***TIC – Filière Informatique***

**Jigé Pont**

**Nicolas Fuchs**

**Alan Sueur**

**Microprocesseur 3**

**Labo 1**



20/02/2018

# Configuration

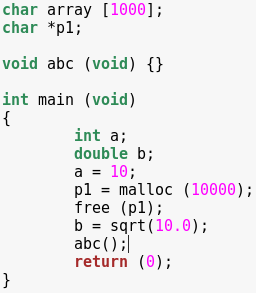
Système d'exploitation : Kali Linux Rolling 64 bits  
Processeur : Intel Core i7-4510U CPU @ 2.00GHz

# Compilation

La version 6.3.0 de gcc qui est actuellement utilisée sur la machine virtuelle réalise le linkage automatiquement. Avec une autre version du compilateur la fonction sqrt qui est utilisée ne serait pas reconnue. Il faudrait alors inclure la librairie dans le code comme suit :

#include <math.h>

file.c



gcc -Wall -Wextra -c file.c -o file.o

Commande compilation :

Sortie console :



Analyse : Aucune erreur n'est apparue par contre plusieurs warnings :

* Les variables a et b sont déclarées mais pas utilisées
* Déclarations implicites des fonctions malloc, free et sqrt

Ces warnings sont dus à l'absence de header qui déclare les prototypes des fonctions utilisées. Pour les supprimer, il suffit d'inclure les headers math.h et stdlib.h.

# Linker

gcc -Wall -Wextra file.o -o file

Commande linker :

Le linkage fonctionne indépendemment de l'inclusion des headers. Le linker connaît le chemin par défaut vers les librairies préinstallées.

# Options

gcc -Xlinker -Map=file.map -Wall -Wextra file.o -o file

Commande linker avec map file :

gcc -S file.c

Commande compilation avec sortie assembleur :

gcc -g -Wall -Wextra -c file.c -o file.o

Commande compilation avec information debugging :

Analyse : Le map file se nomme file.map, le fichier assembleur se nomme file.s et les informations de debugging sont incluses dans le fichier objet file.o qui est passé de la taille 1992 octets à 4120 octets.

# Adresses et mémoire

"rip" pour les processeurs Intel est le registre équivalent à "pc" de ARM. En consultant la valeur de ce registre, on peut y trouver le début du programme.

Pour consulter cette valeur, il faut aller dans l'onglet "status" de ddd, puis sélectionner "registers" et rechercher la ligne "rip". Il est également possible de consulter l’adresse grâce au bouton "display".



L’adresse du début du code est l’adresse du début du code segment. Pour l’obtenir il suffit de prendre l’adresse de la fonction void abc(void)  via un printf (voir plus bas). Comme l’implémentation de la fonction abc est la première chose qui est faite dans le code (les deux déclarations de array et p1 sont des globales et iront donc dans le data segment) ce là correspond bien à l’adresse du début du code.

Les adresses des variables array, p1, \*p1, a et b ont été obtenues en faisant un printf.

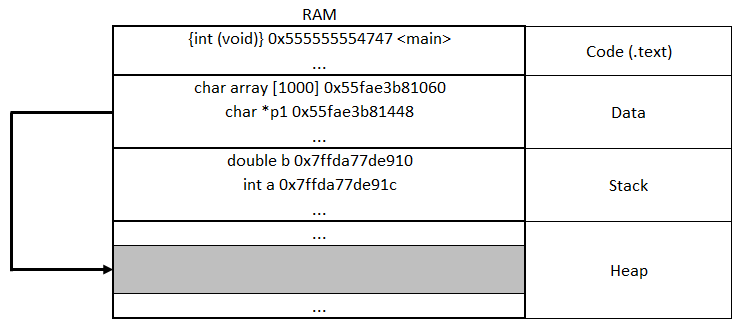


Pour voir l'adresse du pointeur p1, il faut faire printf("\%p", &p1) et si on veut l'adresse pointé il suffit de faire printf("\%p", p1).

Ce qui nous donne le résultat suivant :



Voici la représentation de la mémoire du programme :



# Nom de fonctions

objdump -x file.o

Pour trouver les noms de fonction du fichier ".c", nous avons utilisé la commande :

Le résultat obtenu est alors ceci :

0000000000400626 g F .text 0000000000000007 abc

000000000040062d g F .text 00000000000000ee main

En créant le fichier ".C" nous avons dû ajouter le cast afin que la compilation fonctionne :

P1 = (char\*) malloc (10000) ;

Nous avons réutilisé la commande objdump afin de comparer avec le .c et avons obtenu :

0000000000000000 g F .text 0000000000000007 \_Z3abcv

0000000000000007 g F .text 00000000000000ee main

Pour surcharger la méthode abc, il suffit d’ajouter les lignes suivantes dans le fichier file.C :

void abc (int a)

{

}

Après avoir réutilisé la commande objdump, nous pouvons constater les résultats suivants :

0000000000000000 g F .text 0000000000000007 \_Z3abcv

0000000000000007 g F .text 000000000000000a \_Z3abci

0000000000000011 g F .text 00000000000000ee main

On peut constater que \_Z3 veut dire que la méthode abc retourne un void, alors que le v après abc signifie void en paramètre et i signifie int en paramètre. La méthode main par contre n’a pas changé.

