REVERSE ENGINEERING – CLASS 0x02

THE STRUCTURE OF ELF FILES

Cristian Rusu

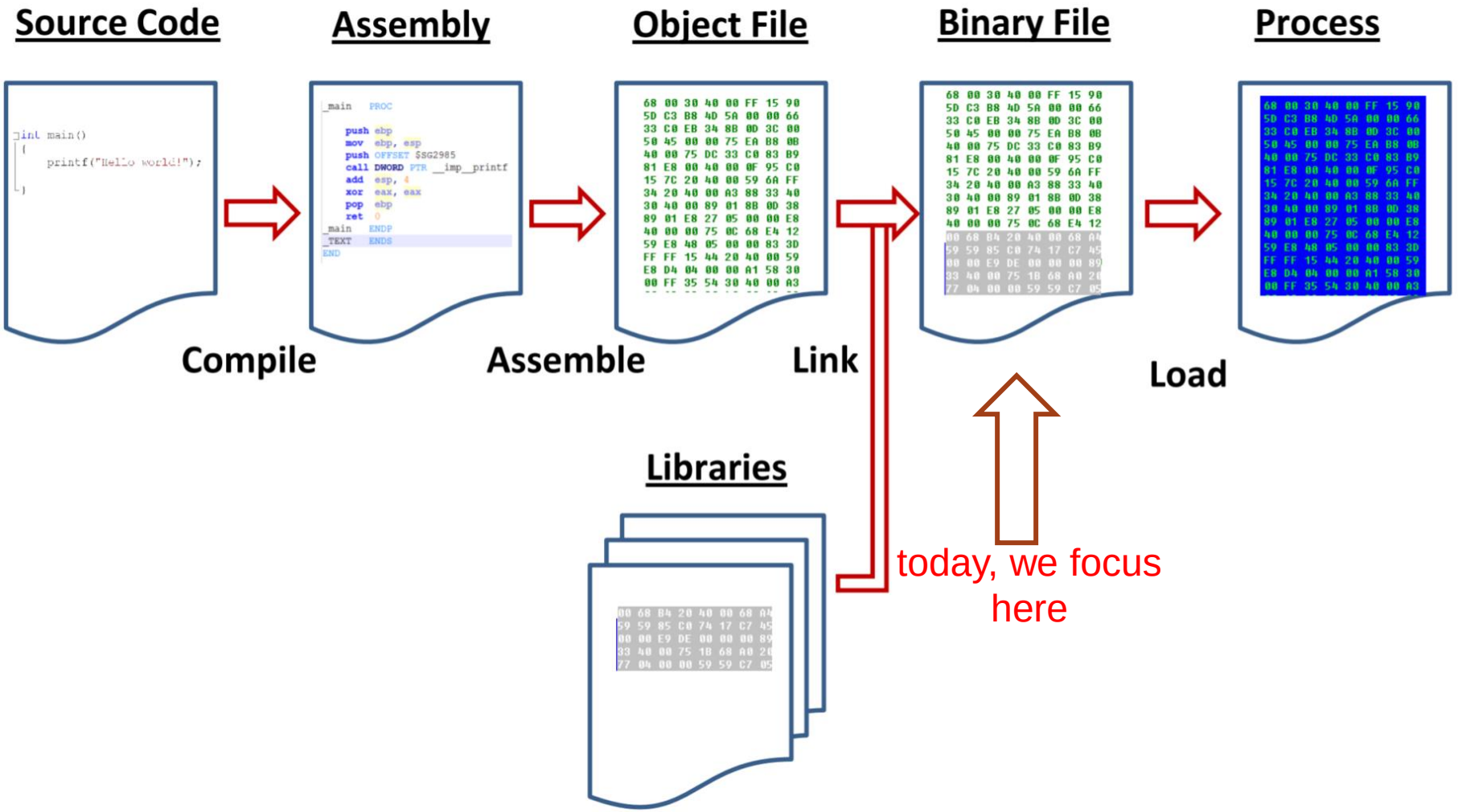
LAST TIME

- **Assembly crash course**
- **compiler tricks**
- **lab session on assembly primer**

TODAY

- assembly in context
- the structure of binary files
- study of the ELF binaries
- PE for next week

