TD0 : Makefile

J.-M Friedt

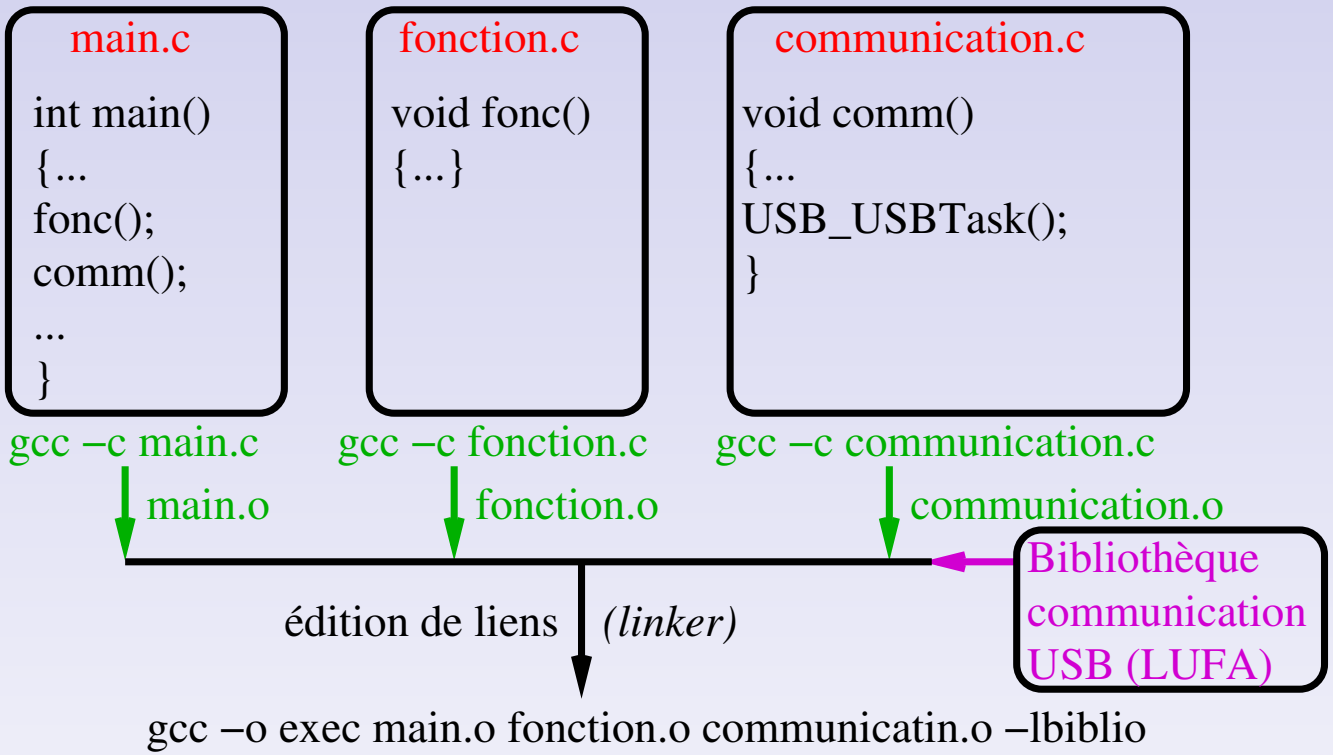
FEMTO-ST/département temps-fréquence

jmfriedt@femto-st.fr

transparents à `jmfriedt.free.fr`

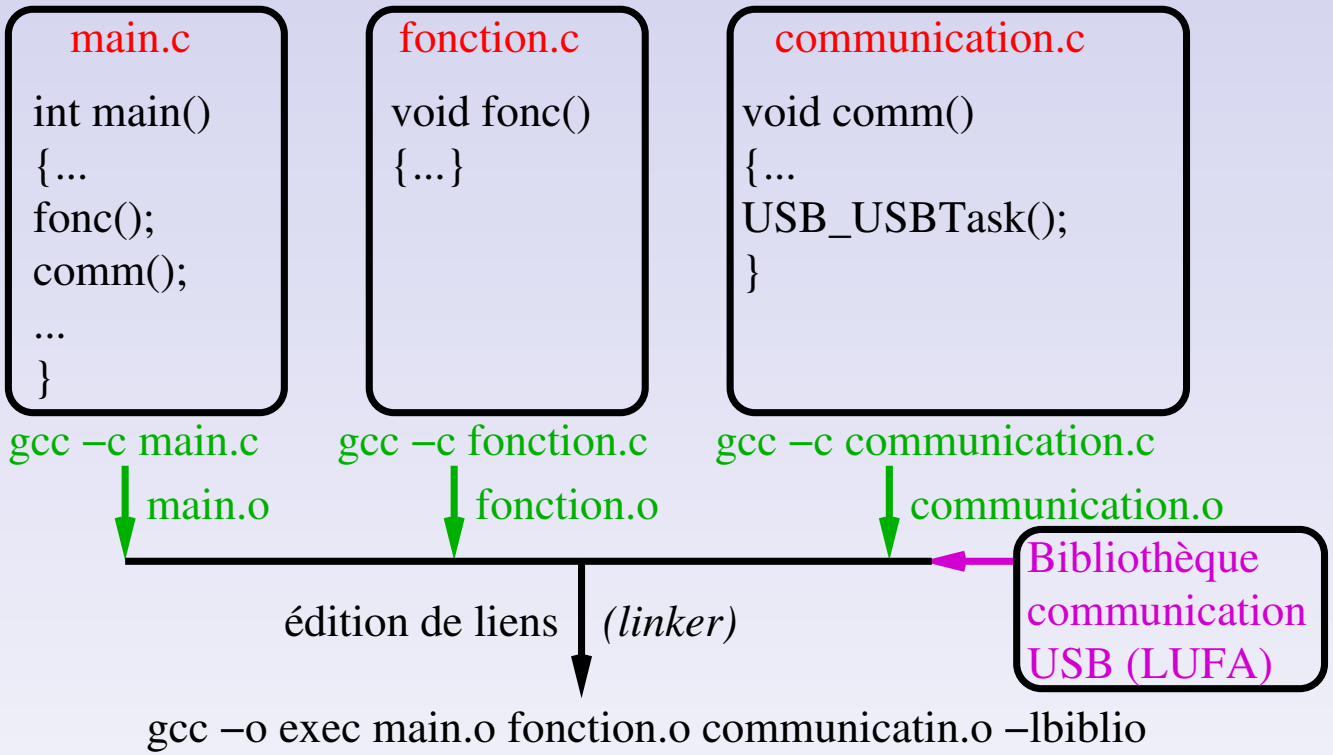
17 janvier 2021

Compilation séparée



- Séparation de groupes de fonctions associées dans des fichiers distincts : lisibilité
- Séparation de l'algorithme (`main()`) des ressources bas-niveau (`comm()`) ou traitements (`fonc()`) : réutilisation
- Exploitation de bibliothèques externes (e.g. communication sur bus USB)
- Pour chaque code source, un fichier d'entête avec les prototypes de fonctions : **JAMAIS** de code C dans un `.h`.

Compilation séparée



- Code source C → objet : `gcc -c` avec chemin des fichiers d'entêtre par `-I`
- Objets + bibliothèques → exécutable : `gcc *.o -o exec` avec chemin des bibliothèques statiques (`.a`) par `-L`
- Lien avec une bibliothèque : `-lbiblio` fait appel à `libbiblio.a`

Makefile : principes

Outil (`make`) d'automatisation de la séquence de compilation selon des règles de dépendance et de résolution des dépendances

- s'appuie sur un script (ASCII) décrivant les étapes de compilation
- ce script se nomme par convention `Makefile` ou `makefile`
- pour un fichier nommé différemment, `make -f fichier`
- une règle de l'objectif à atteindre : `all`
- chaque règle est suivie de `: dépendances`
- chaque règle est suivie d'une ou plusieurs lignes préfixées de TAB [1] avec la solution pour passer de la dépendance à la règle
- Example :

```
programme.o: programme.c
    gcc -c programme.c -o programme.o
```

