

# **NEU CY 5770 Software Vulnerabilities and Security**

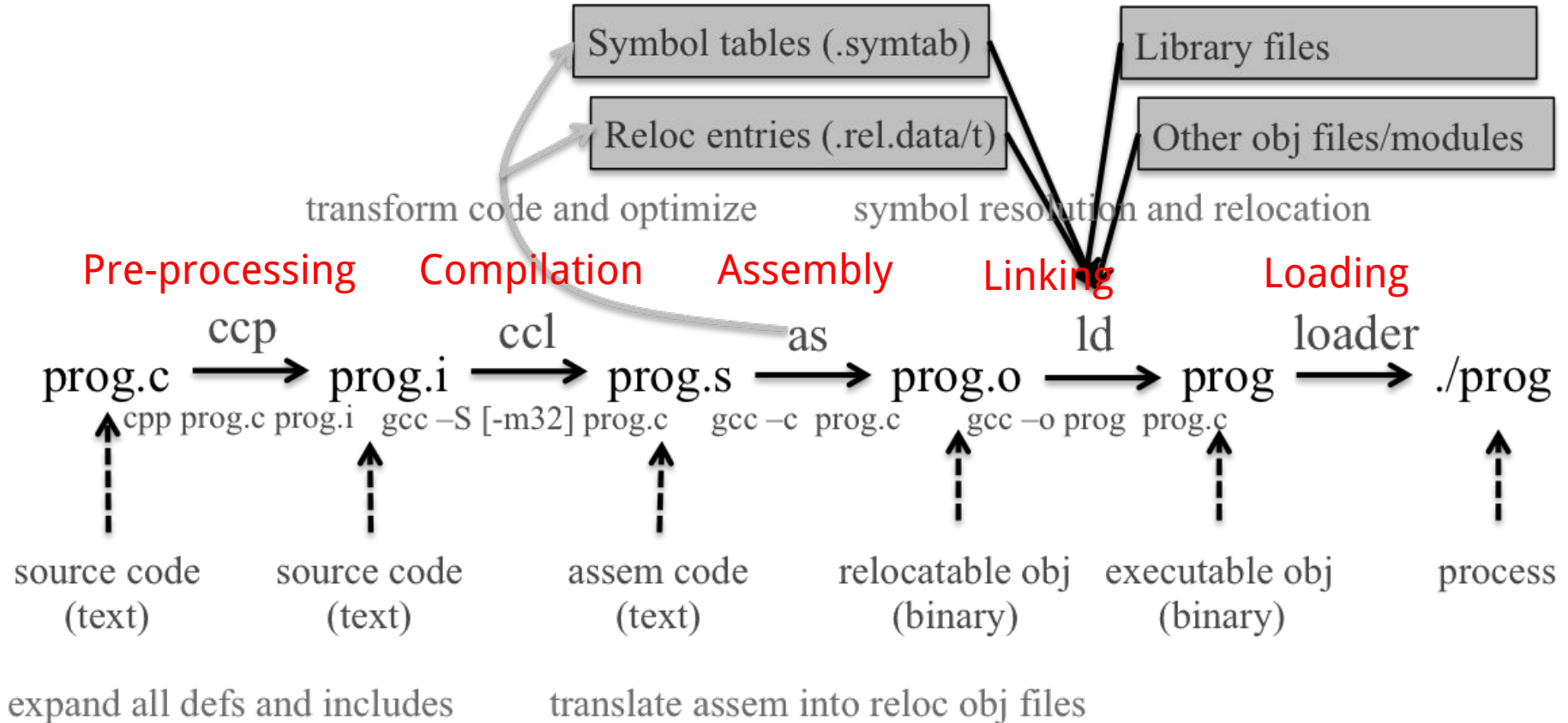
Instructor: Dr. Ziming Zhao

# Agenda

1. Background knowledge
  - a. Compiler, linker, loader
  - b. x86 and x86-64 architectures and ISA
  - c. ARM ISA
  - d. Linux fundamentals
    - i. Linux file permissions
    - ii. Set-UID programs
    - iii. Memory map of a Linux process
    - iv. System calls
    - v. Piping
    - vi. Environment and Shell variables
    - vii. ELF files
    - viii. Reverse engineering tools

# **Background Knowledge: Compiler, linker, and loader**

# From a C program to a process



# A Shell in a Nutshell

```
int pid = fork();  
  
if (pid == 0) {  
    // I am the child process  
  
    exec("ls"); }  
else if (pid == -1)  
{  
    // fork failed  
}  
else {  
    // I am the parent; continue my business being a cool program  
    // I could wait for the child to finish if I want  
}
```

<https://github.com/kamalmarhubi/shell-workshop>

# Loading and Executing a Binary Program on Linux

Validation (permissions, memory requirements etc.)

Operating system starts by setting up a new process for the program to run in, including a virtual address space.

The operating system maps an interpreter into the process's virtual memory.

# Interpreter, e.g., `/lib/ld-linux.so` in Linux

The interpreter loads the binary into its virtual address space (the same space in which the interpreter is loaded).

It then parses the binary to find out (among other things) which dynamic libraries the binary uses.

The interpreter maps these into the virtual address space (using *mmap* or an equivalent function) and then performs any necessary last-minute relocations in the binary's code sections to fill in the correct addresses for references to the dynamic libraries.

1. Copying the command-line arguments on the stack
2. Initializing registers (e.g., the stack pointer)
3. Jumping to the program entry point (`_start`)

# Compiling a C program behind the scene (add\_32 add\_64)

add.c

```
#include "add.h"

#define BASE 50

int add(int a, int b)
{ return a + b +
  BASE;}
```

add.h

```
#ifndef ADD_H
#define ADD_H

int add(int, int);

#endif
```

main.c

```
/* This program has an integer overflow vulnerability. */
#include "add.h"
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define USAGE "Add two integers with 50. Usage: add a b\n"

int main(int argc, char *argv[])
{
    int a = 0;
    int b = 0;

    if (argc != 3)
    {
        printf(USAGE);
        return 0;}

    a = atoi(argv[1]);
    b = atoi(argv[2]);
    printf("%d + %d + 50 = %d\n", a, b, add(a, b));
}
```

```
gcc -Wall -save-temps -P -m32 -O2 add.c main.c -o add_32
```

```
gcc -Wall -save-temps -P -O2 add.c main.c -o add_64
```



# **Background Knowledge:** **x86 architecture**

# Data Types

There are 5 integer data types:

Byte – 8 bits.

Word – 16 bits.

Dword, Doubleword – 32 bits.

Quadword – 64 bits.

Double quadword – 128 bits.

# Endianness

- Little Endian (Intel, ARM)

Least significant byte has lowest address

Dword address: 0x0

Value: 0x78563412

- Big Endian

Least significant byte has highest address

Dword address: 0x0

Value: 0x12345678

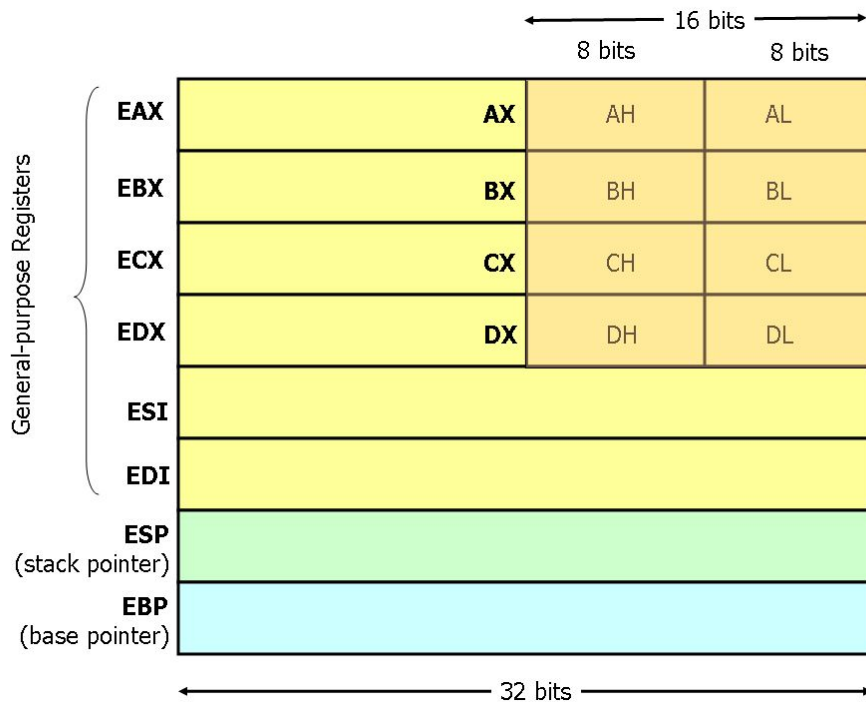
Address 0	0x12
Address 1	0x34
Address 2	0x56
Address 3	0x78

# Base Registers

There are

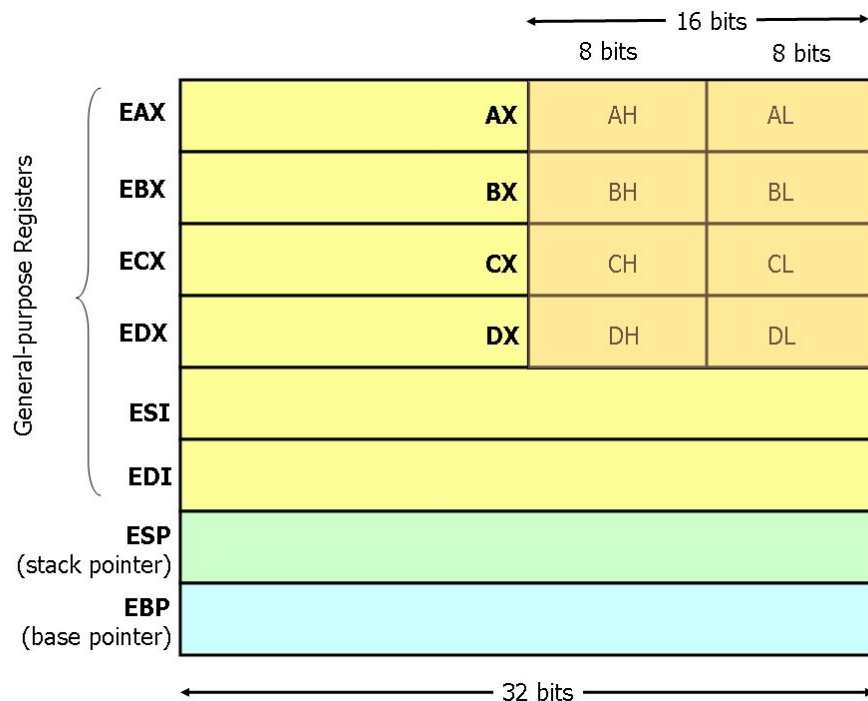
- Eight 32-bit “general-purpose” registers,
- One 32-bit EFLAGS register,
- One 32-bit instruction pointer register (eip), and
- Other special-purpose registers.

# The General-Purpose Registers



- 8 general-purpose registers
- esp is the stack pointer
- ebp is the base pointer
- esi and edi are source and destination index registers for array and string operations

# The General-Purpose Registers



- The registers `eax`, `ebx`, `ecx`, and `edx` may be accessed as 32-bit, 16-bit, or 8-bit registers.
- The other four registers can be accessed as 32-bit or 16-bit.

# EFLAGS Register

The various bits of the 32-bit EFLAGS register are set (1) or reset/clear (0) according to the results of certain operations.

