Ecole Supérieur en Informatique de Sidi Bel Abbès

Module: **Système d'exploitation 1** 1ère année Cycle Secondaire
Semestre 1 Année universitaire: 2020-2021

Fiche de TP n°02

Compilation et édition de liens

Exercice 1: Phases de compilation

Soit le programme suivant (`phases.c`) :

```
  /* ------------------------------
    Programme phases de compilation
   ------------------------------------------------------------ */
#include <stdio.h>
#include <stdlib.h>
#define MAX 12

int ab = 3;
const int cn = 5;
extern int vex;

int main()
{
   int xyz; /* declaration et definition de xyz */
   xyz = MAX;
   printf("Bonjour : %d\n", ab + cn + vex + xyz);
   exit(EXIT_SUCCESS);
}
```

1. Vérifiez dans la page de manuel de *gcc* le sens de l'option *-E*.
2. Produisez le résultat du prétraitement par le préprocesseur sur le programme *phases.c* dans un fichier *phases.i*.
3. Comparez le contenu des fichiers *phases.c* *et* *phases.i*. En particulier :
   a. Combien de lignes comporte chacun des fichiers *phases.c* et *phases.i* ?
   b. Qu'est devenu la ligne *#include <stdio.h>*
   c. Que sont devenus les commentaires ?
   d. Quel traitement a été réalisé sur la ligne *xyz= MAX;*
4. Vérifiez dans la page de manuel de *gcc* le sens de l'option *-S* et produisez le code assembleur correspondant à *phases.c* dans un fichier nommé *phases.s.*
5. Dans ce code assembleur, identifiez les différentes sections *.data*, *.rodata*, et *.text*.
   a. Quels symboles sont associés à la section *.data* ? À quoi correspond cette section ? Vérifiez en ajoutant une déclaration au programme *phases.c*.
   b. Quels symboles et valeurs sont associés à la section *.rodata* ? À quoi correspond cette section ? Vérifiez en modifiant *phases.c* pour ajouter un élément à cette section.
   c. Quels sont les identificateurs à exporter (internes ou connus) et à importer (externes ou inconnus) ? (Nous reviendrons sur ce point lors de l'édition de liens.)
   d. Quel est le point d'entrée de ce programme ?
6. Trouvez comment indiquer à *gcc* de générer le fichier objet *phases.o* à partir de *phases.c* ou de *phases.s.*
7. Quel est le rôle de la commande *nm ?*

a. Comparer les résultats de la commande *nm* sur ce fichier *.o* avec ce que vous aviez identifié comme sections dans le code assembleur.
b. Qu'indique *nm* pour les symboles *printf*, *exit*, *et* *vex* ?
c. Où ces symboles sont-ils définis ? Quel programme est en charge de rechercher ces symboles ?
d. Qu'est ce qui change dans le résultat de *nm* si on supprime l'utilisation de *vex* dans l'instruction *printf* du fichier *phases.c* ?

8. Ouvrir le fichier *phases.o* en utilisant un éditeur de texte (ex :vim). Par quel quelles lettres commence ce fichier *.o* ?
9. Quelles sont les chaînes de caractères discernables dans ce code binaire ?
10. Comment expliquez-vous le résultat de la commande *strings* sur ce fichier *.o* ?
11. En utilisant l'utilitaire *objdump* faite la correspondance entre le code binaire et :
    a. les instructions assembleurs correspondantes (option `-S`).
    b. les différents segments (option `-s`).
    c. la table des symboles (option `-t`)
12. Identifier les symboles (i.e. identificateurs) qui sont référencés dans le fichier objet mais qui ne sont pas définis. Vous pouvez constater que le fichier *phases.o* n'est pas exécutable car tous les identificateurs ne sont pas définis.
13. Un identificateur de notre code source n'est pas défini ; commentez sa déclaration et son utilisation afin d'obtenir un code exécutable. Exécutez votre programme.
14. En utilisant l'utilitaire `objdump` comparez sommairement : les table des symboles et les segments ; du fichier objet *phases.o* et du fichier exécutable *phases* associé (sont-ils les mêmes, quelque chose a-t-il été ajouté ?).

