Compilation – TP : Découverte de x86-64

Université Paris Diderot – Master 1

(2021-2022)

Cette feuille de TP vous donne les étapes à suivre pour découvrir l'architecture x86-64/System V. Voici les ressources documentaires que nous vous conseillons (en suivant les recommandations Jean-Christophe Filliâtre) :

— Les notes d'Andrew Tolmach sur x86-64 :
https://www.lri.fr/~filliatr/ens/compil/x86-64.pdf

— Les notes de Bryant et O'HallaronM
https://www.cs.cmu.edu/~fp/courses/15213-s07/misc/asm64-handout.pdf

— La spécification d'Intel des instructions x86-64 :
https://software.intel.com/en-us/articles/intel-sdm

— La spécification des conventions d'appel System V :
https://wiki.osdev.org/System_V_ABI

Les objectifs de cette séance de travaux pratiques sont :

- de vous familiariser avec quelques rudiments de programmation assembleur x86-64 (quel est le format d'un programme assembleur pour x86-64 en syntaxe GNU, comment assembler un tel programme, quelles sont les instructions qu'il faut connaître pour lire et écrire des programmes basiques, etc.) ;
- de vous apprendre à utiliser un débogueur pour exécuter un programme assembleur pas-à-pas.

Les fichiers sources utilisés pour ces travaux pratiques sont sur le dépôt GIT à l'emplacement `doc/td-x86/code`.

Exercice 1 (Compiler, assembler, exécuter)

1. *Avant de commencer à programmer, partez à la recherche des informations suivantes dans les ressources documentaires données plus haut :*
   (a) *Combien l'architecture x86-64 a-t-elle de registres ?*
   (b) *Quel est le format d'adressage sur cette architecture ?*
   (c) *Quelle est la sémantique des instructions* `movq`, `leaq`, `pushq`, `popq`, `call`, `ret`, `subq`, `j`, `je`, `leave` *et* `cmp` *?*
   (d) *S'agit-il d'une architecture* little endian *ou* big endian *?*
   (e) *Quelles sont les conventions d'appel de l'*Application Binary Interface *x86-64/System V ?*
2. *Après avoir lu et compris le fichier* `exo1.s`, *utilisez* `gcc` *pour le traduire en exécutable avec* `gcc -g -no-pie` *et exécutez-le.*
3. *Observez le code compilé produit par la compilation de* `exo1-2.c` *par la commande :*

   ```
   gcc -fno-PIC -fno-asynchronous-unwind-tables -S exo1-2.c
   ```

   et mettez-le en correspondance avec le programme source.
4. *Donnez l'évolution de la forme de la pile lorsque l'on exécute le programme* `exo1-2` *pour calculer* `fact(2)`.
5. *Comparez le code compilé précédent avec celui obtenu par*

   ```
   gcc -fno-PIC -fno-asynchronous-unwind-tables -O2 -S exo1-2.c
   ```

   Mettez en correspondance les instructions de ce nouveau programme avec les instructions du précédent code compilé. D'après vous, pourquoi le code produit par `-O2` *s'exécutera plus rapidement que celui produit sans cette option ?*

6. *Observez le code compilé du programme* `exo1-3.c`. *Qu'en déduire sur les conventions de passage des arguments aux fonctions ?*

□

Exercice 2 (Débogage de code assembleur) *Dans cet exercice, vous devez utiliser* `gdb` *pour déboguer des programmes écrits en assembleur. Pour pouvoir déboguer un exécutable issu d'un assemblage, il faut l'avoir compilé avec des informations de débogage en passant l'option* `-ggdb` *à* `gcc`, *typiquement :*

```
gcc -o executable-name -ggdb source.s
```

Nous vous recommandons de lancer `gdb` *avec l'option* `--tui` *qui permet d'afficher des informations sur le code source et sur l'état de la machine. Ainsi :*

```
gdb --tui -ex 'layout regs' ./executable-name [options]
```

ouvrira une interface textuelle équipée d'une zone d'affichage du source au milieu de l'écran, d'une zone d'affichage des registres en haut et d'une invite de commande en bas. Vous pourrez alors utiliser les commandes de base suivantes :

- `b label` *: insère un point d'arrêt sur l'étiquette* `label`.
- `r` *: exécute le programme.*
- `r arg0 arg1 .. argN` *: exécute le programme en passant* `arg0` .. `argN` *en arguments.*
- `n` *: passe à l'instruction suivante.*
- `n` *: continue jusqu'au prochain point d'arrêt.*
- `s` *: entre dans le corps d'une fonction appelée par un* `call`.

Vous pouvez compléter ces commandes en lisant la documentation de `gdb`, *en particulier les pages suivantes :*

- *https://sourceware.org/gdb/current/onlinedocs/gdb/Stack.html#Stack pour apprendre à observer la pile,*
- *https://sourceware.org/gdb/current/onlinedocs/gdb/TUI.html#TUI pour apprendre à utiliser l'interface textuelle.*

1. *Le programme* `min.s` *devrait calculer et afficher la valeur minimale du tableau*

   ```
   { 1, 5, 5, 6, 3, -1, 4, 37, -73, 0 }
   ```

   mais il est faux. Corrigez-le !
2. *Le programme* `segfault.s` *produit une erreur à l'exécution. Corrigez-le !*

□

Exercice 3 (Crible d'Eratosthène)

1. *Complétez le fichier* `erato.s` *pour qu'il affiche le nombre de nombres premiers entre 2 et* $2^{28} - 1$ *en utilisant l'algorithme d'Eratosthène.*
2. *Essayez d'optimiser votre programme, par exemple en utilisant des bits pour représenter les cases du crible.*

□