Rapport des exercices du TP

Compilation et exécution

Ce cours portant sur l'assembleur 32 bits microsoft, nous compilons les codes grâce à un fichier `make.bat` dont l'architecture est la suivante :

```
@echo off
c:\masm32\bin\ml /c /Zd /coff path/to/input/file.asm
c:\\masm32\bin\Link /SUBSYSTEM:CONSOLE path/to/output/file.obj
pause
```

Comme tous les membres du groupe travaillent sous Unix, nous utilisons wine pour exécuter le fichier avec cette commande : `wine make.bat`

Le script génère deux fichiers : un .obj et un .exe
Affin d'exécuter le code, nous utilisons encore wine : `wine pathe/to/executable.exe`

Debugging

Le debugger utilise est l'outil x32 de [x64dgb]

A

A.b et A.c

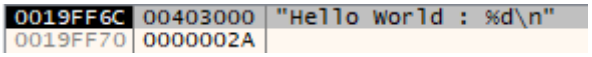
initialisation des variables :

Les deux variables initialisées sont de type define byte, ici, db est utilisé pour déclarer des chaines de caractères. Les chiffres suivant les chaines sont des caractères ASCII (10 pour le saut de ligne et 0 pour indiquer la fin de la chaine)

Programme

L'instruction PUSH permet d'ajouter une valeur en haut de la pile. Ici, 42 est ajouté, puis c'est au tour de l'adresse mémoire de la variable Phrase (mot clé offset).

Jusqu'ici, la pile ressemble à cela :



2A représente 42 en hexadécimal, 00304000 est l'adresse mémoire où est stockée la chaine de caractères. (le haut de la pile est en bas)

L'instruction CALL permet d'appeler des fonctions externes au programme. Dans notre cas, c'est l'instruction crt_printf, qui permet d'afficher les informations passées en paramètres, à la manière du C. La chaine de caractères ne contient qu'un seul spécificateur de format, le processeur va donc dépiler la chaine, puis dépiler 42, pour les utiliser dans le printf.

Les 3 dernières lignes permettent d'attendre que l'utilisateur appuie sur une touche, renvoyer un code de succès d'exécution, puis terminer le processus.

C

C.b

Cette fonction permet de calculer la valeur du n-ieme chiffre de la suite Fibonacci.
Exemple d'execution avec 7 (resultat attendu : 13)

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	1	1	2	3	5	8	13	21	34	55	89	144	233	377	610

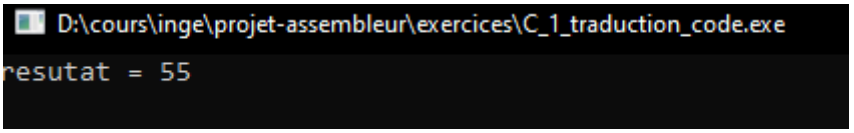
suite de Fibonacci de 0 a 15

n	**i**	**l**	**j**	***k***	
7			1	1	initialisation
7	3	2	1	2	
7	4	3	2	3	
7	5	5	3	5	
7	6	8	5	8	
7	7	13	8	13	
7	8	13	8	13	sortie du for

A la fin de l'exécution, k=13, soit la valeur attendue.

Exécution du programme assembleur

Dans notre programme, nous forçons la valeur de n a 10, nous devons donc nous attendre à un résultat de 55.
À l'exécution du programme, nous obtenons la sortie suivante :



Notre programme fonctionne donc correctement.

E

Cet article nous montre en profondeur comment fonctionnent les appels système. Les appels système sont des appels à des instructions depuis un mode utilisateur, qui seront exécutées dans un mode noyau. Ces niveaux sont aussi appelés rings, il en existe 4, allant de 0 pour le ring kernel (noyau) a 3 pour le ring user (utilisateur). La valeur de ce ring est représenté sur le CS par les 2 bits de poids le plus faible.
Dans cet article, l'auteur nous montre les appels système avec un appel a l'API Windows.

Au début de l'exécution du programme, si nous lisons le CS, sa valeur est 0b101011, le ring est donc de niveau 3, soit user. Si nous poursuivons l'exécution du code, nous remarquons que l'une des fonctions appelees est SYSENTER. C'est précisément cette opération qui permet de passer du ring user au ring kernel. Cette instruction a remplacé les interruptions système, autrefois utilisées pour les appels système, car plus rapide.
Le principe de fonctionnement de SYSENTER est le suivant :

- copier la valeur de IA32_SYSENTER_CS dans le CS
- copier la valeur de IA32_SYSENTER_ESP dans le ESP
- copier la valeur de IA32*SYSENTER_EIP dans le EIP*
  *Les registres IA32_SYSENTER*** sont ce que l'on appelle des MSR (Model Specific Register). Ils permettent entre autres d'activer certaines fonctions du CPU.

Nous pouvons maintenant contrôler le ring dans lequel nous nous trouvons de deux façons différentes :

- consulter le CS, dans ce cas-là, le CS prendra la valeur 0×8, confirmant le ring kernel.
- Consulter l'adresse sur l'EIP.
  En effet, le système d'adressage réserve les adresses supérieures a 80000000 pour les appels kernel. Dans le cas de l'article, la valeur de l'EIP est 80541520.

Avant l'entrée dans le mode kernel, l'instruction a exécuter a été stockée dans EAX, en multipliant par 4 cette valeur, nous retrouvons l'adresse de l'instruction à stocker dans EBX, qui est appelée plus tard dans le programme.

Ensuite, pour retourner dans le userland (ring 3) un appel a la fonction SYSEXIT est fait. son fonctionnement est basiquement l'inverse de SYSENTER, mais l'article ne détaille pas plus.

Pour conclure, nous avons vu comment fonctionnent les appels système, en changeant le mode d'exécution des instructions. De cela, nous pouvons imaginer plusieurs hooks, c'est-à-dire un fonctionnement detourne du programme. Par exemple modifier l'instruction executee en mode kernel. L'utilite d'un tel hook pourrait etre bienveillante comme malveillante.

Maxime Soulié
Joseph de L'estourbeillon
Alan Le Gourrierec
Maxime Soulié