NoSQL

Alejandro Osornio

2023-11-21

SQL

- Las computadoras son *software* y *hardware*
- Escribir programas para la computadora implica escribir instrucciones
- Un programa no es más que una secuencia de instrucciones para el procesador.
- El lenguaje ensamblador es el de **más bajo nivel** que permite escribir programas para una computadora, de la forma más cercana al hardware, usando texto plano que es compilado.

Assembler

- El ensamblador es el programa encargado de **compilar un programa escrito en el lenguaje ensamblador**, es decir, traducir las instrucciones dadas en forma de texto plano, entendible por humanos, a código binario, el lenguaje de máquina que el procesador puede manejar y ejecutar.
- Al archivo compilado se le llama binario o ejecutable.

El proyecto

```
section .data
msg db "Hola mundo!", 0xA

section .text
global _start

_start:
  mov rax, 0x1
  lea rdi, [rip + msg]
  mov rsi, 0xC
  syscall
```

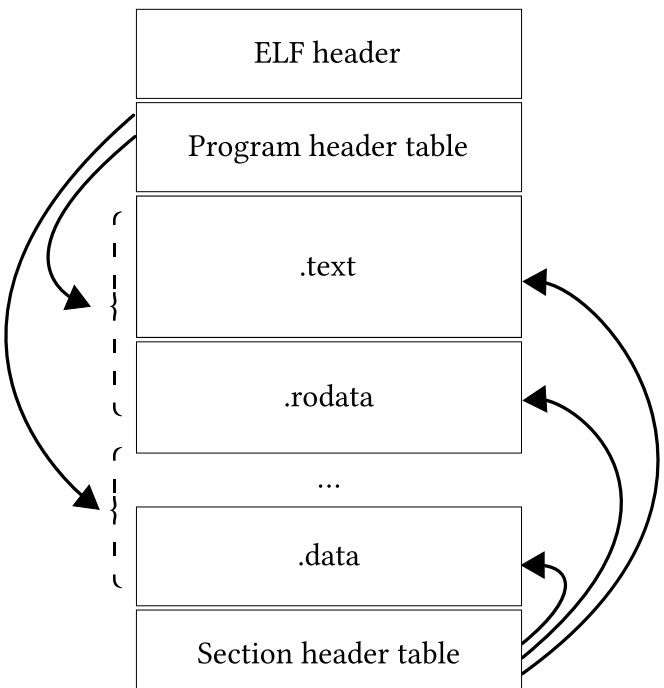
```
01001000 01101111 01101100
01100001 00100000 01110000
01101111 01110010 00100000
01100110 01100001 01110110
01101111 01110010 00100000
01101110 01101111 00100000
01110100 01110010 01100001
01110011 01110000 01101111
01110010 00100000 01100110
01100001 01110110 01101111
01110010 00100000 01101110
```