We will be interested in, at most, the bits

CF – carry flag

PF – parity flag

ZF – zero flag

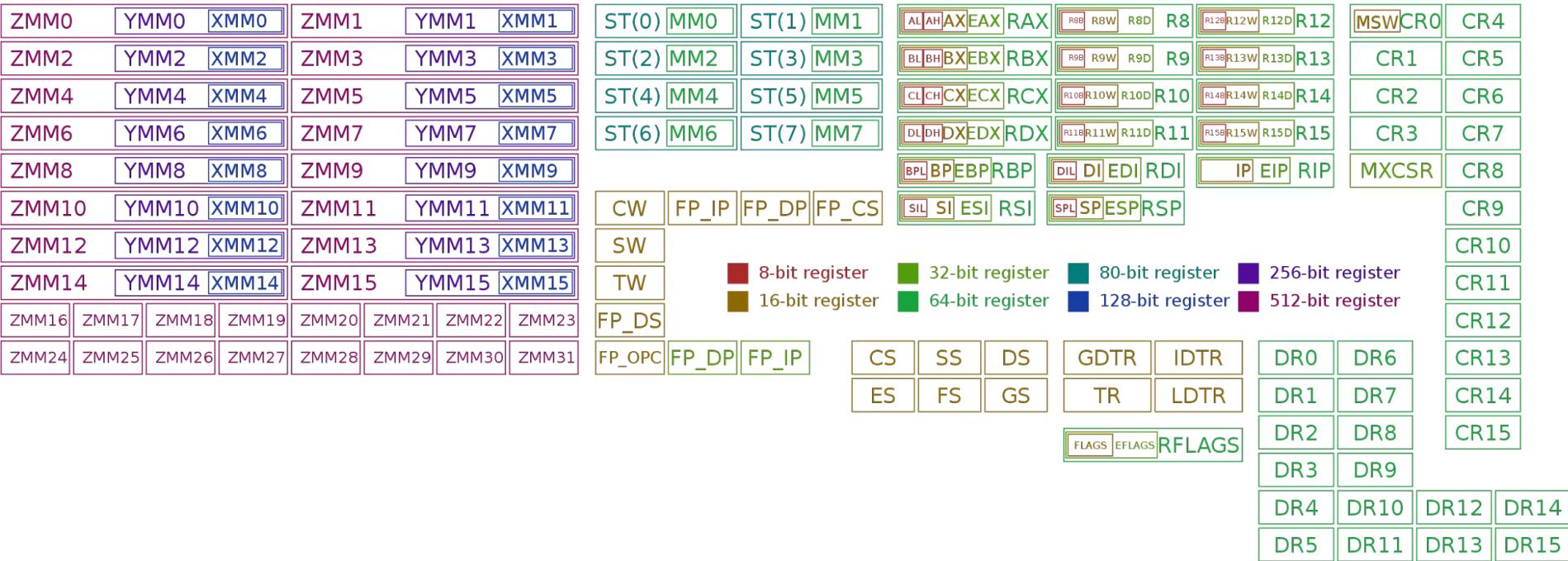
SF – sign flag

# Instruction Pointer (EIP)

Finally, there is the EIP register, which is the instruction pointer (program counter). Register EIP holds the address of the **next** instruction to be executed.



# Registers on x86 and amd64



# Instructions

Each instruction is of the form

label: mnemonic operand1, operand2, operand3

The label is optional.

The number of operands is 0, 1, 2, or 3, depending on the mnemonic .

Each operand is either

- An immediate value,
- A register, or
- A memory address.

# Source and Destination Operands

Each operand is either a source operand or a destination operand.

A source operand, in general, may be

- An immediate value,
- A register, or
- A memory address.

A destination operand, in general, may be

- A register, or
- A memory address.

# Instructions

**hlt** – 0 operands

halts the central processing unit (CPU) until the next external interrupt is fired

**inc** – 1 operand; inc <reg>, inc <mem>

**add** – 2 operands; add <reg>, <reg>

**imul** – 1, 2, or 3 operands; imul <reg32>, <reg32>, <con>

**In Intel syntax the first operand is the destination**

# Intel Syntax Assembly and Disassembly

Machine instructions generally fall into three categories: data movement, arithmetic/logic, and control-flow.

<reg32> Any 32-bit register (eax, ebx, ecx, edx, esi, edi, esp, or ebp)

<reg16> Any 16-bit register (ax, bx, cx, or dx)

<reg8> Any 8-bit register (ah, bh, ch, dh, al, bl, cl, or dl)

<reg> Any register

<mem> A memory address (e.g., **[eax]** or **[eax + ebx\*4]**); **[] square brackets**

<con32> Any 32-bit immediate

<con16> Any 16-bit immediate

<con8> Any 8-bit immediate

<con> Any 8-, 16-, or 32-bit immediate

# Addressing Memory

Move from source (operand 2) to destination (operand 1)

Square bracket [] represents memory location.

**mov [eax], ebx**    Copy 4 bytes from register EBX into memory address specified in EAX.

**mov eax, [esi - 4]**    Move 4 bytes at memory address ESI - 4 into EAX.

**mov [esi + eax \* 1], cl**    Move the contents of CL into the byte at address ESI+EAX\*1.

**mov edx, [esi + ebx\*4]**    Move the 4 bytes of data at address ESI+4\*EBX into EDX.

# Addressing Memory

The size directives BYTE PTR, WORD PTR, and DWORD PTR serve this purpose, indicating sizes of 1, 2, and 4 bytes respectively.

**mov [ebx], 2** isn't this ambiguous? We can have a default.

**mov BYTE PTR [ebx], 2**      Move 2 into the single byte at the address stored in EBX.

**mov WORD PTR [ebx], 2**      Move the 16-bit integer representation of 2 into the 2 bytes starting at the address in EBX.

**mov DWORD PTR [ebx], 2**      Move the 32-bit integer representation of 2 into the 4 bytes starting at the address in EBX.

# Data Movement Instructions

## **mov** — Move

### Syntax

mov <reg>, <reg>

mov <reg>, <mem>

mov <mem>, <reg>

mov <reg>, <con>

mov <mem>, <con>

### Examples

mov eax, ebx — copy the value in EBX into EAX

mov byte ptr [var], 5 — store the value 5 into the byte at location var



# Data Movement Instructions

**push** — Push on stack; decrements ESP by 4, then places the operand at the location ESP points to.

## Syntax

push <reg32>

push <mem>

push <con32>

## Examples

push eax — push eax on the stack

push [var] — push the 4 bytes at address var onto the stack

# Data Movement Instructions

**pop** — Pop from stack

Syntax

pop <reg32>

pop <mem>

Examples

pop edi — pop the top element of the stack into EDI.

pop [ebx] — pop the top element of the stack into memory at the four bytes starting at location EBX.

# LEA Instructions

**lea** — Load effective address; used for quick calculation

Syntax

lea <reg32>, <mem>

Examples

Lea edi, [ebx+4\*esi] — the quantity  $EBX + 4 \cdot ESI$  is placed in EDI.

# Arithmetic and Logic Instructions

**add** eax, 10 — EAX is set to  $EAX + 10$

**addb** byte ptr [eax], 10 — add 10 to the single byte stored at memory address stored in EAX

**sub** al, ah — AL is set to  $AL - AH$

**sub** eax, 216 — subtract 216 from the value stored in EAX

**dec** eax — subtract one from the contents of EAX

**imul** eax, [ebx] — multiply the contents of EAX by the 32-bit contents of the memory at location EBX. Store the result in EAX.

**shr** ebx, cl — Store in EBX the floor of result of dividing the value of EBX by  $2^n$  where  $n$  is the value in CL.

# Control Flow Instructions

## **jmp** — Jump

Transfers program control flow to the instruction at the memory location indicated by the operand.

### Syntax

`jmp <label> # direct jump`

`jmp <reg32> # indirect jump`

### Example

`jmp begin` — Jump to the instruction labeled `begin`.

# Control Flow Instructions

**jcondition** — Conditional jump

Syntax

je <label> (jump when equal)

jne <label> (jump when not equal)

jz <label> (jump when last result was zero)

jg <label> (jump when greater than)

jge <label> (jump when greater than or equal to)

jl <label> (jump when less than)

jle <label> (jump when less than or equal to)

Example

```
cmp ebx, eax
```

```
jle done
```

# Control Flow Instructions

**cmp** — Compare

Syntax

```
cmp <reg>, <reg>
```

```
cmp <mem>, <reg>
```

```
cmp <reg>, <mem>
```

```
cmp <con>, <reg>
```

Example

```
cmp byte ptr [ebx], 10
```

```
jeq loop
```

If the byte stored at the memory location in EBX is equal to the integer constant 10, jump to the location labeled loop.

# Control Flow Instructions

## **call** — Subroutine call

The call instruction first **pushes the current code location onto the hardware supported stack** in memory, and then performs an **unconditional jump to the code** location indicated by the label operand. *Unlike the simple jump instructions, the call instruction saves the location to return to when the subroutine completes.*

### Syntax

call <label>

call <reg32>

Call <mem>



# Control Flow Instructions

**ret** — Subroutine return

The `ret` instruction implements a subroutine return mechanism. This instruction pops a code location off the hardware supported in-memory stack to the program counter.

Syntax

`ret`

# The Run-time Stack

The run-time stack supports procedure calls and the passing of parameters between procedures.

The stack is located in memory.

The stack grows towards **low memory**.

When we push a value, esp is decremented.

When we pop a value, esp is incremented.

# Stack Instructions

**enter** — Create a function frame

Equivalent to:

```
push ebp  
mov ebp, esp  
sub esp, Imm
```

# Stack Instructions

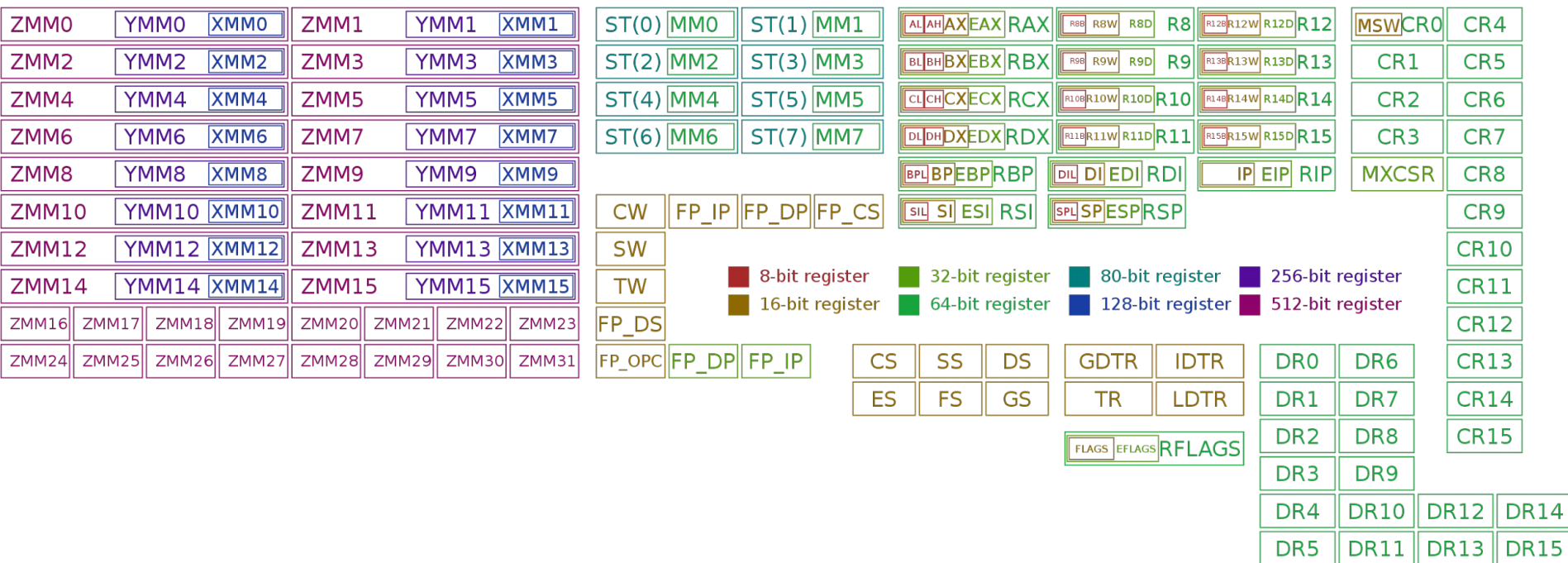
**leave** — Releases the function frame set up by an earlier ENTER instruction.