FROM SOURCE CODE TO EXECUTION



BINARY FILES

- **ELF/SO**
- **PE/DLL**
- **WASM**
- **machine code (assembly translated to CPU readable instructions) is only part of the executable**
- **all of them have some particular structure we need to understand to in order to execute the binary (ABI)**

ELF BINARY

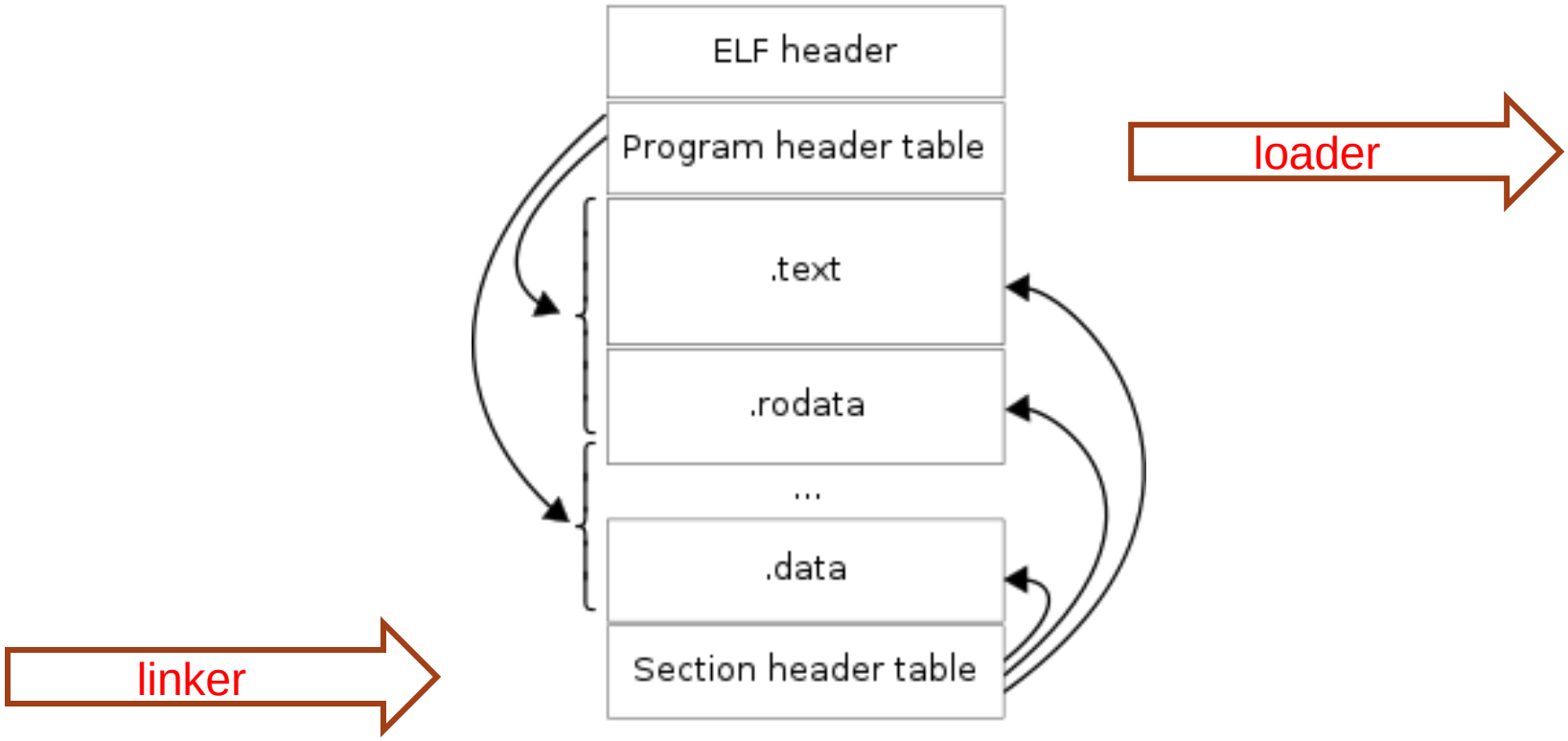
- **Executable and Linkable Format (ELF)**
  - Header
  - Content
    - Segments
    - Sections
    - Instructions/Data

- **relatively recently introduced, from 1999 (standard from ’80)**

- **standard for the Linux OS**
  - binary executables
  - libraries
  - etc.