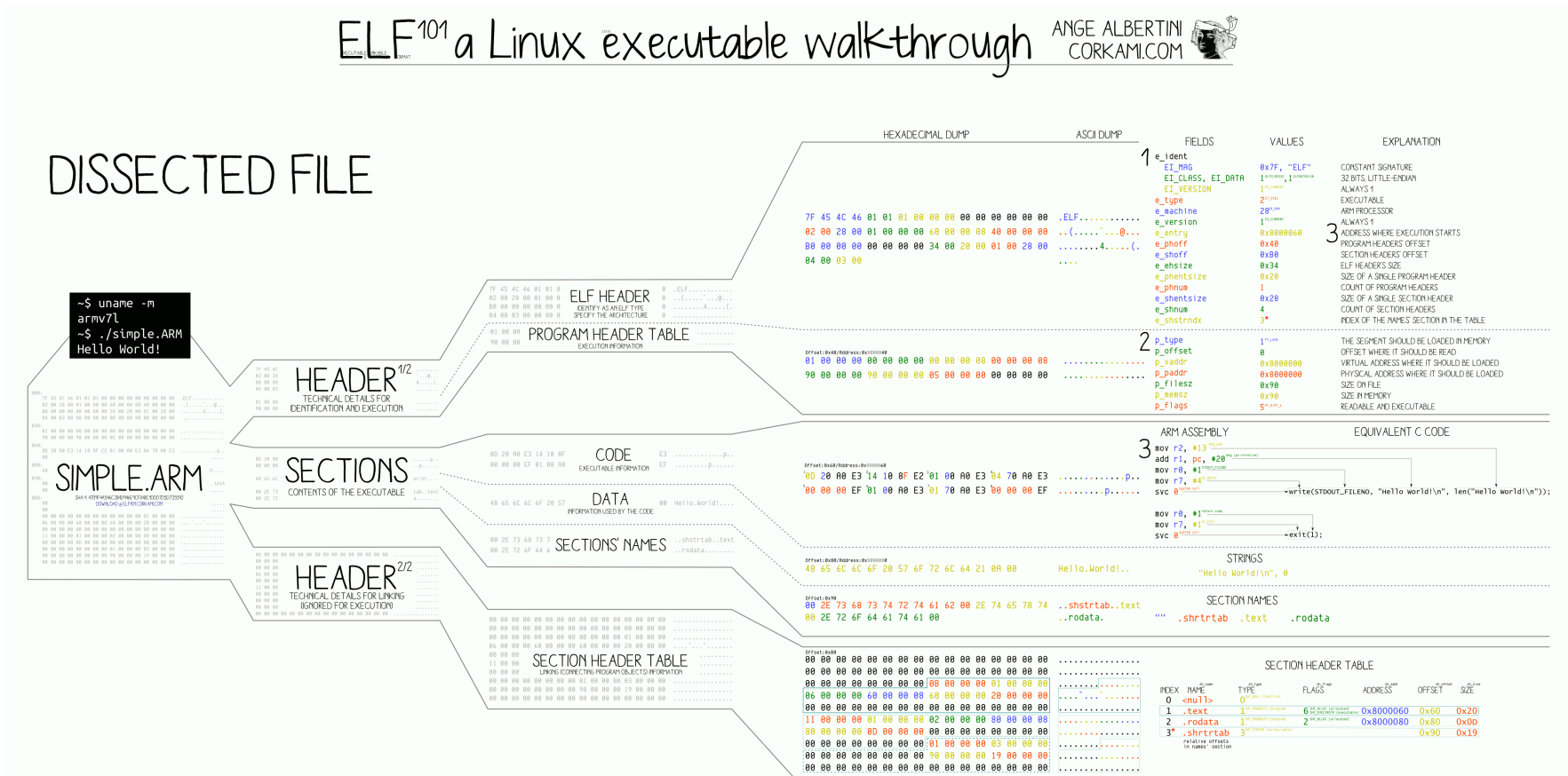
Binario

- El sistema operativo tiene formas determinadas de ejecutar código, por ejemplo, **no cualquier archivo con 1s y 0s de información se ejecuta** por que sí.
- Se le debe dar de la forma y lugar correcto la información necesaria para que el sistema operativo inicie un proceso con código
- En linux, el formato para darle programas al sistema operativo es el ELF, **Executable and Linkable Format**.

ELF

- Un archivo con el formato **ELF** hace que la computadora realice tareas indicadas según instrucciones codificadas.
- Ejecución:
  - La firma del archivo se parsea y carga
  - Las secciones de información especificadas se mapean a la memoria
  - Se pone `CIR` en la dirección de entrada




ELF 101 a Linux executable walkthrough
ANGE ALBERTINI
CORKAMI.COM
DISSECTED FILE
~$ uname -m
armv7l
~$ ./simple.ARM
Hello World!
SIMPLE.ARM
HEADER 1/2
TECHNICAL DETAILS FOR IDENTIFICATION AND EXECUTION
SECTIONS
CONTENTS OF THE EXECUTABLE
HEADER 2/2
TECHNICAL DETAILS FOR LINKING (IGNORED FOR EXECUTION)
ELF HEADER
IDENTIFY AS AN ELF TYPE
SPECIFY THE ARCHITECTURE
PROGRAM HEADER TABLE
EXECUTION INFORMATION
CODE
EXECUTABLE INFORMATION
DATA
INFORMATION USED BY THE CODE
SECTIONS' NAMES
SECTION HEADER TABLE
LINKING (CONNECTING PROGRAM OBJECTS) INFORMATION
HEXADECIMAL DUMP
ASCII DUMP
FIELDS
VALUES
EXPLANATION
1
2
3
ARM ASSEMBLY
EQUIVALENT C CODE
STRINGS
SECTION NAMES
SECTION HEADER TABLE

