INF 3206: Compilation, Makefile et débogage

GBETNKOM N. Jeff, ADAMOU Hamza

Avril 2023

Rappels

Sous Linux, **GCC** signifie **GNU Compiler Collection**. Il s'agit d'un système de compilation pour les différents langages de programmation. Il est principalement utilisé pour compiler les programmes C et C++. Il prend le nom du programme source comme argument nécessaire ; les autres arguments sont facultatifs, comme le débogage, l'avertissement, le fichier objet et les bibliothèques de liaison. Il contient les outils suivants :

- GNU Make : outil pour l'automatisation de la compilation.
- GNU Binutils : il s'agit d'une suite qui contient des outils utilitaires binaires tels que l'éditeur de liens et l'assembleur.
- GNU Debugger : il est utilisé pour le débogage des applications.
- GNU Autotools : il s'agit d'un ooutil de construction qui contient des outils utilitaires tels que Autoconf, Autoheader, Automake et Libtool.
- GNU Bison : c'est un générateur d'analyseur syntaxique comme lex et yacc.

Exercice 1

1. Quelles sont les quatre (04) étapes principales de la compilation en C ?
2. Présenter l'outil **cpp** à l'aide du manuel d'aide Linux.

   NB : parler des fichiers d'entrée et de sortie ainsi que des différentes options principales de la commande.
3. Expliquer l'usage de l'option **-S** avec gcc. Quelle est la particularité du produit de cette opération ?
4. À l'aide du manuel d'aide, prénter et contraster les outils **as** et **ld**.

Exercice 2

1. Écrire un programme C qui retourne un tableau avec des entiers positifs aléatoires. La taille de ce tableau est passée en argument.
2. Renommer votre fichier **myprog.c**.
3. Donner et commenter le résultat des commandes suivantes puis expliquer les options éventuellement utilisées :
   (a) `gcc myprog.c`
   (b) `gcc myprog.c -o runme`
   (c) `gcc -E helper.c o init.i`
   (d) `gcc -S myprog.c`
   (e) `gcc -c myprog.c`
   (f) `as -o myprog2.o myprog.s`
   (g) `ld -o myprog myprog.o`
4. Dire à quelles étapes de la compilation renvoie chacune des étapes précédentes.
5. Quand dit-on d'un programme qu'il est robuste ?

   Un "**warning**" est un message de diagnostic qui identifie une construction de code susceptible d'être une erreur. GCC émet également des messages de diagnostic lorsqu'il rencontre du code ou une utilisation qui semble douteuse ou ambiguë.

6. Pour chacune des options suivantes, donner son utilité et un cas d'utilisation :
   (a) `-fsyntax-only`
   (b) `-pedantic-errors`
   (c) `-Wbad-function-cast`
   (d) `-Wunused`
   (e) `-Werror`
   (f) `-Wall`

Exercice 3

A level 1 student is trying to write a program to calculate the double of a number in C without doing any multiplication. Not being able to get a correct result, he approaches you in the hope of finding a solution to his problem. Answer the following questions to explain to him where the problem is coming from.

1. What is debugging ?
2. Write the following program in a file named **addition.c**.

```c
// Double without multiplication
// Author: XYZ
#include <stdio.h>

int main (int argc, char* argv[]){
    int x;
    int a = x;
    int b = x;
    int c = a + b;
    printf("%d\n", c);
    return 0;
}
```

3. Complete this table with either the `gdb` command or its function.

Commands	Function
file	
r	
b	
	go to next instruction, diving into the function.
until	
	disable a breakpoint.
	used to display the stored value.
delete	
l	

4. What is a **breakpoint** in a debugger ?
5. Compile the program with the following command and explain its options :
   `gcc -std=c99 -g -Wall -o addition addition.c`
6. Open the executable file using `gdb`.
7. Run the program. What is the output ?
8. Add a breakpoint on the main function of the program and display the values stored by the different variables. Where is the error coming from ?

Exercice 4

1. Qu'est ce qu'un **Makefile** ? Pourquoi est-il nécessaire d'en faire usage ?
2. Préparer un fichier **Makefile** contenant deux variables, une pour le compilateur (`gcc`) et une pour les options de compilation (`-g -Wall -pedantic -c`).

   L'ordonnancement est la planification de l'exécution de processus qu'ils soient interdépendants ou non. On considère qu'un processus est représenté par son PID, sa priorité et sa durée d'exécution. Considérons l'usage d'un fichier texte pour les entrées/sorties. Chaque enregistrement de celui-ci se constitue comme suit : PID PRIORITE DUREE.