Equivalent to:

```
mov esp, ebp  
pop ebp
```

# **Background Knowledge: x86-64/amd64 architecture**

# Registers on x86 and x86-64



# x86 vs. x86-64 (code/ladd)

main.c

```
/*  
This program has an integer overflow vulnerability.  
*/
```

```
#include <stdio.h>  
#include <string.h>  
#include <stdlib.h>
```

```
long long ladd(long long *xp, long long y)  
{  
    long long t = *xp + y;  
    return t;  
}
```

```
gcc -Wall -m32 -O2 main.c -o ladd
```

```
gcc -Wall -O2 main.c -o ladd64
```

```
int main(int argc, char *argv[])
```

```
{  
    long long a = 0;  
    long long b = 0;
```

```
    if (argc != 3)  
    {  
        printf("Usage: ladd a b\n");  
        return 0;  
    }
```

```
    printf("The sizeof(long long) is %d\n", sizeof(long long));
```

```
    a = atoll(argv[1]);  
    b = atoll(argv[2]);
```

```
    printf("%lld + %lld = %lld\n", a, b, ladd(&a, b));  
}
```

# x86 vs. x86-64 (code/ladd)

x86

```
000012c0 <ladd>:
 12c0:  f3 0f 1e fb      endbr32
 12c4:  8b 44 24 04      mov  eax,DWORD PTR [esp+0x4]
 12c8:  8b 50 04         mov  edx,DWORD PTR [eax+0x4]
 12cb:  8b 00           mov  eax,DWORD PTR [eax]
 12cd:  03 44 24 08      add  eax,DWORD PTR [esp+0x8]
 12d1:  13 54 24 0c      adc  edx,DWORD PTR [esp+0xc]
 12d5:  c3              ret
```

x86-64

```
00000000000001220 <ladd>:
 1220:  f3 0f 1e fa      endbr64
 1224:  48 8b 07         mov  rax,QWORD PTR [rdi]
 1227:  48 01 f0         add  rax,rsi
 122a:  c3              ret
```

```
objdump -M intel -d ladd_32
objdump -M intel -d ladd_64
```



# **Background Knowledge: ARM Cortex-A/M Architecture**

# Cortex-A 64 bit

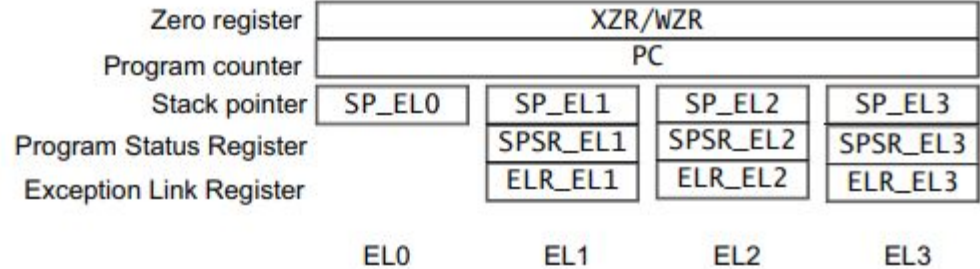
X0/w0
X1/w1
X2/w2
X3/w3
X4/w4
X5/w5
X6/w6
X7/w7
X8/w8
X9/w9
X10/w10
X11/w11
X12/w12
X13/w13
X14/w14
X15/w15
X16/w16
X17/w17
X18/w18
X19/w19
X20/w20
X21/w21
X22/w22
X23/w23
X24/w24
X25/w25
X26/w26
X27/w27
X28/w28
X29/w29
X30/w30

Frame pointer

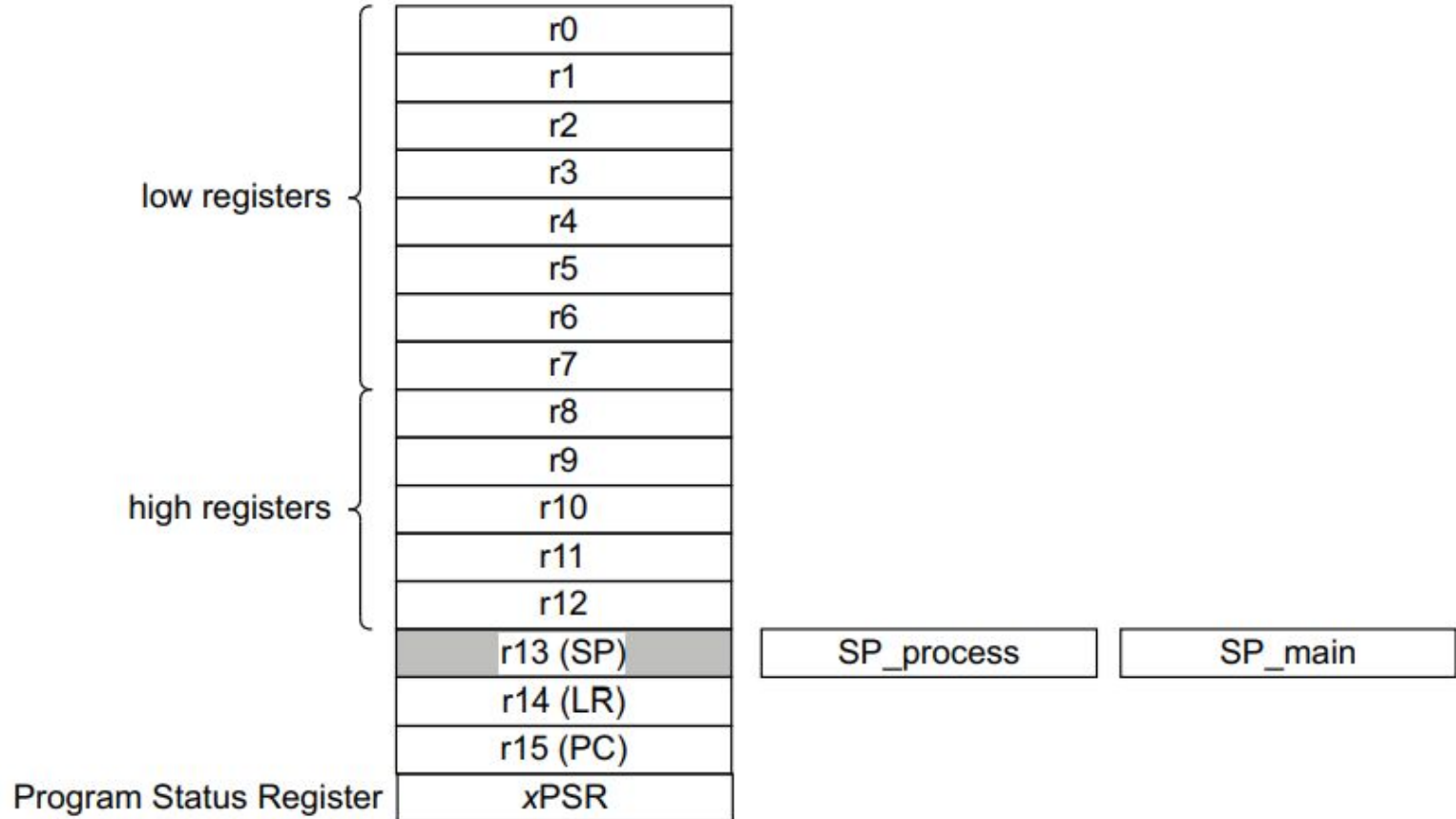
Procedure link register

EL0, EL1,  
EL2, EL3

Special  
registers



# Cortex-M 32 bit



# **Background Knowledge: Linux File Permissions**

# Permission Groups

Each file and directory has three user-based permission groups:

**Owner** – A user is the owner of the file. By default, the person who created a file becomes its owner. The Owner permissions apply only to the owner of the file or directory

**Group** – A group can contain multiple users. All users belonging to a group will have the same access permissions to the file. The Group permissions apply only to the group that has been assigned to the file or directory

**Others** – The others permissions apply to all other users on the system.

# Permission Types

Each file or directory has three basic permission types defined for all the 3 user types:

**Read** – The Read permission refers to a user's capability to read the contents of the file.

**Write** – The Write permissions refer to a user's capability to write or modify a file or directory.

**Execute** – The Execute permission affects a user's capability to execute a file or view the contents of a directory.

**File type:** First field in the output is file type. If there is a - it means it is a plain file. If there is d it means it is a directory, c represents a character device, b represents a block device.

```
ziming@ziming-ThinkPad:~$ ls -l
total 530336
-rw-rw-r-- 1 ziming ziming 742772 Oct 29 2019 14-P2P.pdf
-rw-rw-r-- 1 ziming ziming 32956 Mar 21 23:21 19273679_G.webp
-rw-rw-r-- 1 ziming ziming 94868 Mar 21 23:20 200320_brigham.jpg
-rw-r--r-- 1 ziming ziming 700 Nov 18 2019 2.txt
-rw-r--r-- 1 ziming ziming 145408 Aug 20 2018 acpi_override
drwxr-xr-x 9 ziming ziming 4096 Mar 18 15:48 App
drwxrwxr-x 4 ziming ziming 4096 Apr 11 2019 Arduino
-rw-r--r-- 1 ziming ziming 163225 Jul 14 2019 autoproxy.pac
drwxr-xr-x 3 ziming ziming 4096 May 21 10:22 Desktop
drwxr-xr-x 3 ziming ziming 4096 Oct 11 2018 devel
drwxr-xr-x 3 ziming ziming 4096 Oct 26 2018 develqemu
drwxr-xr-x 4 ziming ziming 4096 May 19 14:31 Documents
drwxr-xr-x 4 ziming ziming 69632 May 24 10:11 Downloads
drwx----- 58 ziming ziming 4096 May 24 09:51 Dropbox
-rw-r--r-- 1 ziming ziming 144272 Aug 20 2018 dsdt.aml
-rw-r--r-- 1 ziming ziming 1075439 Aug 20 2018 dsdt.dsl
-rw-r--r-- 1 ziming ziming 1075439 Aug 20 2018 dsdt.dsl.ziming.manual
-rw-r--r-- 1 ziming ziming 1352883 Aug 20 2018 dsdt.hex
-rw-r--r-- 1 ziming ziming 0 Nov 6 2019 enclave.token
-rw-rw-r-- 1 ziming ziming 57747 Mar 21 23:20 ETj0lBjXkAMXVJs-630x390.jpg
-rw-r--r-- 1 ziming ziming 8980 Aug 16 2018 examples.desktop
```

## Permissions for owner, group, and others

```
ziming@ziming-ThinkPad:~$ ls -l
total 530336
-rw-rw-r-- 1 ziming ziming 742772 Oct 29 2019 14-P2P.pdf
-rw-rw-r-- 1 ziming ziming 32956 Mar 21 23:21 19273679_G.webp
-rw-rw-r-- 1 ziming ziming 94868 Mar 21 23:20 200320_brigham.jpg
-rw-r--r-- 1 ziming ziming 700 Nov 18 2019 2.txt
-rw-r--r-- 1 ziming ziming 145408 Aug 20 2018 acpi_override
drwxr-xr-x 9 ziming ziming 4096 Mar 18 15:48 App
drwxrwxr-x 4 ziming ziming 4096 Apr 11 2019 Arduino
-rw-r--r-- 1 ziming ziming 163225 Jul 14 2019 autoproxy.pac
drwxr-xr-x 3 ziming ziming 4096 May 21 10:22 Desktop
drwxr-xr-x 3 ziming ziming 4096 Oct 11 2018 devel
drwxr-xr-x 3 ziming ziming 4096 Oct 26 2018 develqemu
drwxr-xr-x 4 ziming ziming 4096 May 19 14:31 Documents
drwxr-xr-x 4 ziming ziming 69632 May 24 10:11 Downloads
drwx----- 58 ziming ziming 4096 May 24 09:51 Dropbox
-rw-r--r-- 1 ziming ziming 144272 Aug 20 2018 dsdt.aml
-rw-r--r-- 1 ziming ziming 1075439 Aug 20 2018 dsdt.dsl
-rw-r--r-- 1 ziming ziming 1075439 Aug 20 2018 dsdt.dsl.ziming.manual
-rw-r--r-- 1 ziming ziming 1352883 Aug 20 2018 dsdt.hex
-rw-r--r-- 1 ziming ziming 0 Nov 6 2019 enclave.token
-rw-rw-r-- 1 ziming ziming 57747 Mar 21 23:20 ETj0lBjXkAMXVJs-630x390.jpg
-rw-r--r-- 1 ziming ziming 8980 Aug 16 2018 examples.desktop
```