m4b/faerie

```
let mut obj = ArtifactBuilder::new(triple!("x86_64-unknown-...")).finish();

obj.declarations(
    [("main",    Decl::function().global().into()),
    ("printf",   Decl::function_import().into()) ].iter().cloned()
)?;

obj.define("main", vec![0x55, 0xc3])?;
obj.define("str.1", b"deadbeef: 0x%x\n\0".to_vec())?;

obj.link(Link { from: "main", to: "printf", at: 29 })?;
obj.link(Link { from: "deadbeef", to: "DEADBEEF", at: 7 })?;

obj.write(file)?;
```

Parse

- Primero debemos pasar el texto del archivo a una representación que nos permita identificar y trabajar las instrucciones desde el software. A este proceso se le llama generar el **Abstract Syntax Tree o AST**.

```
enum Line {
  Mnemonic(Mnemonic),
  DataDefine(DataDefine),
  Extern(String),
  Comment(String),
  Function(Fun),
  Global(String),
  Empty,
}
```

Parse

```
extern scanf                    ;;  { line: Line::Extern(_),     line_num: 1 }
extern exit                     ;;  { line: Line::Extern(_),     line_num: 2 }
                                ;;  { line: Line::Empty,          line_num: 3 }
numb dw 0x0                     ;;  { line: Line::DataDefine(_), line_num: 4 }
scan db "%d", 0                 ;;  { line: Line::DataDefine(_), line_num: 5 }
                                ;;  { line: Line::Empty,          line_num: 6 }
main:                           ;;  { line: Line::Function(_),   line_num: 7 }
    mov rax, 0                  ;;  { line: Line::Mnemonic(_),   line_num: 8 }
    lea rdi, [rip + scan]       ;;  { line: Line::Mnemonic(_),   line_num: 9 }
    lea rsi, [rip + numb]       ;;  { line: Line::Mnemonic(_),   line_num: A }
    call scanf                  ;;  { line: Line::Mnemonic(_),   line_num: B }
    call exit                   ;;  { line: Line::Mnemonic(_),   line_num: C }
```

Codificar

```
fn assemble(&mut self) -> Result<&mut Vec<u8>> {
        self.bytecode.clear();

        let lines = take(&mut self.lines);
        for line in lines.into_iter() {
            if let Err(e) = self.assemble_line(line.line) {
                bail!("Error on line {}: {}", line.line_num, e);
            }
        }

        Ok(&mut self.bytecode)
    }
```

Codificar

Encoding:

Prefix	**Opcode**	ModR/M	**SIB**	Displacement	Immediate

ModR/M:

Mod	Mod	**Reg**	**Reg**	**Reg**	R/M	R/M	R/M

SIB

Scale	Scale	**Index**	**Index**	**Index**	Base	Base	Base

Gracias!

- Wikipedia (2023-07-13). *Executable and Linkable Format*. https://en.wikipedia.org/w/index.php?title=Executable_and_Linkable_Format&oldid=1162401948
- Wikipedia (2023-07-13). *Executable*. https://en.wikipedia.org/w/index.php?title=Executable&oldid=1155627042
- X86-64 Instruction Encoding. *osdev.org*. https://wiki.osdev.org/X86-64_Instruction_Encoding#Table10Note2 el día 13/07/2023
- Irma Patricia Quiroga. (2010). Arquitectura de computadoras. Alfaomega.
- Morris Mano, M. (1994). Arquitectura de computadores. Prentice-Hall Hispanoamericana.
- Intel Corporation (2023). Intel® 64 and IA-32 Architectures Software Developer's Manual. Combined Volumes: 1-4