3. Proposer un type de données pour les processus ainsi que quelques fonctions élémentaires (setters et getters), dans les fichiers **process.h** et **process.c**.
4. Écrire une fonction permettant de créer un tableau de processus dont la taille est spécifiée en paramètre avec des propriétés aléatoires strictement positives, dans les fichiers **process_handling.h** et **process_handling.c**.
5. Écrire une fonction permettant de stocker les informations d'un tableau de processus dans un fichier texte de ceux-ci.
6. Écrire la fonction permettant d'obtenir un tableau de processus à partir d'un fichier texte contenant leurs informations.
7. Implémenter l'algorithme **Shortest Job First** qui prend un tableau de processus et retourne l'ordre d'exécution en fonction des temps d'exécution, dans les fichiers **scheduling.h** et **scheduling.c**.
8. À ce stade, vos fichiers devraient contenir les fonctions suivantes :

   ```
   process :
       - setters
       - getters
   process_handling :
       - process_creation
       - processes_to_file
       - file_to_processes
   scheduling :
       - shortest_job_first
   ```

   Préparer la fonction **main** de votre fichier principal :
   — Le programme principal servira à utiliser la fonction d'ordonnancement.
   — On passera le nombre de processus ainsi que les noms des fichiers de stockage pour les questions 5 et 7 en argument.
   — Attention à la robustesse !
9. Compléter votre Makefile et exécutez-le.

Exercice 5 : Calcul matriciel

Le calcul matriciel est une branche des mathématiques qui traite de l'algèbre linéaire axée sur les matrices. Le but de cet exercice est de créer une bibliothèque permettant de créer et d'effectuer des opérations sur les matrices comme vous le faites par exemple avec **math.h** pour certaines opérations mathématiques.

1. Dans le module **type**, proposer un type **matrice** composé de son nombre de lignes, de son nombre de colonnes et d'un tableau à deux dimensions de réels contenant les données. Proposer également la fonction de **création** de la matrice, génération d'une matrice de valeurs réelles aléatoires, de chargement d'une matrice à partir d'un fichier, d'affichage d'une matrice ; et trois fonctions **no_lignes**, **no_colonnes** et **donnees** qui renvoient respectivement le nombre de lignes de la matrice, son nombre de colonne et les données contenues dans celles-ci.
   — La création prend en entrée le nombre de lignes, le nombre de colonnes, un tableau à deux dimensions et retourne une matrice.

— La génération prend en entrée le nombre de lignes et le nombre de colonnes et retourne une matrice possédant des valeurs rélles aléatoires.
— Le chargement prend en entrée un fichier et retourne une matrice.

2. Dans le module **matrix**, proposer les fonctions de transposition d'une matrice, d'addition, de soustraction, de multiplication par un scalaire et de multiplication de deux matrices.
   — La transposition d'une matrice $m \times n$ prend en entrée une matrice et retourne une matrice $n \times n$ qui est la transposée de la première.
   — L'addition et la soustraction prennent en entrée deux matrices de même dimension et retourne une matrice résultat.
   — La multiplication par un scalaire prend en entrée une matrice de dimension $m \times n$ et un nombre réel et retourne une matrice.
   — La multiplication prend en entrée deux matrices A et B, de dimension $m \times p$ et $p \times n$ respectivement, et retourne une matrice de dimension $m \times n$. résultat, si et seulement si **le nombre de colonnes de A est égal aux nombres de lignes de B**.
3. Écrire le Makefile. Les fichiers type.o et matrix.o doivent être stockés dans le répertoire `./build/`
4. À l'aide de la commande `ar`, créer une archive nommée **libmatrix.a** à partir du fichier objet des opérations. Vous venez de créer une bibliothèque statique.
5. Écrire un programme nommé **program.c**. Il se constitue d'un menu permettant à l'utilisateur de choisir une des opérations du module **matrix**, puis de préciser l'opération à effectuer et enfin d'avoir le résultat dans un fichier.
6. Ajouter une directive de compilation du programme **program.c** et lier votre archive à la compilation.
7. Maintenant, modifier le Makefile de manière à compiler le module matrix en ajoutant l'option `-fPIC` pour la création d'une bibliothèque dynamique. À ce stade, le fichier Makefile devrait contenir une directive pour la compilation "simple" du module `matrix` et une directive pour la compilation en vue de la création d'une bibliothèque dynamique.
8. Créer la bibliothèque partagée avec `gcc`. La nommer **libmatrix.so**.
9. Pour finir, modifier le Makefile de manière à créer une deuxième exécutable nommé **program_dyn** qui vient de la compilation du programme de la question 5. lié à notre librairie précédemment créée.

Formules sur le calcul matriciel :
— **Transposée :** $\forall i, j \in \mathbf{N}$, $0 \leq i < m$ et $0 \leq j < n$, $A^T_{ij} = A_{ji}$
— **Addition :** $\forall i, j \in \mathbf{N}$, $0 \leq i < m$ et $0 \leq j < n$, $C_{ij} = A_{ij} + B_{ij}$.
— **Soustraction :** $\forall i, j \in \mathbf{N}$, $0 \leq i < m$ et $0 \leq j < n$, $C_{ij} = A_{ij} - B_{ij}$.
— **Multiplication par un scalaire :** $\forall i, j \in \mathbf{N}$, $0 \leq i < m$ et $0 \leq j < n$, $C_{ij} = \lambda * A_{ij}$.
— **Multiplication :** ici, nous considérons que la matrice A est une matrice $m \times n$ et B, une matrice $n \times p$. On définit alors la matrice résultat C, $\forall i, j, k \in \mathbf{N}$, $0 \leq i < m$ et $0 \leq j < p$, $C_{ij} = \sum_{k=0}^{m-1} A_{ik} B_{kj}$.