Link count

```
ziming@ziming-ThinkPad:~$ ls -l
total 530336
-rw-rw-r-- 1 ziming ziming 742772 Oct 29 2019 14-P2P.pdf
-rw-rw-r-- 1 ziming ziming 32956 Mar 21 23:21 19273679_G.webp
-rw-rw-r-- 1 ziming ziming 94868 Mar 21 23:20 200320_brigham.jpg
-rw-r--r-- 1 ziming ziming 700 Nov 18 2019 2.txt
-rw-r--r-- 1 ziming ziming 145408 Aug 20 2018 acpi_override
drwxr-xr-x 9 ziming ziming 4096 Mar 18 15:48 App
drwxrwxr-x 4 ziming ziming 4096 Apr 11 2019 Arduino
-rw-r--r-- 1 ziming ziming 163225 Jul 14 2019 autoproxy.pac
drwxr-xr-x 3 ziming ziming 4096 May 21 10:22 Desktop
drwxr-xr-x 3 ziming ziming 4096 Oct 11 2018 devel
drwxr-xr-x 3 ziming ziming 4096 Oct 26 2018 develqemu
drwxr-xr-x 4 ziming ziming 4096 May 19 14:31 Documents
drwxr-xr-x 4 ziming ziming 69632 May 24 10:11 Downloads
drwx----- 58 ziming ziming 4096 May 24 09:51 Dropbox
-rw-r--r-- 1 ziming ziming 144272 Aug 20 2018 dsdt.aml
-rw-r--r-- 1 ziming ziming 1075439 Aug 20 2018 dsdt.dsl
-rw-r--r-- 1 ziming ziming 1075439 Aug 20 2018 dsdt.dsl.ziming.manual
-rw-r--r-- 1 ziming ziming 1352883 Aug 20 2018 dsdt.hex
-rw-r--r-- 1 ziming ziming 0 Nov 6 2019 enclave.token
-rw-rw-r-- 1 ziming ziming 57747 Mar 21 23:20 ETj0lBjXkAMXVJs-630x390.jpg
-rw-r--r-- 1 ziming ziming 8980 Aug 16 2018 examples.desktop
```

**Owner:** This field provide info about the creator of the file.

```
ziming@ziming-ThinkPad:~$ ls -l
total 530336
-rw-rw-r-- 1 ziming ziming 742772 Oct 29 2019 14-P2P.pdf
-rw-rw-r-- 1 ziming ziming 32956 Mar 21 23:21 19273679_G.webp
-rw-rw-r-- 1 ziming ziming 94868 Mar 21 23:20 200320_brigham.jpg
-rw-r--r-- 1 ziming ziming 700 Nov 18 2019 2.txt
-rw-r--r-- 1 ziming ziming 145408 Aug 20 2018 acpi_override
drwxr-xr-x 9 ziming ziming 4096 Mar 18 15:48 App
drwxrwxr-x 4 ziming ziming 4096 Apr 11 2019 Arduino
-rw-r--r-- 1 ziming ziming 163225 Jul 14 2019 autoproxy.pac
drwxr-xr-x 3 ziming ziming 4096 May 21 10:22 Desktop
drwxr-xr-x 3 ziming ziming 4096 Oct 11 2018 devel
drwxr-xr-x 3 ziming ziming 4096 Oct 26 2018 develqemu
drwxr-xr-x 4 ziming ziming 4096 May 19 14:31 Documents
drwxr-xr-x 4 ziming ziming 69632 May 24 10:11 Downloads
drwx----- 58 ziming ziming 4096 May 24 09:51 Dropbox
-rw-r--r-- 1 ziming ziming 144272 Aug 20 2018 dsdt.aml
-rw-r--r-- 1 ziming ziming 1075439 Aug 20 2018 dsdt.dsl
-rw-r--r-- 1 ziming ziming 1075439 Aug 20 2018 dsdt.dsl.ziming.manual
-rw-r--r-- 1 ziming ziming 1352883 Aug 20 2018 dsdt.hex
-rw-r--r-- 1 ziming ziming 0 Nov 6 2019 enclave.token
-rw-rw-r-- 1 ziming ziming 57747 Mar 21 23:20 ETj0lBjXkAMXVJs-630x390.jpg
-rw-r--r-- 1 ziming ziming 8980 Aug 16 2018 examples.desktop
```

Group

```
ziming@ziming-ThinkPad:~$ ls -l
```

```
total 530336
```

-rw-rw-r--	1	ziming	ziming	742772	Oct	29	2019	14-P2P.pdf
-rw-rw-r--	1	ziming	ziming	32956	Mar	21	23:21	19273679_G.webp
-rw-rw-r--	1	ziming	ziming	94868	Mar	21	23:20	200320_brigham.jpg
-rw-r--r--	1	ziming	ziming	700	Nov	18	2019	2.txt
-rw-r--r--	1	ziming	ziming	145408	Aug	20	2018	acpi_override
drwxr-xr-x	9	ziming	ziming	4096	Mar	18	15:48	App
drwxrwxr-x	4	ziming	ziming	4096	Apr	11	2019	Arduino
-rw-r--r--	1	ziming	ziming	163225	Jul	14	2019	autoproxy.pac
drwxr-xr-x	3	ziming	ziming	4096	May	21	10:22	Desktop
drwxr-xr-x	3	ziming	ziming	4096	Oct	11	2018	devel
drwxr-xr-x	3	ziming	ziming	4096	Oct	26	2018	develqemu
drwxr-xr-x	4	ziming	ziming	4096	May	19	14:31	Documents
drwxr-xr-x	4	ziming	ziming	69632	May	24	10:11	Downloads
drwx-----	58	ziming	ziming	4096	May	24	09:51	Dropbox
-rw-r--r--	1	ziming	ziming	144272	Aug	20	2018	dsdt.aml
-rw-r--r--	1	ziming	ziming	1075439	Aug	20	2018	dsdt.dsl
-rw-r--r--	1	ziming	ziming	1075439	Aug	20	2018	dsdt.dsl.ziming.manual
-rw-r--r--	1	ziming	ziming	1352883	Aug	20	2018	dsdt.hex
-rw-r--r--	1	ziming	ziming	0	Nov	6	2019	enclave.token
-rw-rw-r--	1	ziming	ziming	57747	Mar	21	23:20	ETj0lBjXkAMXVJs-630x390.jpg
-rw-r--r--	1	ziming	ziming	8980	Aug	16	2018	examples.desktop



File size

```
ziming@ziming-ThinkPad:~$ ls -l
total 530336
-rw-rw-r-- 1 ziming ziming 742772 Oct 29 2019 14-P2P.pdf
-rw-rw-r-- 1 ziming ziming 32956 Mar 21 23:21 19273679_G.webp
-rw-rw-r-- 1 ziming ziming 94868 Mar 21 23:20 200320_brigham.jpg
-rw-r--r-- 1 ziming ziming 700 Nov 18 2019 2.txt
-rw-r--r-- 1 ziming ziming 145408 Aug 20 2018 acpi_override
drwxr-xr-x 9 ziming ziming 4096 Mar 18 15:48 App
drwxrwxr-x 4 ziming ziming 4096 Apr 11 2019 Arduino
-rw-r--r-- 1 ziming ziming 163225 Jul 14 2019 autoproxy.pac
drwxr-xr-x 3 ziming ziming 4096 May 21 10:22 Desktop
drwxr-xr-x 3 ziming ziming 4096 Oct 11 2018 devel
drwxr-xr-x 3 ziming ziming 4096 Oct 26 2018 develqemu
drwxr-xr-x 4 ziming ziming 4096 May 19 14:31 Documents
drwxr-xr-x 4 ziming ziming 69632 May 24 10:11 Downloads
drwx----- 58 ziming ziming 4096 May 24 09:51 Dropbox
-rw-r--r-- 1 ziming ziming 144272 Aug 20 2018 dsdt.aml
-rw-r--r-- 1 ziming ziming 1075439 Aug 20 2018 dsdt.dsl
-rw-r--r-- 1 ziming ziming 1075439 Aug 20 2018 dsdt.dsl.ziming.manual
-rw-r--r-- 1 ziming ziming 1352883 Aug 20 2018 dsdt.hex
-rw-r--r-- 1 ziming ziming 0 Nov 6 2019 enclave.token
-rw-rw-r-- 1 ziming ziming 57747 Mar 21 23:20 ETj0lBjXkAMXVJs-630x390.jpg
-rw-r--r-- 1 ziming ziming 8980 Aug 16 2018 examples.desktop
```

Last modify time

```
ziming@ziming-ThinkPad:~$ ls -l
```

```
total 530336
```

-rw-rw-r--	1	ziming	ziming	742772	Oct	29	2019	14-P2P.pdf
-rw-rw-r--	1	ziming	ziming	32956	Mar	21	23:21	19273679_G.webp
-rw-rw-r--	1	ziming	ziming	94868	Mar	21	23:20	200320_brigham.jpg
-rw-r--r--	1	ziming	ziming	700	Nov	18	2019	2.txt
-rw-r--r--	1	ziming	ziming	145408	Aug	20	2018	acpi_override
drwxr-xr-x	9	ziming	ziming	4096	Mar	18	15:48	App
drwxrwxr-x	4	ziming	ziming	4096	Apr	11	2019	Arduino
-rw-r--r--	1	ziming	ziming	163225	Jul	14	2019	autoproxy.pac
drwxr-xr-x	3	ziming	ziming	4096	May	21	10:22	Desktop
drwxr-xr-x	3	ziming	ziming	4096	Oct	11	2018	devel
drwxr-xr-x	3	ziming	ziming	4096	Oct	26	2018	develqemu
drwxr-xr-x	4	ziming	ziming	4096	May	19	14:31	Documents
drwxr-xr-x	4	ziming	ziming	69632	May	24	10:11	Downloads
drwx-----	58	ziming	ziming	4096	May	24	09:51	Dropbox
-rw-r--r--	1	ziming	ziming	144272	Aug	20	2018	dsdt.aml
-rw-r--r--	1	ziming	ziming	1075439	Aug	20	2018	dsdt.dsl
-rw-r--r--	1	ziming	ziming	1075439	Aug	20	2018	dsdt.dsl.ziming.manual
-rw-r--r--	1	ziming	ziming	1352883	Aug	20	2018	dsdt.hex
-rw-r--r--	1	ziming	ziming	0	Nov	6	2019	enclave.token
-rw-rw-r--	1	ziming	ziming	57747	Mar	21	23:20	ETj0lBjXkAMXVJs-630x390.jpg
-rw-r--r--	1	ziming	ziming	8980	Aug	16	2018	examples.desktop