ELF BINARY

- **structure of ELF binaries**

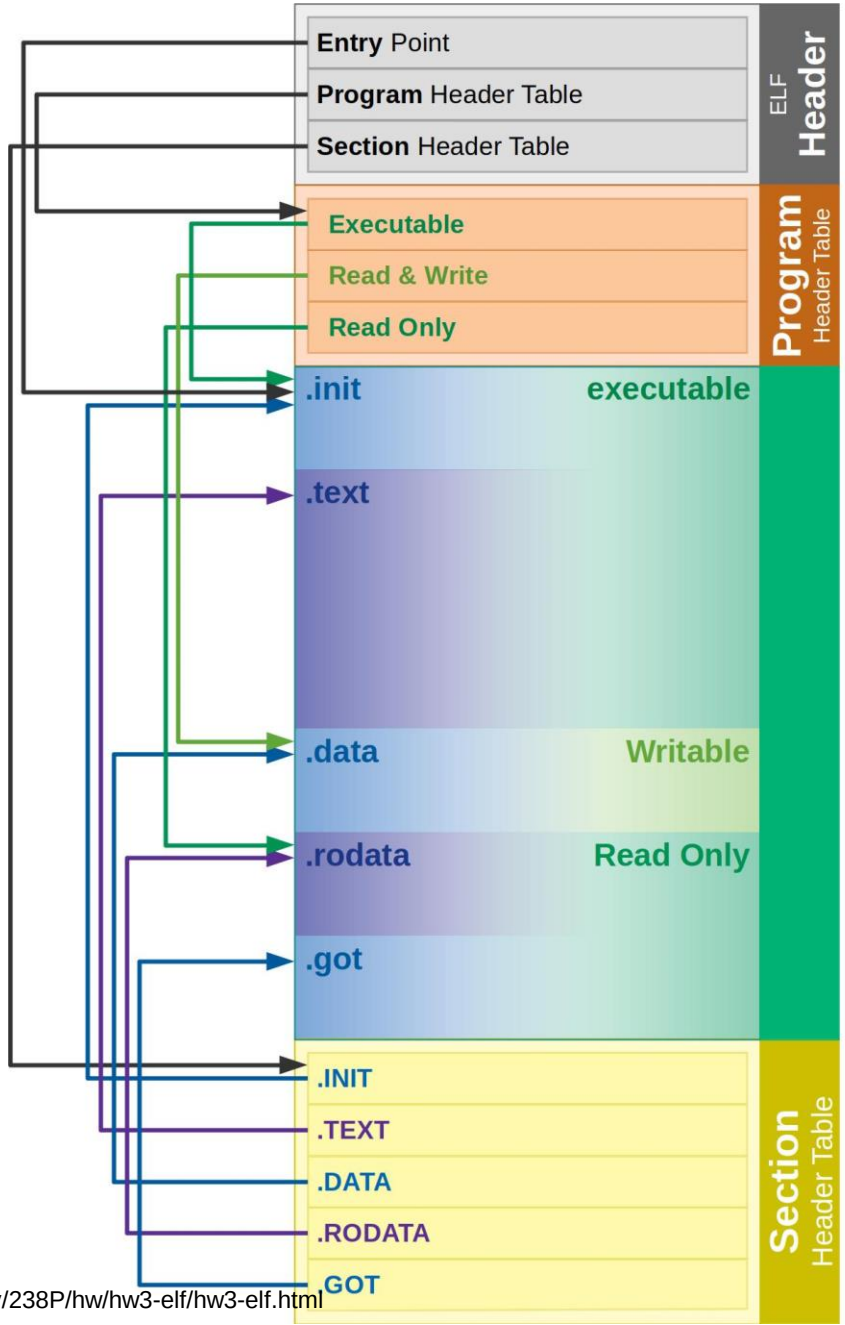


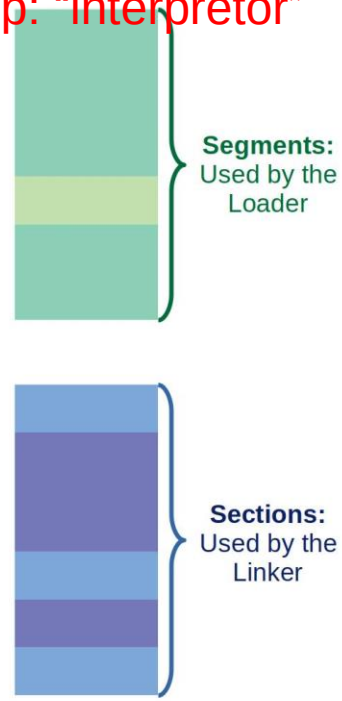
Linker: places pointers to *sections* from the binary, not relevant at execution
Loader: places pointers to *segments* from the binary, used at execution

