

[ASM] Systèmes, Microprocesseurs, Architecture (3)

Par Rhaeven, relecture et adaptation par Find3r

- link
- ELF format
- Exécution du programme

```
objdump -d program_name  
readelf
```

```
1  int a = 12;  
2  
3  int foo()  
4  {  
5      return 14;  
6  }  
7  
8  int calc(int a, int b);  
9  
10 int main()  
11 {  
12     printf("%d\n", calc(foo(), a));  
13 }
```

```
1  int calc(int a, int b)  
2  {  
3      int c;  
4  
5      c = a + b;  
6      return c;  
7  }
```

```
// arguments: a=%edi, b=%esi
// return value: %eax
// stack:
//     -8(%rbp)    a
//    -16(%rbp)    b
//    -24(%rbp)    c
```

calc:

```
    push %rtp
    mov %rsp, %rbp
    sub $24, %rsp

    movl %edi, -8(%rbp)
    movl %esi, -16(%rbp)
    movl g, -24(%rbp)

    // addl a, c
    addl -8(%rbp), %edi
    addl %edi, -24(%rbp)
    // addl b, c
    addl -16(%rbp), %edi
    addl %edi, -24(%rbp)

    addl a, c
    addl b, c

    movl -24(%rbp), %eax
    leave
    ret
```

Relocation

lea : Load Effective Address

Calcule une adresse à la place de déréférencer

```
mov %rax, %rdx // move la data
lea %rax, %rdx // met l'adresse calculée dans %rdx

mov -0x10(%rbp), %rdx    %rdx <- local;
lea -0x10(%rbp), %rdx    %rdx <- &local;

lea(%rax, %rax, 2), %rdx    %rdx = %rax * 3;
```

ELF

Elf : Extensible Loading Format

- Exécutable
- Relocatable (.o)
- Dynamiques (.so, etc)
- Coredump (image mémoire d'un programme stocké sur le disque)

Écrit le contenu de la mémoire + registres d'un fichier dans un fichier, pratique pour le debug

2 utilisations de l'ELF :

- Statique : Compilation et edition de lien
- Dynamique : Chargement et execution d'un programme

Commence par une en-tete

Header ELF

```
readelf -h program_name
```

2 tables :

- Table des **sections** : partie statique
- Table des **segments** (program headers) : partie dynamique

```
readelf -l program_name : program headers
```

LOAD : Indique de quoi charger en mémoire, et où (MemSiz)

AUXV
ENVP
ARGV
Argc
RPS ↗

```
readelf -S 42sh : sections
```

Linker : prend les .o et prend les sections bout à bout

Écrire un fichier assembleur

.s : fichier assembleur (pour la machine)

.S : fichier assembleur avec directives de préprocesseur (pour les humains)

```
1  int calc(int a, int b)
2  {
3      int c;
4      c = a + b;
5      return c;
6  }
```

```
.section text
```

```
.global calc // calc est une fonction globale
```

```
.type calc, @function
```

```
calc:
```

```
    xor %eax, %eax
```

```
    // movl g, %eax essaye d'accéder a adresse absolue
```

```
    movl g(%rip), %eax
```

```
    addl %esi, %eax
```

```
    addl %edi, %eax
```

```
    //ret
```

```
    .byte 0xc3
```

```
.Lend_calc:
```

```
    .size calc, .Lend_calc - calc // on déclare la taille de la fonction
```

```
    // .size calc, . - calc avec . l'adresse courante
```

```
.section .data
```

```
.type g, @object
```

```
// .global g si variable non statique
```

```
g:
```

```
    .long 12
```

```
    .size g, . - g
```

Syscall en assembleur

```
1 | #!/bin/sh
2 |
3 | set -e
4 |
5 | gcc -c empty.S -o empty.o
6 | ld -o empty empty.o
```

empty.S :

```
#include <asm/unistd.h>
```

```
.global _start
```

```
_start:
```

```
mov $1, %rdi
```

```
call _exit
```

```
_exit:
```

```
// mov $60, %rax ou 60 = NR_exit
```

```
mov $__NR_exit, %rax
```

```
syscall
```

```
ret
```

/usr/include/asm/unistd_64 : numéros des syscalls

Appeler un syscall

Arch/ABI	Instruction	System call #	Ret val	Ret val2	Error	Notes
alpha	callsys	v0	v0	a4	a3	1, 6
arc	trap0	r8	r0	-	-	
arm/OABI	swi NR	-	a1	-	-	2
arm/EABI	swi 0x0	r7	r0	r1	-	
arm64	svc #0	x8	x0	x1	-	
blackfin	excpt 0x0	P0	R0	-	-	
i386	int \$0x80	eax	eax	edx	-	
ia64	break 0x100000	r15	r8	r9	r10	1, 6
m68k	trap #0	d0	d0	-	-	
microblaze	brki r14,8	r12	r3	-	-	
mips	syscall	v0	v0	v1	a3	1, 6
nios2	trap	r2	r2	-	r7	
parisc	ble 0x100(%sr2, %r0)	r20	r28	-	-	
powerpc	sc	r0	r3	-	r0	1
powerpc64	sc	r0	r3	-	cr0.S0	1
riscv	ecall	a7	a0	a1	-	
s390	svc 0	r1	r2	r3	-	3
s390x	svc 0	r1	r2	r3	-	3
superh	trap #0x17	r3	r0	r1	-	4, 6
sparc/32	t 0x10	g1	o0	o1	psr/csr	1, 6
sparc/64	t 0x6d	g1	o0	o1	psr/csr	1, 6
tile	swint1	R10	R00	-	R01	1
x86-64	syscall	rax	rax	rdx	-	5
x32	syscall	rax	rax	rdx	-	5
xtensa	syscall	a2	a2	-	-	

Passer des arguments au syscall

Arch/ABI	arg1	arg2	arg3	arg4	arg5	arg6	arg7	Notes
alpha	a0	a1	a2	a3	a4	a5	-	
arc	r0	r1	r2	r3	r4	r5	-	
arm/OABI	a1	a2	a3	a4	v1	v2	v3	
arm/EABI	r0	r1	r2	r3	r4	r5	r6	
arm64	x0	x1	x2	x3	x4	x5	-	
blackfin	R0	R1	R2	R3	R4	R5	-	
i386	ebx	ecx	edx	esi	edi	ebp	-	
ia64	out0	out1	out2	out3	out4	out5	-	
m68k	d1	d2	d3	d4	d5	a0	-	
microblaze	r5	r6	r7	r8	r9	r10	-	
mips/o32	a0	a1	a2	a3	-	-	-	1
mips/n32,64	a0	a1	a2	a3	a4	a5	-	
nios2	r4	r5	r6	r7	r8	r9	-	
parisc	r26	r25	r24	r23	r22	r21	-	
powerpc	r3	r4	r5	r6	r7	r8	r9	
powerpc64	r3	r4	r5	r6	r7	r8	-	
riscv	a0	a1	a2	a3	a4	a5	-	
s390	r2	r3	r4	r5	r6	r7	-	
s390x	r2	r3	r4	r5	r6	r7	-	
superh	r4	r5	r6	r7	r0	r1	r2	
sparc/32	o0	o1	o2	o3	o4	o5	-	
sparc/64	o0	o1	o2	o3	o4	o5	-	
tile	R00	R01	R02	R03	R04	R05	-	
x86-64	rdi	rsi	rdx	r10	r8	r9	-	
x32	rdi	rsi	rdx	r10	r8	r9	-	
xtensa	a6	a3	a4	a5	a8	a9	-	