ATI01 Algo avancé

Prise de note

intel et amd = cpu x86-64
téléphone = cpu ARM
unité de contrôle : lis le code instruction (101101111)
flags = ca indique l'état de la dernière opération, sans historique comme les logs
registre = 64 bits
mme traduit les adresses, cela a était rajouté au moment du passage des cpu a 32 bits

a chaque démarage les cpu refonds l'histoire de 16 bits a 64 bits en passant pour bootloader, le kernel démare en 64 bits (systemd ou pib 1)

uefi démarre a la place du bios "classique"

grub legacy : permet de boot sur un certain os

ne pouvais avoir que 4 partitions (annés 1970-80)

et pour envoir lus en devais définir une mbr pour créer des sous partitions

architecture ARM pas de rétrocompatibité, utilisé sur les raspberry py

w11 permet plus la rétrocoptibité, mais w10 oui

intel n'est pas réellement un vrais cisc

`axel.mura@linux:~/Desktop$ size script text data bss dec hex filename 1509 604 4 2117 845 script`
bss = réserver les octets
data = stocke la variable

voir la traduction de C en ARM

axel.mura@linux:~/Desktop$ objdump -S script

GDB :

```
axel.mura@linux:~/Desktop$ gcc -o -g script script.c
axel.mura@linux:~/Desktop$ gdb ./script
```

n = next pas a pas
c= continue
run = lancer le programme
delete breakpoints

désactiver une breakpont = enable ou disable p

show breakpoints

```
Breakpoint 1, main () at script.c:6
6       for (i=0;i<10; i++){
(gdb) n
7       printf("hello \n");
(gdb) n
hello
6       for (i=0;i<10; i++){
(gdb) n
7       printf("hello \n");
(gdb) n
hello
6       for (i=0;i<10; i++){
(gdb) print i
$1 = 1
(gdb) n
7       printf("hello \n");
(gdb) n
hello
6       for (i=0;i<10; i++){
(gdb) n
7       printf("hello \n");
(gdb) print i
$2 = 3
(gdb) print &i
$3 = (int *) 0x555555755018 <i>
(gdb) print i+42
$4 = 45
(gdb)
```

cpu différent mode de fonctionnement,

- il passe de real mode (16bits sans mmu)
- Protectd mode (32bits avec mmu)
- long mode (64 bits avec mmu).
  différente mode de protections :

- ring 0 (droit d'utiliser toutes les instructions) kernel space : **KERNEL**
- ring 1 (ne sert a rien sur linux mac, et win)
- ring 2 (ne sert à rien sur linux, mac et win)
- ring 3 (droits réduit : safe) user space : **tous les processus**
  a tout moment il veux passer de ring 0 a 1 a 2 ...

IRQ : Une interruption matérielle (en anglais Interrupt ReQuest ou IRQ) est une interruption déclenchée par un périphérique d'entrée-sortie d'un microprocesseur ou d'un microcontrôleur.

l'intérer : les associer avec une fontion, quand on déclenche une IRQ, le cpu passe en RING 0

pour accéder a un fichier : déclanchement irq logicielle

syscall : Un appel système ou syscall est une méthode utilisée par les programmes d'application pour communiquer avec le noyau du système

syscalls LInux => 1920 => AT&T => Maltix/unix

free bsd = laptop
net bsd = server

driver = une fonction en c pour répondre a une besoin matériel

```c
#include <stdio.h>
#include <unistd.h>
#include <sys/types.h>

int main(){
        pid_t A;
        A=fork();
        if(A==0)
                printf("Pouet\n");
        else
                printf("meuh\n");
        return 0;
}
```

```
axel@axel-G3-3500:~/seb$ ./main2
meuh
Pouet
```

tous les process sont créer par un fork

pour vérifier une code :

```
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP05$ objdump -S main |
less
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP05$ strace ./main
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP05$ gdb main
```

```
axel@axel-G3-3500:~/Github/ATI01-Algo_avancee/TP/TP05$ gdb main
GNU gdb (Ubuntu 10.2-0ubuntu1~20.04~1) 10.2
Copyright (C) 2021 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later
<http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<https://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from main...
(gdb) catch syscall write
Catchpoint 1 (syscall 'write' [1])
(gdb) run
Starting program: /home/axel/Github/ATI01-Algo_avancee/TP/TP05/main

Catchpoint 1 (call to syscall write), 0x0000000000450777 in write ()
(gdb) bt
#0  0x0000000000450777 in write ()
#1  0x000000000041b93d in _IO_new_file_write ()
#2  0x000000000041ca91 in _IO_new_do_write ()
#3  0x000000000041c125 in _IO_new_file_xsputn ()
#4  0x000000000041425d in __vfprintf_internal ()
#5  0x00000000004109fc in printf ()
#6  0x0000000000401cde in main () at main.c:5
(gdb)
```

code c en arm dans gdb :

```
(gdb) layout asm
```

Liens :

http://www.gdbtutorial.com/tutorial/how-install-gdb
https://syscalls64.paolostivanin.com/