https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

ELF BINARY

.text: machine code
.rodata: readonly data
.data: initialized data
.bss: uninitialized data
.init: init before main()
.plt: Procedure Linking Table
.got: Global Offset Table
.interp: “interpretor”





https://www.ics.uci.edu/~aburtsev/238P/hw/hw3-elf/hw3-elf.html

READELF SECTIONS

- describes *program headers*

our binary is called a2.out

```
└─$ readelf --program-headers a2.out

Elf file type is DYN (Shared object file)
Entry point 0x3a17e0
There are 11 program headers, starting at offset 64

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  PHDR           0x0000000000000040 0x0000000000000040 0x0000000000000040
                 0x0000000000000268 0x0000000000000268  R      0x8
  INTERP         0x00000000000002a8 0x00000000000002a8 0x00000000000002a8
                 0x000000000000001c 0x000000000000001c  R      0x1
      [Requesting program interpreter: /lib64/ld-linux-x86-64.so.2]
  LOAD           0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x00000000000006c8 0x00000000000006c8  R      0x1000
  LOAD           0x0000000000001000 0x0000000000001000 0x0000000000001000
                 0x000000000003a093d 0x000000000003a093d  R E    0x1000
  LOAD           0x000000000003a2000 0x000000000003a2000 0x000000000003a2000
                 0x00000000001071a0 0x00000000001071a0  R      0x1000
  LOAD           0x00000000004a9de0 0x00000000004aade0 0x00000000004aade0
                 0x0000000000000280 0x0000000000000288  RW     0x1000
  DYNAMIC        0x00000000004a9df8 0x00000000004aadf8 0x00000000004aadf8
                 0x00000000000001e0 0x00000000000001e0  RW     0x8
  NOTE           0x00000000000002c4 0x00000000000002c4 0x00000000000002c4
                 0x0000000000000044 0x0000000000000044  R      0x4
```

...

READELF SECTIONS

- descrie *program headers*

```
  DYNAMIC        0x00000000004a9df8 0x00000000004aadf8 0x00000000004aadf8
                 0x00000000000001e0 0x00000000000001e0  RW     0x8
  NOTE           0x00000000000002c4 0x00000000000002c4 0x00000000000002c4
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_EH_FRAME   0x00000000004a9010 0x00000000004a9010 0x00000000004a9010
                 0x0000000000000044 0x0000000000000044  R      0x4
  GNU_STACK      0x0000000000000000 0x0000000000000000 0x0000000000000000
                 0x0000000000000000 0x0000000000000000  RW     0x10
  GNU_RELRO      0x00000000004a9de0 0x00000000004aade0 0x00000000004aade0
                 0x0000000000000220 0x0000000000000220  R      0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .interp .note.gnu.build-id .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version .
gnu.version_r .rela.dyn .rela.plt
   03     .init .plt .plt.got .text .fini
   04     .rodata .eh_frame_hdr .eh_frame
   05     .init_array .fini_array .dynamic .got .got.plt .data .bss
   06     .dynamic
   07     .note.gnu.build-id .note.ABI-tag
   08     .eh_frame_hdr
   09
   10     .init_array .fini_array .dynamic .got
```