filename

```
ziming@ziming-ThinkPad:~$ ls -l
```

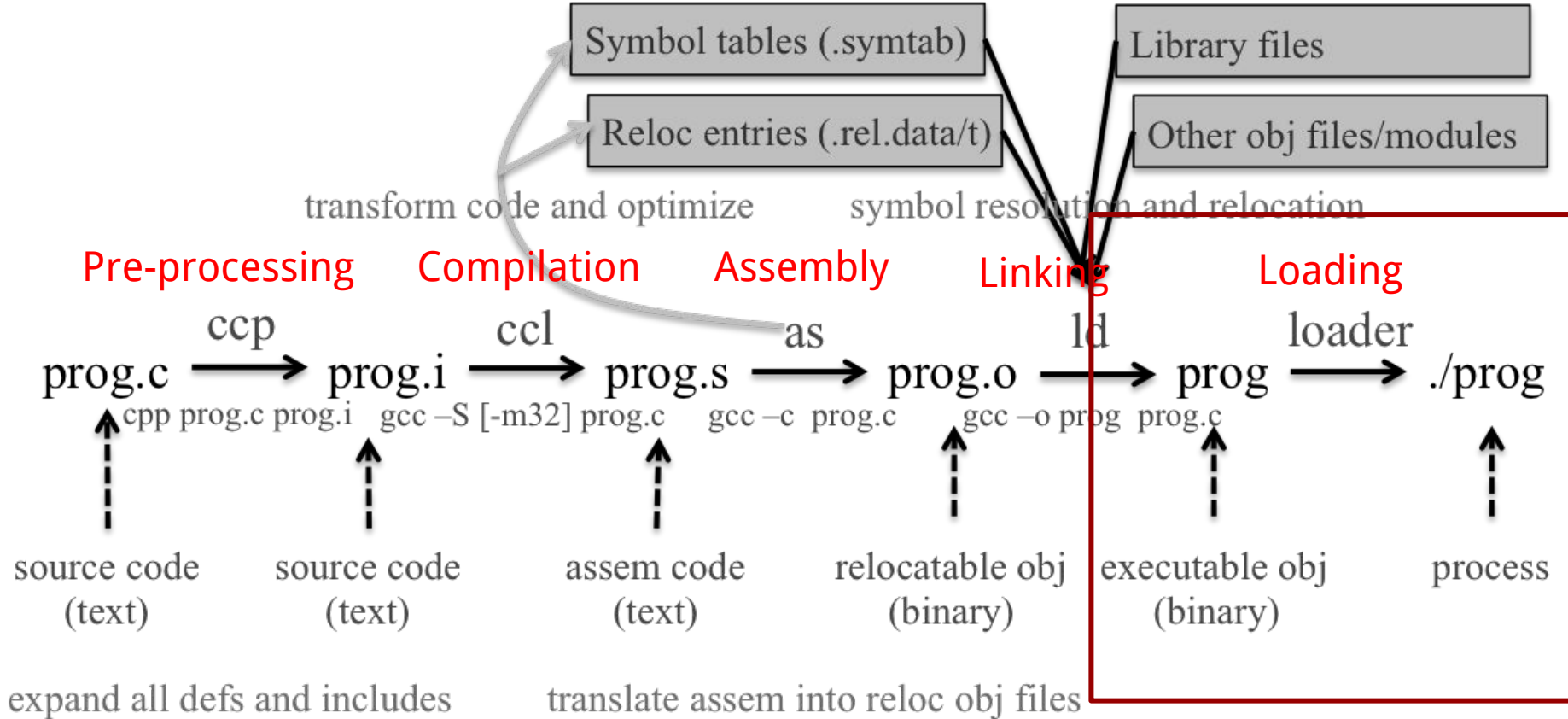
```
total 530336
```

-rw-rw-r--	1	ziming	ziming	742772	Oct	29	2019	14-P2P.pdf
-rw-rw-r--	1	ziming	ziming	32956	Mar	21	23:21	19273679_G.webp
-rw-rw-r--	1	ziming	ziming	94868	Mar	21	23:20	200320_brigham.jpg
-rw-r--r--	1	ziming	ziming	700	Nov	18	2019	2.txt
-rw-r--r--	1	ziming	ziming	145408	Aug	20	2018	acpi_override
drwxr-xr-x	9	ziming	ziming	4096	Mar	18	15:48	App
drwxrwxr-x	4	ziming	ziming	4096	Apr	11	2019	Arduino
-rw-r--r--	1	ziming	ziming	163225	Jul	14	2019	autoproxy.pac
drwxr-xr-x	3	ziming	ziming	4096	May	21	10:22	Desktop
drwxr-xr-x	3	ziming	ziming	4096	Oct	11	2018	devel
drwxr-xr-x	3	ziming	ziming	4096	Oct	26	2018	develqemu
drwxr-xr-x	4	ziming	ziming	4096	May	19	14:31	Documents
drwxr-xr-x	4	ziming	ziming	69632	May	24	10:11	Downloads
drwx-----	58	ziming	ziming	4096	May	24	09:51	Dropbox
-rw-r--r--	1	ziming	ziming	144272	Aug	20	2018	dsdt.aml
-rw-r--r--	1	ziming	ziming	1075439	Aug	20	2018	dsdt.dsl
-rw-r--r--	1	ziming	ziming	1075439	Aug	20	2018	dsdt.dsl.ziming.manual
-rw-r--r--	1	ziming	ziming	1352883	Aug	20	2018	dsdt.hex
-rw-r--r--	1	ziming	ziming	0	Nov	6	2019	enclave.token
-rw-rw-r--	1	ziming	ziming	57747	Mar	21	23:20	ETj0lBjXkAMXVJs-630x390.jpg
-rw-r--r--	1	ziming	ziming	8980	Aug	16	2018	examples.desktop

# **Background Knowledge: Set-UID Programs**



# From a C program to a process





# Real UID, Effective UID, and Saved UID

Each Linux/Unix **process** has 3 UIDs associated with it.

**Real UID (RUID):** This is the UID of the user/process that created THIS process. It can be changed only if the running process has EUID=0.

**Effective UID (EUID):** This UID is used to evaluate privileges of the process to perform a particular action. EUID can be changed either to RUID, or SUID if EUID!=0. If EUID=0, it can be changed to anything.

**Saved UID (SUID):** If the binary image file, that was launched has a Set-UID bit on, SUID will be the UID of the owner of the file. Otherwise, SUID will be the RUID.

# Set-UID Program

The kernel makes the decision whether a process has the privilege by looking on the **EUID** of the process.

For non Set-UID programs, the effective uid and the real uid are the same. For Set-UID programs, **the effective uid is the owner of the program**, while the real uid is the user of the program.

What will happen is when a setuid binary executes, the process changes its Effective User ID (EUID) from the default RUID to the owner of this special binary executable file which in this case is - root.

```
ziming@ziming-ThinkPad:~$ ls -al /bin/
```

```
total 12676
drwxr-xr-x  2 root root    4096 May 26 00:14 .
drwxr-xr-x 26 root root    4096 May 18 09:57 ..
-rwxr-xr-x  1 root root 1113504 Jun  6 2019 bash
-rwxr-xr-x  1 root root  748968 Aug 29 2018 brltty
-rwxr-xr-x  3 root root   34888 Jul  4 2019 bunzip2
-rwxr-xr-x  1 root root 2062296 Mar  6 2019 busybox
-rwxr-xr-x  3 root root   34888 Jul  4 2019 bzipcat
lrwxrwxrwx  1 root root      6 Jul  4 2019 bzcmp -> bzdiff
-rwxr-xr-x  1 root root   2140 Jul  4 2019 bzdiff
lrwxrwxrwx  1 root root      6 Jul  4 2019 bzegrep ->
-rwxr-xr-x  1 root root   4877 Jul  4 2019 bzexe lrwxrwxrwx  1 root root      4 Aug 16 2018 sh -> dash
lrwxrwxrwx  1 root root      6 Jul  4 2019 bzfgrep -> lrwxrwxrwx  1 root root      4 Aug 16 2018 sh.distrib -> dash
-rwxr-xr-x  1 root root   3642 Jul  4 2019 bzgrep -> -rwxr-xr-x  1 root root   35000 Jan 18 2018 sleep
-rwxr-xr-x  3 root root   34888 Jul  4 2019 bzip2 -rwxr-xr-x  1 root root 139904 May 11 10:40 ss
-rwxr-xr-x  1 root root 14328 Jul  4 2019 bzip2recover lrwxrwxrwx  1 root root      7 Mar  6 2019 static-sh -> busybox
lrwxrwxrwx  1 root root      6 Jul  4 2019 bzless -> -rwxr-xr-x  1 root root   75992 Jan 18 2018 stty
-rwxr-xr-x  1 root root   1297 Jul  4 2019 bzmores -rwsr-xr-x  1 root root   44664 Mar 22 2019 su
-rwxr-xr-x  1 root root   35064 Jan 18 2018 cat -rwxr-xr-x  1 root root   35000 Jan 18 2018 sync
-rwxr-xr-x  1 root root 14328 Apr 21 2017 chacl -rwxr-xr-x  1 root root 182352 May  3 07:30 systemctl
lrwxrwxrwx  1 root root   63672 Jan 18 2018 chgrp lrwxrwxrwx  1 root root    20 May  3 07:30 systemd -> /lib/systemd/systemd
-rwxr-xr-x  1 root root   59608 Jan 18 2018 chmod -rwxr-xr-x  1 root root 10320 May  3 07:30 systemd-ask-password
-rwxr-xr-x  1 root root   67768 Jan 18 2018 chown -rwxr-xr-x  1 root root 14400 May  3 07:30 systemd-escape
-rwxr-xr-x  1 root root 10312 Jan 22 2018 chvt -rwxr-xr-x  1 root root 84328 May  3 07:30 systemd-hwdb
-rwxr-xr-x  1 root root 141528 Jan 18 2018 cp -rwxr-xr-x  1 root root 14416 May  3 07:30 systemd-inhibit
-rwxr-xr-x  1 root root 157224 Nov  5 2019 cpio -rwxr-xr-x  1 root root 18496 May  3 07:30 systemd-machine-id-setup
-rwxr-xr-x  1 root root 121432 Jan 25 2018 dash -rwxr-xr-x  1 root root 14408 May  3 07:30 systemd-notify
-rwxr-xr-x  1 root root 100568 Jan 18 2018 date -rwxr-xr-x  1 root root 43080 May  3 07:30 systemd-sysusers
-rwxr-xr-x  1 root root  76000 Jan 18 2018 dd -rwxr-xr-x  1 root root 71752 May  3 07:30 systemd-tmpfiles
-rwxr-xr-x  1 root root 84776 Jan 18 2018 df -rwxr-xr-x  1 root root 26696 May  3 07:30 systemd-tty-ask-password-agent
-rwxr-xr-x  1 root root 133792 Jan 18 2018 dir -rwxr-xr-x  1 root root 423312 Jan 21 2019 tar
-rwxr-xr-x  1 root root  72000 Mar  5 12:23 dmesg -rwxr-xr-x  1 root root 10104 Dec 30 2017 tempfile
-rwxr-xr-x  1 root root -rwxr-xr-x  1 root root 88280 Jan 18 2018 touch
-rwxr-xr-x  1 root root 30904 Jan 18 2018 true
-rwxr-xr-x  1 root root 584072 May  3 07:30 udevadm
-rwxr-xr-x  1 root root 14328 Aug 11 2016 ulockmgr_server
-rwsr-xr-x  1 root root 26696 Mar  5 12:23 umount
-rwxr-xr-x  1 root root 35032 Jan 18 2018 uname
```