1. oubli de la tabulation : `Makefile:8: *** missing separator. Stop.`

Makefile : introduction

Dans l'exemple précédent :

```
all: exec

exec: main.o communication.o fonction.o
	gcc -o exec main.o communication.o fonction.o -Lrep/→
	    ↪biblio -lbiblio

main.o: main.c
	gcc -I rep/inc -c main.c

fonction.o: fonction.c
	gcc -I rep/inc -c fonction.c

communication.o: communication.c
	gcc -I rep/inc -c communication.c
```

- la règle `all` (mot clé de `make`) nécessite les objets
- les objets nécessitent les codes sources (C)
- tenter de compiler un objet donc le code source n'existe pas :
  `make: *** No rule to make target 'x.c', needed by 'x.o'. Stop.`

Makefile : variables

Variables :

- éliminer les redondances et les risques d'oubli lors de modifications du Makefile
- réutilisation du Makefile (ne changer que la cible)
- passage d'arguments depuis la ligne de commande en shell (?=)

```
CFLAGS=-Irep/inc
LDFLAGS=-Lrep/biblio -lbiblio
EXECUTABLE?=exec

all: $(EXECUTABLE)

$(EXECUTABLE): main.o communication.o fonction.o
	gcc -o $(EXECUTABLE) main.o communication.o fonction.o→
	    ↪ $(LDFLAGS)

main.o: main.c
	gcc $(CFLAGS) -c main.c

fonction.o: fonction.c
	gcc $(CFLAGS) -c fonction.c

communication.o: communication.c
	gcc $(CFLAGS) -c communication.c
```

valeur par défaut de exécutable (exec) peut être modifiée en ligne de commande →

```
EXECUTABLE=toto make
```

Makefile : principe

- Automatise la séquence de commandes pour compiler un code source
- `make` ne recompile que les dépendances qui sont plus récentes que les règles (basé sur la date de création des fichiers) ⇒ gain de temps sur les gros projet (e.g. noyau Linux)
- il existe des générateurs de Makefile : `cmake`, `configure` (autoconf/M4) ...
- ajout de règles en complément de `all:` : habituellement `clean`, peut être `flash` ...

Makefile : conclusion

```
CFLAGS=-Irep/inc
LDFLAGS=-Lrep/biblio -lbiblio
MCU=atmega32u4
EXECUTABLE?=exec

all: $(EXECUTABLE).bin

$(EXECUTABLE).bin: main.o communication.o fonction.o
	gcc -o $(EXECUTABLE) main.o communication.o fonction.o $(LDFLAGS)
	objcopy -O binary $(EXECUTABLE) $(EXECUTABLE).bin

main.o: main.c
	gcc $(CFLAGS) -c main.c

fonction.o: fonction.c
	gcc $(CFLAGS) -c fonction.c

communication.o: communication.c
	gcc $(CFLAGS) -c communication.c

clean:
	rm *.o $(EXECUTABLE) $(EXECUTABLE).bin

flash:
	avrdude -c avr109 -b57600 -D -p $(MCU) -P /dev/ttyACM0 -e -U flash:w:$(EXECUTABLE).bin
```

`make clean` pour effacer les objets et le résultat, `make flash` pour flasher le microcontrôleur

Makefile : avancé

Des symboles qui facilitent la vie :

- $@ pour la règle (cible)
- $? pour les dépendances plus anciennes que la cible
- $^ pour toutes les dépendances
- $(wildcard *.o) pour toutes les dépendances

```
CFLAGS=-Irep/inc
LDFLAGS=-Lrep/biblio -lbiblio
EXECUTABLE?=exec
OBJ=$(wildcard *.c)

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJ)
	gcc -o $@ $(OBJ) $(LDFLAGS)

clean:
	rm *.o
```

Une liste de variables définies par défaut par `make` [2]

2. www.gnu.org/software/make/manual/html_node/Implicit-Variables.html

Makefile : exercices

1. Proposez un `Makefile` capable de compiler un de vos programmes à destination de l'Atmega32U4
2. Déplacez la phase d'initialisation de l'Atmega32U4 (code avant `while (1) {...}`) dans un code source `init.c` séparé du programme principal et modifiez `Makefile` en conséquent.
3. Ajouter la règle pour nettoyer le répertoire de travail des fichiers temporaires et de l'exécutable lors du lancement de la commande `make propre`
4. Ajouter la règles pour transférer l'exécutable au microcontrôleur par la commande `make ecrit`