2 - Assembleur et débogage

Vous pouvez travailler en binôme si vous le souhaitez, mais chacun d'entre vous doit programmer sur une machine :
- soit sur la machine virtuelle Arch Linux de l'IUT appelée AL_SYS (id : iut, mdp : iut) ;
- soit sur un ordinateur personnel disposant d'un système Linux 64 bits (virtuel ou réel) :
  - vous pouvez notamment télécharger la machine virtuelle ci-dessus à l'adresse suivante : `ftp://164.81.120.20/` par une connexion filaire depuis l'IUT.

Les programmes dont il sera question dans cette séance et les suivantes ne sont pas à recopier à la main, mais à télécharger sur `https://thomas-hugel.gitlab.io`

1 Assembleur

1. Créez un répertoire **`M2101/HelloWorld`**. Mettez-y le programme appelé **`hello_world.c`**. Compilez-le avec l'option de débogage dans un fichier **`hello_world.out`**. Comparez la taille de ces deux fichiers. Ouvrez le fichier **`hello_world.out`** dans un éditeur de texte (**`gedit`**), et cherchez-y la chaîne «main».

Listing 1 – hello_world.c

```
1  #include <stdio.h>
2
3  int main(void) {
4      char message[] = "Bonjour le Limousin !\n";
5      for (int i = 0; i < 10; ++i) {
6          printf("%s", message);
7      }
8      return 0;
9  }
```

2. Afin de rendre le code assembleur lisible par un humain, tapez la commande suivante :

```
objdump -M intel -Dtx hello_world.out > hello_world.out.txt
```

3. Ouvrez le fichier obtenu, et cherchez-y cette fois le code de la fonction **`main`**. Ce code est représenté au listing 3.
4. À l'aide de la commande **`man ascii`**, déterminez l'encodage ASCII hexadécimal de "Bonjour le Limousin !\n".
5. Regardez dans l'assembleur comment cette chaîne est traitée. Remarquez bien l'ordre d'affichage : l'affichage est tel qu'il permet de lire directement les adresses de 64 bits, et l'architecture est petit-boutiste...
6. Essayez de deviner ce que la variable **`i`** est devenue.

2 Débogage avec gdb

1. En une ligne de commande (souvenirs, souvenirs...), ajoutez **`set disassembly intel`** à la fin du fichier **`~/.gdbinit`** (c'est pour configurer l'affichage dans **`gdb`**).
2. Lancez les commandes du listing 2 et analysez ce que vous voyez (faites le lien avec le code source).
3. Quelle est l'adresse de **`i`** ? et celle de **`main`** ? Comment se fait-il qu'il y ait une telle différence entre ces deux adresses ?
4. Quel est, en hexadécimal, le plus grand entier non signé représentable sur 47 bits ? Pourquoi cette question ?
5. Comment la boucle se traduit-elle en assembleur ?
6. Dans quelle(s) partie(s) de la mémoire est mise la chaîne «Bonjour le Limousin !\n» ?
7. Faites un dessin de la situation (prenez une page A4 complète), en faisant apparaître **`main`**, **`i`**, **`message`**, **`rip`**, **`rsp`** et **`rbp`**, avec leurs adresses et leurs valeurs.
8. Tapez **`q`** pour quitter le débogueur (et **`y`** pour confirmer).

Listing 2 – Session de débogage

```
1  gdb ./hello_world.out
2  list
3  b main
4  r
5  disass main
6  <Enter>
7  n
8  disass main
9  <Enter>
10 n 11
11 disass main
12 <Enter>
13 x/1wd &i
14 x/1wd $rbp-0x24
15 x/1s $rbp-0x18
16 ni
17 x/1wd &i
18 x/5i $rip
19 ni
20 x/5i $rip
21 ni
22 disass main
23 <Enter>
24 x/12gx $rsp
25 x/12gx $rbp
```

Listing 3 – La fonction main compilée.

```
000000000000069a <main>:
 69a:   55                      push   rbp
 69b:   48 89 e5                mov    rbp,rsp
 69e:   48 83 ec 30             sub    rsp,0x30
 6a2:   64 48 8b 04 25 28 00    mov    rax,QWORD PTR fs:0x28
 6a9:   00 00
 6ab:   48 89 45 f8             mov    QWORD PTR [rbp-0x8],rax
 6af:   31 c0                   xor    eax,eax
 6b1:   48 b8 42 6f 6e 6a 6f    movabs rax,0x2072756f6a6e6f42
 6b8:   75 72 20
 6bb:   48 ba 6c 65 20 4c 69    movabs rdx,0x756f6d694c20656c
 6c2:   6d 6f 75
 6c5:   48 89 45 e0             mov    QWORD PTR [rbp-0x20],rax
 6c9:   48 89 55 e8             mov    QWORD PTR [rbp-0x18],rdx
 6cd:   c7 45 f0 73 69 6e 20    mov    DWORD PTR [rbp-0x10],0x206e6973
 6d4:   66 c7 45 f4 21 0a       mov    WORD PTR [rbp-0xc],0xa21
 6da:   c6 45 f6 00             mov    BYTE PTR [rbp-0xa],0x0
 6de:   c7 45 dc 00 00 00 00    mov    DWORD PTR [rbp-0x24],0x0
 6e5:   eb 1c                   jmp    703 <main+0x69>
 6e7:   48 8d 45 e0             lea    rax,[rbp-0x20]
 6eb:   48 89 c6                mov    rsi,rax
 6ee:   48 8d 3d bf 00 00 00    lea    rdi,[rip+0xbf]
 6f5:   b8 00 00 00 00          mov    eax,0x0
 6fa:   e8 81 fe ff ff          call   580 <printf@plt>
 6ff:   83 45 dc 01             add    DWORD PTR [rbp-0x24],0x1
 703:   83 7d dc 09             cmp    DWORD PTR [rbp-0x24],0x9
 707:   7e de                   jle    6e7 <main+0x4d>
 709:   b8 00 00 00 00          mov    eax,0x0
 70e:   48 8b 4d f8             mov    rcx,QWORD PTR [rbp-0x8]
 712:   64 48 33 0c 25 28 00    xor    rcx,QWORD PTR fs:0x28
 719:   00 00
 71b:   74 05                   je     722 <main+0x88>
 71d:   e8 4e fe ff ff          call   570 <__stack_chk_fail@plt>
 722:   c9                      leave
 723:   c3                      ret
 724:   66 2e 0f 1f 84 00 00    nop    WORD PTR cs:[rax+rax*1+0x0]
 72b:   00 00 00
 72e:   66 90                   xchg   ax,ax
```