READELF SECTIONS

- descrie *program headers*

```
└─$ readelf --section-headers a2.out
There are 37 section headers, starting at offset 0×4aaed0:

Section Headers:
  [Nr] Name              Type             Address           Offset
       Size              EntSize          Flags  Link  Info  Align
  [ 0]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         00000000000002a8  000002a8
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.gnu.bu[ ... ] NOTE           00000000000002c4  000002c4
       0000000000000024  0000000000000000   A       0     0     4
  [ 3] .note.ABI-tag     NOTE             00000000000002e8  000002e8
       0000000000000020  0000000000000000   A       0     0     4
  [ 4] .gnu.hash         GNU_HASH         0000000000000308  00000308
       0000000000000024  0000000000000000   A       5     0     8
  [ 5] .dynsym           DYNSYM           0000000000000330  00000330
       0000000000000138  0000000000000018   A       6     1     8
  [ 6] .dynstr           STRTAB           0000000000000468  00000468
       00000000000000a3  0000000000000000   A       0     0     1
  [ 7] .gnu.version      VERSYM           000000000000050c  0000050c
       000000000000001a  0000000000000002   A       5     0     2
  [ 8] .gnu.version_r    VERNEED          0000000000000528  00000528
       0000000000000020  0000000000000000   A       6     1     8
  [ 9] .rela.dyn         RELA             0000000000000548  00000548
```

...

READELF SECTIONS

- descrie *program headers*

```
  [27] .debug_aranges    PROGBITS         0000000000000000  004aa07f
       0000000000000030  0000000000000000           0     0     1
  [28] .debug_info       PROGBITS         0000000000000000  004aa0af
       0000000000000064  0000000000000000           0     0     1
  [29] .debug_abbrev     PROGBITS         0000000000000000  004aa113
       000000000000004d  0000000000000000           0     0     1
  [30] .debug_line       PROGBITS         0000000000000000  004aa160
       0000000000000077  0000000000000000           0     0     1
  [31] .debug_str        PROGBITS         0000000000000000  004aa1d7
       000000000000012c  0000000000000001  MS       0     0     1
  [32] .debug_loc        PROGBITS         0000000000000000  004aa303
       0000000000000059  0000000000000000           0     0     1
  [33] .debug_ranges     PROGBITS         0000000000000000  004aa35c
       0000000000000020  0000000000000000           0     0     1
  [34] .symtab           SYMTAB           0000000000000000  004aa380
       0000000000000768  0000000000000018          35    54     8
  [35] .strtab           STRTAB           0000000000000000  004aaae8
       0000000000000283  0000000000000000           0     0     1
  [36] .shstrtab         STRTAB           0000000000000000  004aad6b
       0000000000000160  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  l (large), p (processor specific)
```

READELF HEADER

- describes the header

```
└─$ readelf -h a2.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x3a17e0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4894416 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         11
  Size of section headers:           64 (bytes)
  Number of section headers:         37
  Section header string table index: 36
```

.ELF...

READELF HEADER

- describes the header

```
└─$ readelf -h a2.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x3a17e0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4894416 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         11
  Size of section hea
  Number of section he
  Section header stri
```

```
└─$ hexdump a2.out -n 64
0000000 457f 464c 0102 0001 0000 0000 0000 0000
0000010 0003 003e 0001 0000 17e0 003a 0000 0000
0000020 0040 0000 0000 0000 aed0 004a 0000 0000
0000030 0000 0000 0040 0038 000b 0040 0025 0024
0000040
```

.ELF...

READELF HEADER

- describes the header

```
└─$ readelf -h a2.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x3a17e0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4894416 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         11
  Size of section headers:           64 (bytes)
  Number of section headers:         37
  S └─$ file a2.out
    a2.out: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpr
    eter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=18fbba2db7d9c5002d78d2b718dfab2e8ba84f3c, f
    or GNU/Linux 3.2.0, with debug_info, not stripped
```

.ELF...