-rwxr-xr-x	1	root	root	39103	Apr	23	2019	setupcon
lrwxrwxrwx	1	root	root	4	Aug	16	2018	sh -> dash
lrwxrwxrwx	1	root	root	4	Aug	16	2018	sh.distrib -> dash
-rwxr-xr-x	1	root	root	35000	Jan	18	2018	sleep
-rwxr-xr-x	1	root	root	139904	May	11	10:40	ss
lrwxrwxrwx	1	root	root	7	Mar	6	2019	static-sh -> busybox
-rwxr-xr-x	1	root	root	75992	Jan	18	2018	stty
-rwsr-xr-x	1	root	root	44664	Mar	22	2019	su
-rwxr-xr-x	1	root	root	35000	Jan	18	2018	sync
-rwxr-xr-x	1	root	root	182352	May	3	07:30	systemctl
lrwxrwxrwx	1	root	root	20	May	3	07:30	systemd -> /lib/systemd/systemd
-rwxr-xr-x	1	root	root	10320	May	3	07:30	systemd-ask-password
-rwxr-xr-x	1	root	root	14400	May	3	07:30	systemd-escape
-rwxr-xr-x	1	root	root	84328	May	3	07:30	systemd-hwdb
-rwxr-xr-x	1	root	root	14416	May	3	07:30	systemd-inhibit
-rwxr-xr-x	1	root	root	18496	May	3	07:30	systemd-machine-id-setup
-rwxr-xr-x	1	root	root	14408	May	3	07:30	systemd-notify
-rwxr-xr-x	1	root	root	43080	May	3	07:30	systemd-sysusers
-rwxr-xr-x	1	root	root	71752	May	3	07:30	systemd-tmpfiles
-rwxr-xr-x	1	root	root	26696	May	3	07:30	systemd-tty-ask-password-agent
-rwxr-xr-x	1	root	root	423312	Jan	21	2019	tar
-rwxr-xr-x	1	root	root	10104	Dec	30	2017	tempfile
-rwxr-xr-x	1	root	root	88280	Jan	18	2018	touch
-rwxr-xr-x	1	root	root	30904	Jan	18	2018	true
-rwxr-xr-x	1	root	root	584072	May	3	07:30	udevadm
-rwxr-xr-x	1	root	root	14328	Aug	11	2016	unlockmgr_server
-rwsr-xr-x	1	root	root	26696	Mar	5	12:23	umount
-rwxr-xr-x	1	root	root	35032	Jan	18	2018	uname

# Example: rdsecret

main.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <pwd.h>
```

```
int main(int argc, char *argv[])
```

```
{
    FILE *fp = NULL;
    char buffer[100] = {0};

    // get ruid and euid
    uid_t uid = getuid();
    struct passwd *pw = getpwuid(uid);
    if (pw)
    {
        printf("UID: %d, USER: %s.\n", uid, pw->pw_name);
    }
}
```

```
uid_t euid = geteuid();
pw = getpwuid(euid);
```

```
if (pw)
{
    printf("EUID: %d, EUSER: %s.\n", euid, pw->pw_name);
}
```

```
print_flag();
```

```
return(0);
```

```
}
```

```
void print_flag()
```

```
{
    FILE *fp;
    char buff[MAX_FLAG_SIZE];
    fp = fopen("flag", "r");
    fread(buff, MAX_FLAG_SIZE, 1, fp);
    printf("flag is : %s\n", buff);
    fclose(fp);
}
```

# **Background Knowledge: ELF Binary Files**

# ELF Files

The **Executable and Linkable Format (ELF)** is a common standard file format for *executable files, object code, shared libraries, and core dumps*. Filename extension *none, .axf, .bin, .elf, .o, .prx, .puff, .ko, .mod* and *.so*

Contains the program and its data. Describes how the program should be loaded (program/segment headers). Contains metadata describing program components (section headers).

# Command *file*

```
ziming@ziming-XPS-13-9300:~$ file /bin/ls
/bin/ls: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically lin
ked, interpreter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=2f15ad836be3339dec0e
2e6a3c637e08e48aacbd, for GNU/Linux 3.2.0, stripped
ziming@ziming-XPS-13-9300:~$
```

file /bin/ls



```

zmling@zmling-XPS-13-9300:~$ readelf -a /bin/ls
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 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:      0x67d0
  Start of program headers: 64 (bytes into file)
  Start of section headers: 140224 (bytes into file)
  Flags:                    0x0
  Size of this header:      64 (bytes)
  Size of program headers:  56 (bytes)
  Number of program headers: 13
  Size of section headers:  64 (bytes)
  Number of section headers: 30
  Section header string table index: 29

```

#### Section Headers:

[Nr]	Name	Type	Address	Offset
	Size	EntSize	Flags Link Info Align	
[ 0]	0000000000000000	NULL	0000000000000000	00000000
[ 1]	.interp 000000000000001c	PROGBITS	000000000000318 A 0 0 1	00000318
[ 2]	.note.gnu.propert 0000000000000020	NOTE	000000000000338 A 0 0 8	00000338
[ 3]	.note.gnu.build-i 0000000000000024	NOTE	000000000000358 A 0 0 4	00000358
[ 4]	.note.ABI-tag 0000000000000020	NOTE	00000000000037c A 0 0 4	0000037c
[ 5]	.gnu.hash 00000000000000e4	GNU_HASH	0000000000003a0 A 6 0 8	000003a0
[ 6]	.dynsym 00000000000000d8	DYNSYM	000000000000488 A 7 1 8	00000488
[ 7]	.dynstr 0000000000000064c	STRTAB	0000000000001190 A 0 0 1	00001190
[ 8]	.gnu.version 0000000000000116	VERSYM	00000000000017dc A 6 0 2	000017dc
[ 9]	.gnu.version_r 0000000000000070	VERNEED	00000000000018f8 A 7 1 8	000018f8
[10]	.rela.dyn 0000000000001350	RELA	0000000000001968 A 6 0 8	00001968
[11]	.rela.plt 00000000000009f0	RELA	0000000000002cb8 AI 6 25 8	00002cb8
[12]	.init 000000000000001b	PROGBITS	0000000000004000 AX 0 0 4	00004000
[13]	.plt 00000000000006b0	PROGBITS	0000000000004020 AX 0 0 16	00004020

**INTERP:** defines the library that should be used to load this ELF into memory.

**LOAD:** defines a part of the file that should be loaded into memory.

#### Sections:

**.text:** the executable code of your program.

**.plt** and **.got:** used to resolve and dispatch library calls.

**.data:** used for pre-initialized global writable data (such as global arrays with initial values)

**.rodata:** used for global read-only data (such as string constants)

**.bss:** used for uninitialized global writable data (such as global arrays without initial values)

# Tools for ELF

**gcc** to make your ELF.

**readelf** to parse the ELF header.

**objdump** to parse the ELF header and disassemble the source code.

**nm** to view your ELF's symbols.

**patchelf** to change some ELF properties.

**objcopy** to swap out ELF sections.

**strip** to remove otherwise-helpful information (such as symbols).

**kaitai struct** (<https://ide.kaitai.io/>) to look through your ELF interactively.

# **Background Knowledge: Memory Map of a Linux Process**

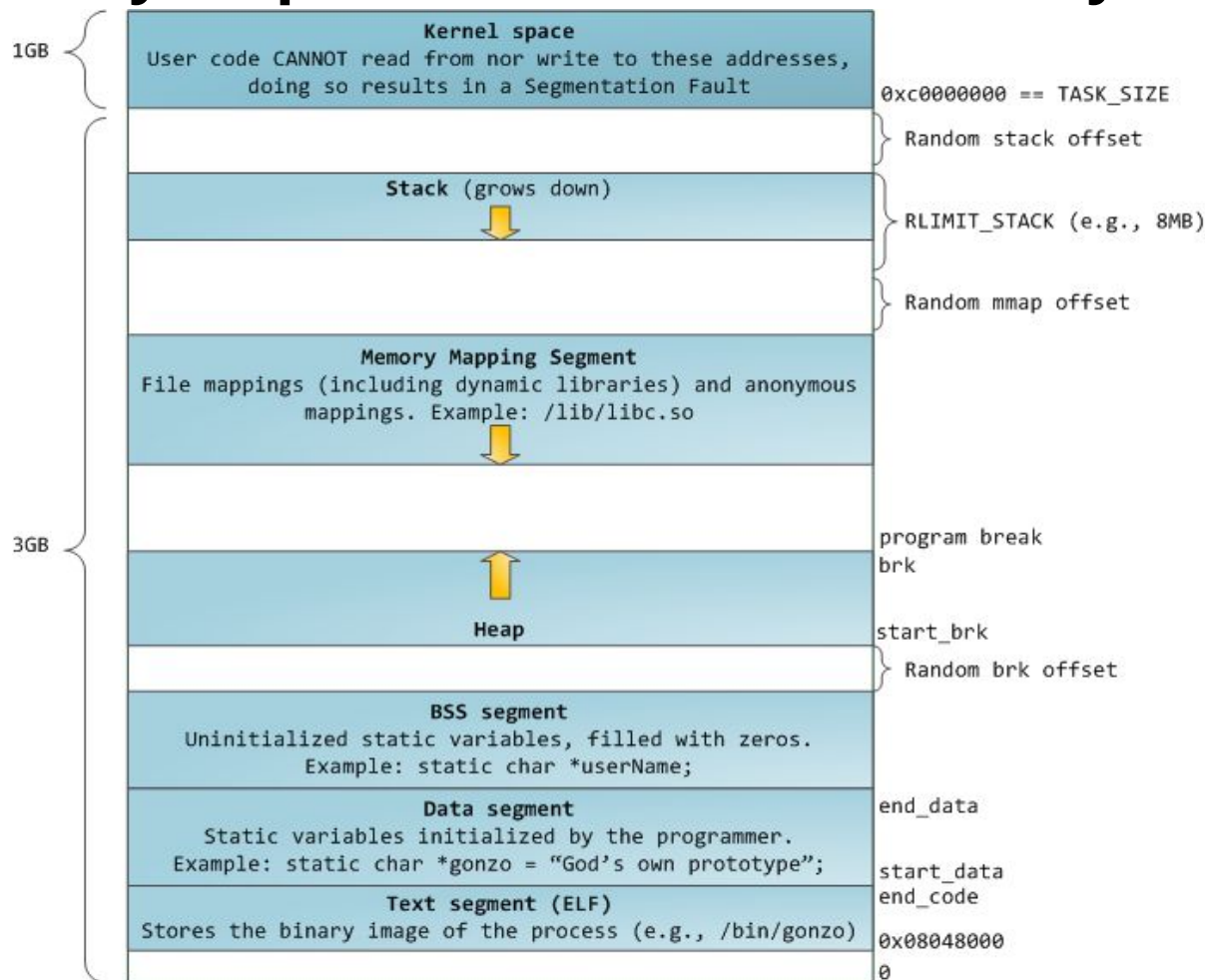
# Memory Map of Linux Process (32 bit)

Each process in a multi-tasking OS runs in its own memory sandbox.

This sandbox is the **virtual address space**, which in 32-bit mode is **always a 4GB block of memory addresses**.

These virtual addresses are mapped to physical memory by **page tables**, which are maintained by the operating system kernel and consulted by the processor.

# Memory Map of Linux Process (32 bit system)



<https://manybutfinite.com/pos-anatomy-of-a-program-in-memory/>

# NULL Pointer in C/C++

```
int * pInt = NULL;
```

In possible definitions of NULL in C/C++:

