vous pouvez alors aller consulter l'image d'origine à cet URL ci et l'image miniature à cet URL là.

II. Savoir rentrer dans un plus gros projet

Une des compétences qui vous sera aussi utile à l'avenir (hors de ce cours) est de savoir appréhender de gros projets déjà existants. Dans cet esprit, et aussi afin de vous faire pratiquer les `Makefile`, nous fournissons (dans `done/bigprj`) un projet C déjà existant, dont vous avez à écrire le `Makefile` (toujours dans `done/bigprj`).

REMARQUE IMPORTANTE : le code fourni ici est soumis à droit d'auteur et ne doit en aucun cas être recopié ailleurs, ni réutilisé en aucune façon.

(Il est par ailleurs criticable de plusieurs points de vue et n'est donc *pas du tout* fourni en tant que source d'inspiration. . .)
Il vous est fourni *uniquement* pour que vous écriviez un `Makefile` permettant de le compiler.

Le code fourni utilise des sous-répertoires et une fonction de C99 (`roundf()`).

Pour indiquer de rechercher des fichiers d'en-tête dans des sous-répertoires, il faut ajouter une option `-I` par sous-répertoire voulu. Par exemple, lors de la compilation de `machin.c`, pour demander au compilateur de rechercher un fichier d'en-tête dans le sous-répertoire `truc`, il faut faire :

```
gcc -c -I truc machin.c -o machin.o
```

(Voir aussi la variable `CFLAGS` dans la documentation sur `make` fournie la semaine passée.)

Pour compiler suivant la norme C99 (ou au dessus), passer l'option `-std=c99` (ou `-std=c11` ou `-std=c17` ou `-std=c2x`) au compilateur (voir la variable `CFLAGS`).

Enfin, pour pouvoir utiliser la fonction `roundf()` de C99, il faut faire l'édition de liens avec la bibliothèque mathématique en ajoutant `-lm`. (Voir aussi la variable `LDLIBS` dans la documentation sur `make` fournie la semaine passée.)

Notes :

1. Le code fourni compile avec moult « *warnings* ». Nous **ne** vous demandons **pas** de corriger ces erreurs ; simplement d'écrire un `Makefile` qui parvient à produire un exécutable `main`.
2. Il n'est pas nécessaire de lancer le `main` en question. Si vous l'avez fait, vous pouvez simplement le quitter en tapant `Ctrl-C`.
3. Pour avoir tous les points, votre `Makefile` devra non seulement produire un exécutable `main` lorsque l'on tape `make`, mais aussi *avoir toutes les dépendances correctement spécifiées.*

Feedback tests

Le feedback via Docker doit toujours être demandé depuis `done/` lui-même. Comme nous avons cette semaine des sous-répertoire, la façon de le faire est de lancer :

```
make -f myprj/Makefile feedback
```

(depuis `done/`, donc) ; c.-à-d. que l'on indique à `make` où aller chercher son `Makefile`.

Rendu

Pour rendre le devoir, « commitez » (`commit`) et « poussez » (`push`) toutes vos modifications avant le dimanche 14 mars 23:59. Pensez également à mettre

votre fichier `time.csv` tel que décrit dans le barème du cours. Le rendu se fera automatiquement en prenant la version enregistrée (`commit` + `push`) dans le répertoire `done` de votre dépôt GitHub (branche `master`) à cette date. Il est donc impératif que vous ayez `commit` puis `push` votre version finale avant cette date. Aucune autre solution ne sera acceptée.

Et n'oubliez pas de faire le rendu de la semaine passée avant ce dimanche soir.