ELF BY HAND

```
# >>>>>>>>>>>>> ELF FILE HEADER <<<<<<<<<<<<<
                # All numbers (except in names) are in base sixteen (hexadecimal)
                # 00 <- number of bytes listed so far
7F 45 4C 46     # 04 e_ident[EI_MAG]: ELF magic number
01              # 05 e_ident[EI_CLASS]: 1: 32-bit, 2: 64-bit
   01           # 06 e_ident[EI_DATA]: 1: little-endian, 2: big-endian
      01        # 07 e_ident[EI_VERSION]: ELF header version; must be 1
         00     # 08 e_ident[EI_OSABI]: Target OS ABI; should be 0

00              # 09 e_ident[EI_ABIVERSION]: ABI version; 0 is ok for Linux
   00 00 00     # 0C e_ident[EI_PAD]: unused, should be 0
00 00 00 00     # 10

02 00           # 12 e_type: object file type; 2: executable
      03 00     # 14 e_machine: instruction set architecture; 3: x86, 3E: amd64
01 00 00 00     # 18 e_version: ELF identification version; must be 1

54 80 04 08     # 1C e_entry: memory address of entry point (where process starts)
34 00 00 00     # 20 e_phoff: file offset where program headers begin

00 00 00 00     # 24 e_shoff: file offset where section headers begin
00 00 00 00     # 28 e_flags: 0 for x86

34 00           # 2A e_ehsize: size of this header (34: 32-bit, 40: 64-bit)
      20 00     # 2C e_phentsize: size of each program header (20: 32-bit, 38: 64-bit)
01 00           # 2E e_phnum: #program headers
      28 00     # 30 e_shentsize: size of each section header (28: 32-bit, 40: 64-bit)

00 00           # 32 e_shnum: #section headers
      00 00     # 34 e_shstrndx: index of section header containing section names

# >>>>>>>>>>>>> ELF PROGRAM HEADER <<<<<<<<<<<<<

01 00 00 00     # 38 p_type: segment type; 1: loadable

54 00 00 00     # 3C p_offset: file offset where segment begins
54 80 04 08     # 40 p_vaddr: virtual address of segment in memory (x86: 08048054)

00 00 00 00     # 44 p_paddr: physical address of segment, unspecified by 386 supplement
0C 00 00 00     # 48 p_filesz: size in bytes of the segment in the file image ############

0C 00 00 00     # 4C p_memsz: size in bytes of the segment in memory; p_filesz <= p_memsz
05 00 00 00     # 50 p_flags: segment-dependent flags (1: X, 2: W, 4: R)

00 10 00 00     # 54 p_align: 1000 for x86

# >>>>>>>>>>>>> PROGRAM SEGMENT <<<<<<<<<<<<<

B8 01 00 00 00  # 59 eax <- 1 (exit)
BB 00 00 00 00  # 5E ebx <- 0 (param)
CD 80           # 60 syscall >> int 80
```

ELF header + machine code for EXIT program

Handmade Linux x86 executables, https://www.youtube.com/watch?v=XH6jDiKxod8
https://dacvs.neocities.org/1exit

AN EXERCISE

```
└─$ ls -al a2.out
-rwxr-xr-x 1 kali kali 4896784 Jan 20 16:52 a2.out
```

```
└─$ readelf -h a2.out
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x3a17e0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          4894416 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         11
  Size of section headers:           64 (bytes)
  Number of section headers:         37
  Section header string table index: 36
```

where is the header entry for the .text section?

AN EXERCISE

- **readelf –S a2.out**

```
[14] .text             PROGBITS         00000000000010b0  000010b0
     00000000003a0881  0000000000000000  AX       0     0     16
```

- **start of section header (StOSH): 4894416 bytes (sau 0x4AAED0)**
- **index of section .text: 14**
- **size of section headers (SiOSH): 64 bytes**

- **header for .text starts at:**
  - StOSH + 14 x SiOSH = 4895312 = 0x4AB250
  - there is a structure there which described the properties
    - https://github.com/torvalds/linux/blob/master/include/uapi/linux/elf.h
    - struct is *elf32_shdr* or *elf64_shdr*

EXECUTING A STATIC BINARY

- **syscall for execution**
  - EXEC
- **reads the header of the binary**
- **all LOAD directive are executed**
- **execution resumes at *entry point address* (*_start* and then *main()*)**

https://man7.org/linux/man-pages/man3/exec.3.html

STATIC OR DYNAMIC BINARY

- **simbols are references (to functions and variable) in binaries**
  - nm a2.out
  - in gdb when you do „break main”, main here is a symbol
    - function name is there in the file, but not essential to execution
  - remove the symbols by „strippping” the binary
    - stripping symbols
    - debug and RE are much more difficult
    - binaries with smaller size

- **static linking**
  - libraries symbols are included in the binary at link time

- **dynamic linking**
  - links to symbols are added by linker and the loader resolves the links
  - resolving symbols at runtime

```
└─$ file a2.out
a2.out: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interprete
r /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=18fbba2db7d9c5002d78d2b718dfab2e8ba84f3c, for GNU
/Linux 3.2.0, with debug_info, not stripped
```

gcc test.c -o test -static

STATIC OR DYNAMIC BINARY

- **dynamic linking**
  - for example: libc.so
  - done dynamically by linker
  - Machine code is in a shared memory location