```
#define NULL ((char *)0)
```

```
#define NULL 0
```

```
//since C++11
```

```
#define NULL nullptr
```

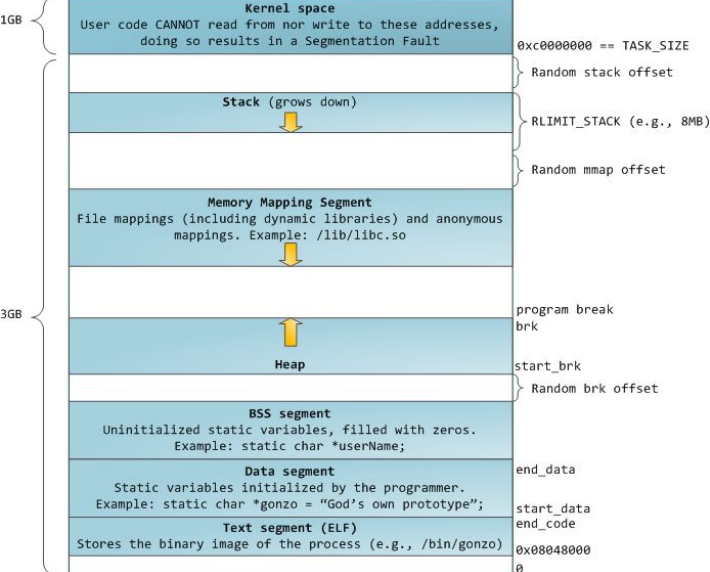
# /proc/pid\_of\_process/maps

## Example processmap.c

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    getchar();
    return 0;
}
```

```
cat /proc/pid/maps
pmap -X pid
pmap -X `pidof pm`
```



```

ziming@ziming-ThinkPad:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/processmap$ pmap -X 21732
21732:  ./pm
Address Perm  Offset Device      Inode  Size  Rss  Pss  Referenced Anonymous LazyFree ShmemPmdMapped Shared_Hugetlb Private_Hugetlb Swap SwapPss Locked Mapping
56569000 r-xp 00000000 103:02 28575310 4 4 4 4 0 0 0 0 0 0 0 0 pm
5656a000 r--p 00000000 103:02 28575310 4 4 4 4 4 0 0 0 0 0 0 0 pm
5656b000 rw-p 00001000 103:02 28575310 4 4 4 4 4 0 0 0 0 0 0 0 pm
57cf2000 rw-p 00000000 00:00 0 136 4 4 4 4 0 0 0 0 0 0 0 [heap]
f7d73000 r-xp 00000000 103:02 2883591 1876 772 772 772 0 0 0 0 0 0 0 0 libc-2.27.so
f7f48000 ---p 001d5000 103:02 2883591 4 0 0 0 0 0 0 0 0 0 0 0 libc-2.27.so
f7f49000 r--p 001d5000 103:02 2883591 8 8 8 8 8 0 0 0 0 0 0 0 0 libc-2.27.so
f7f4b000 rw-p 001d7000 103:02 2883591 4 4 4 4 4 0 0 0 0 0 0 0 0 libc-2.27.so
f7f4c000 rw-p 00000000 00:00 0 12 8 8 8 8 0 0 0 0 0 0 0 0
f7f75000 rw-p 00000000 00:00 0 8 8 8 8 8 0 0 0 0 0 0 0 0
f7f77000 r--p 00000000 00:00 0 12 0 0 0 0 0 0 0 0 0 0 0 [vvar]
f7f7a000 r-xp 00000000 00:00 0 8 8 8 8 8 0 0 0 0 0 0 0 [vdso]
f7f7c000 r-xp 00000000 103:02 2883587 152 144 144 144 0 0 0 0 0 0 0 0 ld-2.27.so
f7fa2000 r--p 00025000 103:02 2883587 4 4 4 4 4 0 0 0 0 0 0 0 0 ld-2.27.so
f7fa3000 rw-p 00026000 103:02 2883587 4 4 4 4 4 0 0 0 0 0 0 0 0 ld-2.27.so
ffef3000 rw-p 00000000 00:00 0 132 12 12 12 12 12 0 0 0 0 0 0 0 [stack]
=====
2372 988 988 988 60 0 0 0 0 0 0 0 0 0 0 0 0 KB

```



# Memory Map of Linux Process (64 bit system)

```
ziming@ziming-ThinkPad:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/processmap$ pmap -X 22891
```

22891:	./pm64																	
	Address	Perm	Offset	Device	Inode	Size	Rss	Pss	Referenced	Anonymous	LazyFree	ShmemPmdMapped	Shared_Hugetlb	Private_Hugetlb	Swap	SwapPss	Locked	Mapping
	55bf7ae37000	r-xp	00000000	103:02	28577490	4	4	4	4	0	0	0	0	0	0	0	0	pm64
	55bf7b037000	r--p	00000000	103:02	28577490	4	4	4	4	4	0	0	0	0	0	0	0	pm64
	55bf7b038000	rw-p	00001000	103:02	28577490	4	4	4	4	4	0	0	0	0	0	0	0	pm64
	55bf7cc0c000	rw-p	00000000	00:00	0	132	4	4	4	4	0	0	0	0	0	0	0	[heap]
	7fc7ebdb6000	r-xp	00000000	103:02	660090	1948	992	5	992	0	0	0	0	0	0	0	0	libc-2.27.so
	7fc7ebf9d000	---p	001e7000	103:02	660090	2048	0	0	0	0	0	0	0	0	0	0	0	libc-2.27.so
	7fc7ec19d000	r--p	001e7000	103:02	660090	16	16	16	16	16	0	0	0	0	0	0	0	libc-2.27.so
	7fc7ec1a1000	rw-p	001eb000	103:02	660090	8	8	8	8	8	0	0	0	0	0	0	0	libc-2.27.so
	7fc7ec1a3000	rw-p	00000000	00:00	0	16	12	12	12	12	0	0	0	0	0	0	0	
	7fc7ec1a7000	r-xp	00000000	103:02	660062	156	156	0	156	0	0	0	0	0	0	0	0	ld-2.27.so
	7fc7ec3a6000	rw-p	00000000	00:00	0	8	8	8	8	8	0	0	0	0	0	0	0	
	7fc7ec3ce000	r--p	00027000	103:02	660062	4	4	4	4	4	0	0	0	0	0	0	0	ld-2.27.so
	7fc7ec3cf000	rw-p	00028000	103:02	660062	4	4	4	4	4	0	0	0	0	0	0	0	ld-2.27.so
	7fc7ec3d0000	rw-p	00000000	00:00	0	4	4	4	4	4	0	0	0	0	0	0	0	
	7ffe05803000	rw-p	00000000	00:00	0	132	12	12	12	12	0	0	0	0	0	0	0	[stack]
	7ffe058b9000	r--p	00000000	00:00	0	12	0	0	0	0	0	0	0	0	0	0	0	[vvar]
	7ffe058bc000	r-xp	00000000	00:00	0	8	4	0	4	0	0	0	0	0	0	0	0	[vdso]
	fffffffff600000	r-xp	00000000	00:00	0	4	0	0	0	0	0	0	0	0	0	0	0	[vsyscall]
						====	====	==	=====	=====	=====	=====	=====	=====	=====	=====	=====	
						4512	1236	89	1236	80	0	0	0	0	0	0	0	0 KB

# **Background Knowledge: System Calls**

# What is System Call?

When a process needs to invoke a kernel service, it invokes a procedure call in the operating system interface using special instructions (not a **call** instruction in x86). Such a procedure is called a system call.

The system call enters the kernel; the kernel performs the service and returns. Thus a process alternates between executing in user space and kernel space.

System calls are generally not invoked directly by a program, but rather via wrapper functions in glibc (or perhaps some other library).

# Popular System Call

On **Unix**, **Unix-like** and other **POSIX**-compliant operating systems, popular system calls are **open**, **read**, **write**, **close**, **wait**, **exec**, **fork**, **exit**, and **kill**.

Many modern operating systems have hundreds of system calls. For example, **Linux** and **OpenBSD** each have over 300 different calls, **FreeBSD** has over 500, Windows 7 has close to 700.

# Glibc interfaces

Often, but not always, the name of the wrapper function is the same as the name of the system call that it invokes.

For example, glibc contains a function `chdir()` which invokes the underlying "chdir" system call.

## Tools: strace & ltrace

[illegible]

**misc/firstflag**

## main.c

```
int main(int argc, char *argv[])
{
    printf("Congratulations on getting your first flag!!\n");
    print_flag();
}
```

## flag.h

```
int print_flag()
{
    FILE *fp = NULL;
    char buff[MAX_FLAG_SIZE] = {0};

    fp = fopen("/flag", "r");

    if (fp == NULL)
    {
        printf("Error: Cannot open the flag file!!!\n");
        return 1;
    }

    fread(buff, MAX_FLAG_SIZE - 2, 1, fp);
    printf("The flag is: %s\n", buff);
    fclose(fp);
    return 0;
}
```

[illegible]

Use "man 2 syscall\_name" to check out its usage

# Making a System Call in x86/64 Assembly

On x86/x86-64, most system calls rely on the software interrupt.

A software interrupt is caused either by an **exceptional condition** in the processor itself, or a **special instruction** (the **int 0x80** instruction or **syscall** instruction).

For example: a divide-by-zero exception will be thrown if the processor's arithmetic logic unit is commanded to divide a number by zero as this instruction is in error and impossible.



# Making a System Call in x86 Assembly (INT 0x80)

x86 (32-bit)

Compiled from [Linux 4.14.0 headers](#).

NR	syscall name	references	%eax	arg0 (%ebx)	arg1 (%ecx)	arg2 (%edx)	arg3 (%esi)	arg4 (%edi)	arg5 (%ebp)
0	restart_syscall	<a href="#">man/ cs/</a>	0x00	-	-	-	-	-	-
1	exit	<a href="#">man/ cs/</a>	0x01	int error_code	-	-	-	-	-
2	fork	<a href="#">man/ cs/</a>	0x02	-	-	-	-	-	-
3	read	<a href="#">man/ cs/</a>	0x03	unsigned int fd	char *buf	size_t count	-	-	-
4	write	<a href="#">man/ cs/</a>	0x04	unsigned int fd	const char *buf	size_t count	-	-	-
5	open	<a href="#">man/ cs/</a>	0x05	const char *filename	int flags	umode_t mode	-	-	-
6	close	<a href="#">man/ cs/</a>	0x06	unsigned int fd	-	-	-	-	-
7	waitpid	<a href="#">man/ cs/</a>	0x07	pid_t pid	int *stat_addr	int options	-	-	-
8	creat	<a href="#">man/ cs/</a>	0x08	const char *pathname	umode_t mode	-	-	-	-
9	link	<a href="#">man/ cs/</a>	0x09	const char *oldname	const char *newname	-	-	-	-
10	unlink	<a href="#">man/ cs/</a>	0x0a	const char *pathname	-	-	-	-	-
11	execve	<a href="#">man/ cs/</a>	0x0b	const char *filename	const char *const *argv	const char *const *envp	-	-	-
12	chdir	<a href="#">man/ cs/</a>	0x0c	const char *filename	-	-	-	-	-
13	time	<a href="#">man/ cs/</a>	0x0d	time_t *tloc	-	-	-	-	-
14	mknod	<a href="#">man/ cs/</a>	0x0e	const char *filename	umode_t mode	unsigned dev	-	-	-
15	chmod	<a href="#">man/ cs/</a>	0x0f	const char *filename	umode_t mode	-	-	-	-
16	lchown	<a href="#">man/ cs/</a>	0x10	const char *filename	uid_t user	gid_t group	-	-	-

# Making a System Call in x86 Assembly

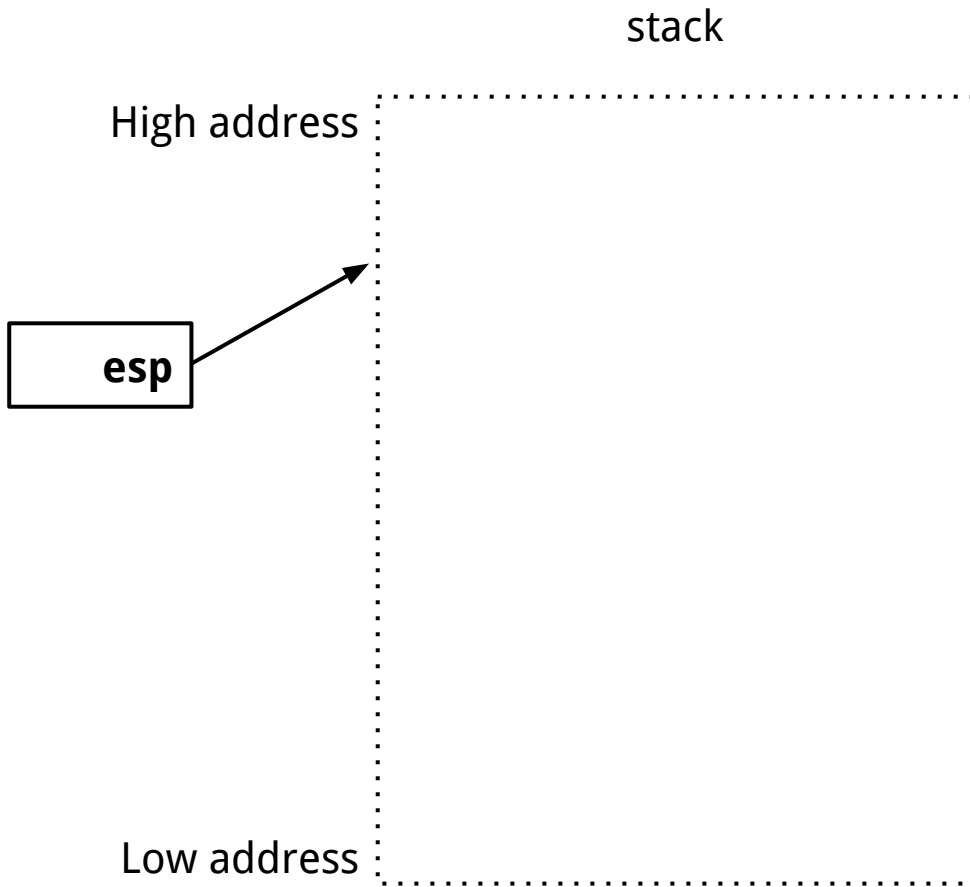
```
xor    eax,eax
push   eax
push   0x68732f2f
push   0x6e69622f
mov     ebx,esp
push   eax
push   ebx
mov     ecx,esp
mov     al,0xb
int     0x80
```