Exercice 2 : Edition des liens

Il est possible de créer un fichier exécutable à partir de plusieurs fichiers objets. On parle alors de compilation séparée. Ceci permet de découper un programme, et d'éviter d'avoir un seul (immense) fichier source : il est possible d'avoir plusieurs fichiers source en `.c`, chacun sera compilé pour donner un fichier objet en `.o`, et enfin l'édition de liens de tous ces fichiers `.o` produira l'exécutable. Nous allons illustrer ce concept à l'aide du programme `compilSeparee.c` :

```
#include <stdio.h>
int main(void){
    int op1, op2 ;
    printf("Entrez le premier entier ") ;
    scanf("%d",&op1) ;
    printf("Entrez le second entier ") ;
    scanf("%d",&op2) ;
    printf("Le plus grand est %d\n", plusgrand(op1,op2)) ;
    return 0 ;
}
```

qui utilise la fonction `plusgrand`, définie dans le fichier `plusgrand.c` :

```
int plusgrand(int arg1, int arg2){
    return arg1<arg2 ? arg2 : arg1 ;
}
```

1. Produisez les fichiers objets `plusgrand.o` et `compilSeparee.o`, correspondant respectivement aux fichiers source `plusgrand.c` et `compilSeparee.c`.
2. Ensuite, réalisez l'édition de lien à partir des deux fichiers objets, pour créer l'exécutable `compilSeparee`, et testez-le (Qu'est ce que vous remarquez ?),
3. En utilisant l'utilitaire `objdump`, inspectez la table des symboles de `plusgrand.o` et `compilSeparee.o`. Intéressez-vous particulièrement au symbole `plusgrand` dans ces deux fichiers objets : que signifie la mention `*UND*` a gauche de `plusgrand` ?

Exercice 3: Programmes et données

Soit le programme C suivant : *`tailledonnees.c`*

```
#include <stdio.h>
#include <stdlib.h>
#define K 100
const int KBIS=100;
int tab[K];
static float f;
int main(int argc, char *argv[]){
    int l;
    char*s[]={"un","deux"};
    static int j=2;
    int *td=(int*)malloc(5*sizeof(int));
    printf("nom \t taille\t valeur\t adresse\n");
    printf("%s \t %d \t %d \t %p \n","l",sizeof(l),l,&l);
    printf("%s \t %d \t %p \t %p \n","s",sizeof(s),s,&s);
    printf("%s \t %d \t %p \t %p \n","s[0]",sizeof(s[0]),s[0],&(s[0]));
    printf("%s \t %d \t %d \t %p \n","argc",sizeof(argc),argc,&argc);
    printf("%s \t %d \t %p \t %p \n","tab",sizeof(tab),tab,&tab);
    printf("%s \t %d \t %d \t %p \n","tab[1]",sizeof(tab[1]),tab[1],&(tab[1]));
    printf("%s \t %d \t %-5.2f \t %p \n","f",sizeof(f),f,&f);
    printf("%s \t %d \t %d \t %p \n","j",sizeof(j),j,&j);
    printf("%s \t %d \t %p \t %p \n","td",sizeof(td),td,&td);
    printf("%s \t %d \t %d \t %p \n","td[0]",sizeof(td[0]),td[0],&(td[0]));
}
```

1. Compilez puis exécuter ce programme
2. Quelle est la taille de chaque objet (variable ou une zone de données pointée) sur votre machine ?
3. En fonction des valeurs obtenues, que dire de l'initialisation des variables ?
4. L'espace mémoire d'un processus étant vu comme un ensemble de segments, dans quel segment est défini chaque objet ?
5. Quelle est la durée de vie de chaque objet ?