  - when do you compute symbol addresses? *binding*
    - when binary is executed *immediate binding*
    - when symbol is used for the first time *lazy binding*

  - shared libraries
    - lib + name + -major + .minor + so
      - libc-2.31.so
    - lib + name + .so + major
      - libc.so.6

STATIC OR DYNAMIC BINARY

- **for these reasons, multiple running times are affected**
  - compile time
    - one time
    - codul este absolut (*absolute code*)
  - load time
    - each time we execute the binary
    - relocatable code
    - some addresses are computed when loading the binary
  - execute time
    - affected by *lazy binding*

STATIC OR DYNAMIC BINARY

- **an issue that can create confusion**

- **libraries can be of two types:**
  - static
    - library is added at compile time
  - dynamic/shared
    - library is linked when executed
    - no need to recompile
    - is placed in *shared memory*
    - *Position Independent Code* (*Position Independent Execution*)
      - *Global Offset Table*

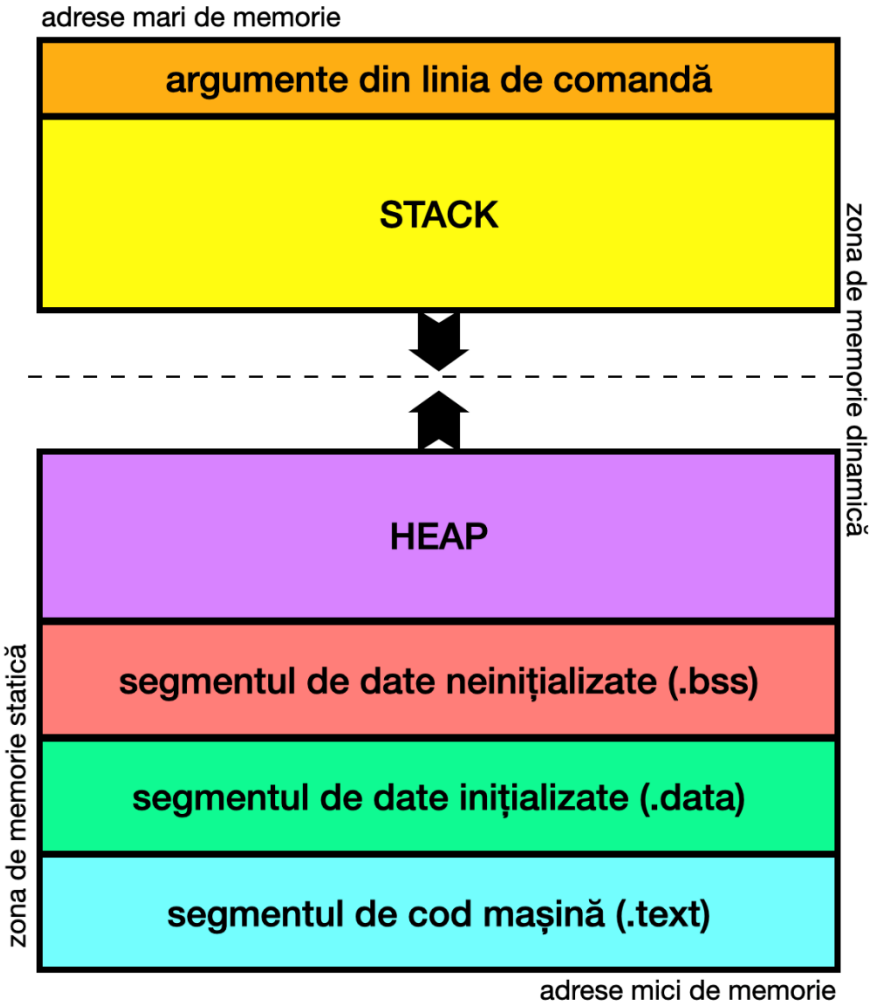
STATIC OR DYNAMIC BINARY

- **PIE vs. NO PIE**

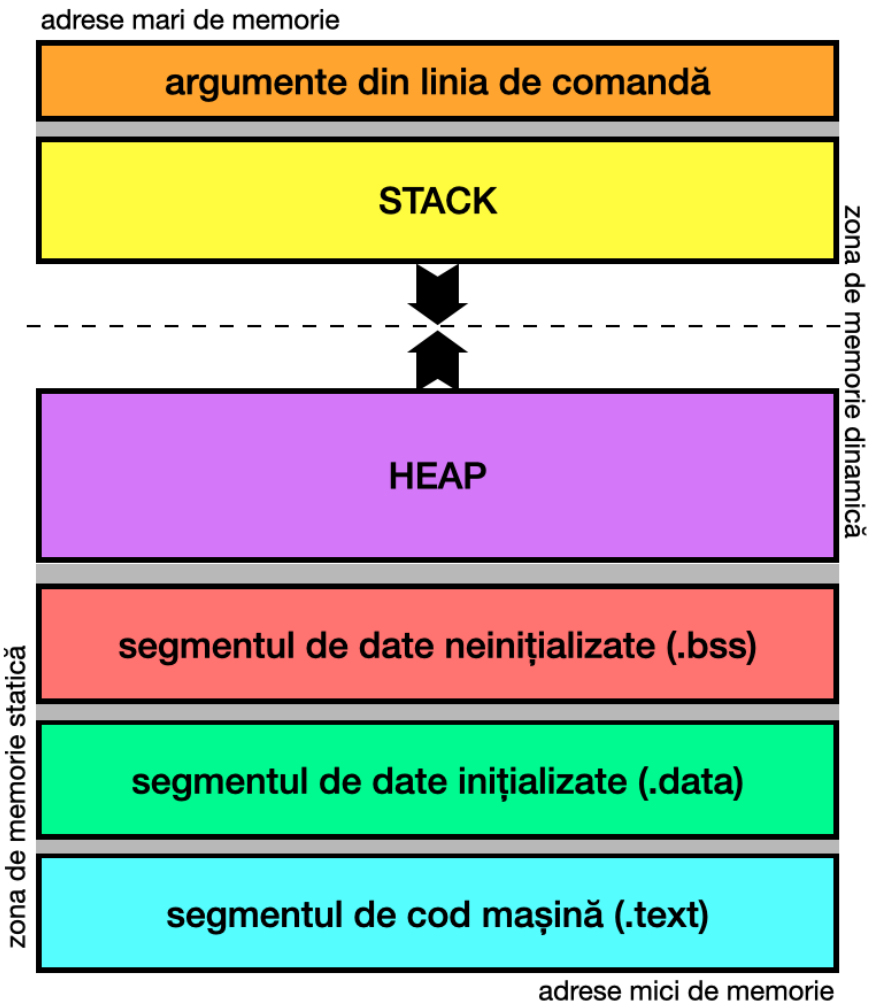
```
┌──(kali㉿kali)-[~]
└─$ gcc write.c -o write -no-pie

┌──(kali㉿kali)-[~]
└─$ ./write
hello!

┌──(kali㉿kali)-[~]
└─$ file write
write: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /l
ib64/ld-linux-x86-64.so.2, BuildID[sha1]=e990629e0423ecf432dd3e0d6f1afe6e4532bc5d, for GNU/Lin
ux 3.2.0, not stripped

┌──(kali㉿kali)-[~]
└─$ gcc write.c -o write

┌──(kali㉿kali)-[~]
└─$ ./write
hello!

┌──(kali㉿kali)-[~]
└─$ file write
write: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter
 /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=cb9a8367c4d68d2555b21eb6838241601e3fcd78, for GNU/
Linux 3.2.0, not stripped
```

BINARE STATICE ȘI DINAMICE

- **PIE vs. NO PIE**



NO PIE



PIE

WHAT WE DID TODAY

- **ELF binaries**
  - readelf
  - objdump
  - nm
- **static and dynamic binaries**

.

NEXT TIME …

- **Windows binaries**
- **Focus on dissasembly**
- **IDA**

REFERENCES

- In-depth: ELF - The Extensible & Linkable Format, https://www.youtube.com/watch?v=nC1U1LJQL8o
- Handmade Linux x86 executables, https://www.youtube.com/watch?v=XH6jDiKxod8
- Creating and Linking Static Libraries on Linux with gcc, https://www.youtube.com/watch?v=t5TfYRRHG04
- Creating and Linking Shared Libraries on Linux with gcc, https://www.youtube.com/watch?v=mUbWcxSb4fw
- Performance matters, https://www.youtube.com/watch?v=r-TLSBdHe1A

.