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	
0	0	000	NUL	(null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	SOH	(start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	STX	(start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	ETX	(end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	EOT	(end of transmission)	36	24	044	&#36;	\$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	ENQ	(enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	ACK	(acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	BEL	(bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	BS	(backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	TAB	(horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	LF	(NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	VT	(vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	FF	(NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	CR	(carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	SO	(shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	SI	(shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	DLE	(data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	DC1	(device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	DC2	(device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	DC3	(device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	DC4	(device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	NAK	(negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	SYN	(synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	ETB	(end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	CAN	(cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	EM	(end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	SUB	(substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	ESC	(escape)	59	3B	073	&#59;	:	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	FS	(file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	
29	1D	035	GS	(group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	RS	(record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	US	(unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: [www.LookupTables.com](http://www.LookupTables.com)

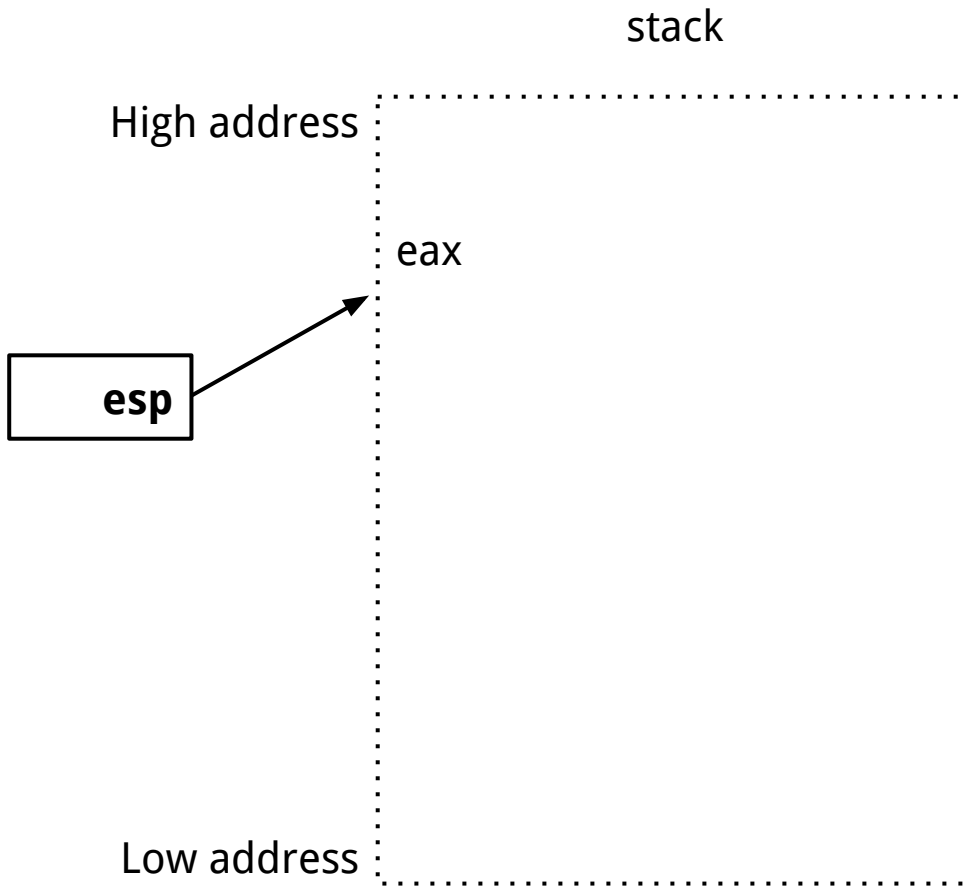
# Making a System Call in x86 Assembly

```
xor  eax,eax  
push  eax  
push  0x68732f2f  
push  0x6e69622f  
mov   ebx,esp  
push  eax  
push  ebx  
mov   ecx,esp  
mov   al,0xb  
int   0x80
```



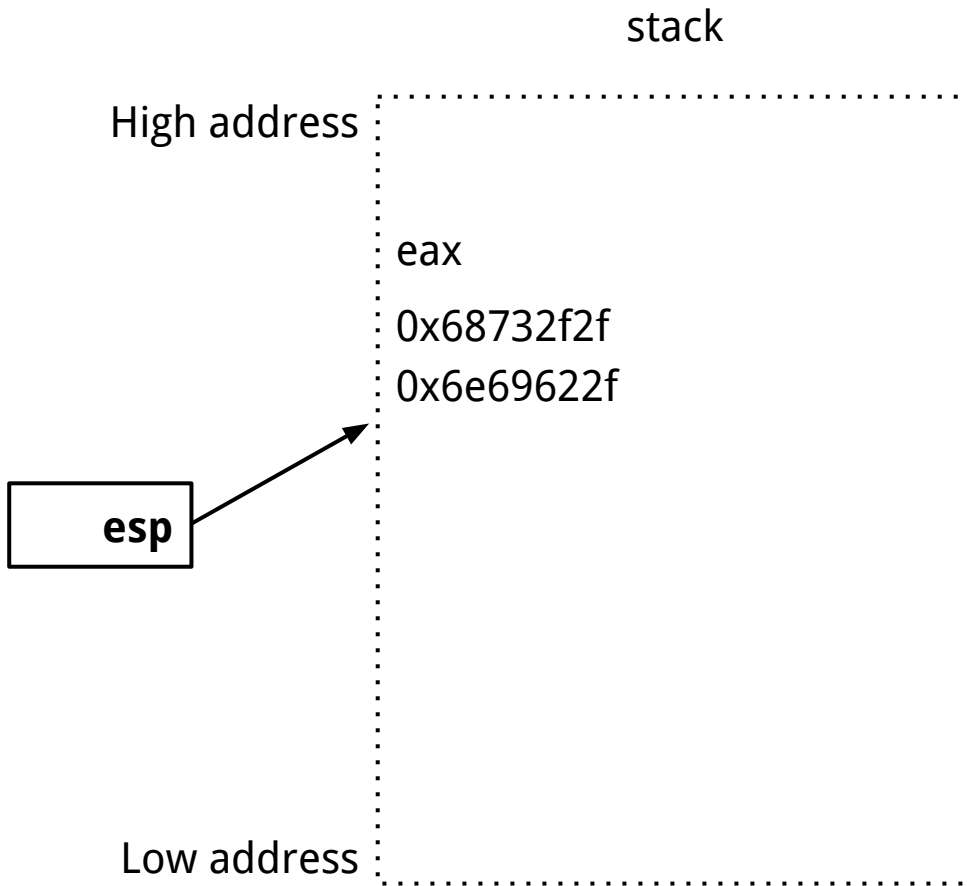
# Making a System Call in x86 Assembly

```
xor  eax,eax
push  eax
push  0x68732f2f
push  0x6e69622f
mov   ebx,esp
push  eax
push  ebx
mov   ecx,esp
mov   al,0xb
int   0x80
```



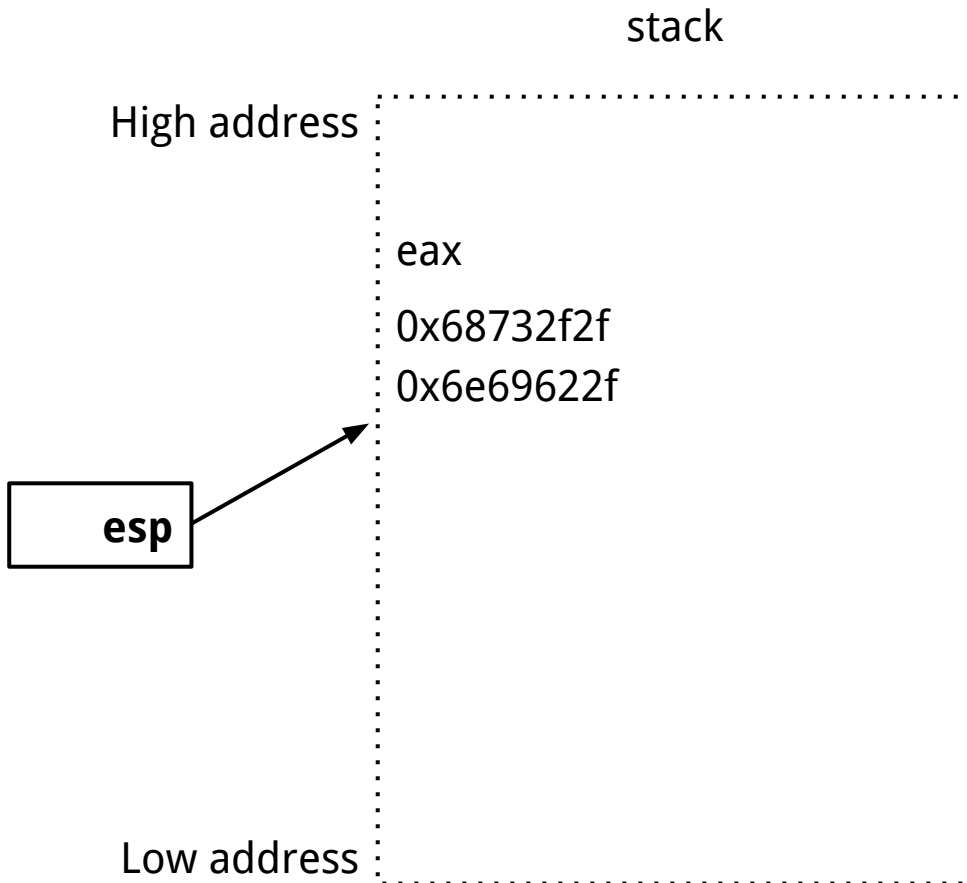
# Making a System Call in x86 Assembly

```
xor  eax,eax
push  eax
push  0x68732f2f
push  0x6e69622f
mov   ebx,esp
push  eax
push  ebx
mov   ecx,esp
mov   al,0xb
int   0x80
```



# Making a System Call in x86 Assembly

```
xor  eax,eax
push  eax
push  0x68732f2f
push  0x6e69622f
mov   ebx,esp
push  eax
push  ebx
mov   ecx,esp
mov   al,0xb
int   0x80
```



# Making a System Call in x86 Assembly

EXECVE(2) Linux Programmer's Manual

**NAME**  
execve - execute program

**SYNOPSIS**  
`#include <unistd.h>`  
  
`int execve(const char *filename, char *const argv[],  
          char *const envp[]);`

Diagram illustrating the arguments passed to `execve` and their corresponding x86 registers:

- `filename` (EBX): `/bin/sh, 0x0`
- `argv` (EDX): `0x00000000`
- `envp` (ECX): `Address of /bin/sh, 0x00000000`

`execve("/bin/sh", address of string "/bin/sh", 0)`

# Making a System Call in x86\_64 (64-bit) Assembly

x86\_64 (64-bit)

Compiled from [Linux 4.14.0 headers](#).

NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)	arg3 (%r10)	arg4 (%r8)	arg5 (%r9)
0	read	<a href="#">man/ cs/</a>	0x00	unsigned int fd	char *buf	size_t count	-	-	-
1	write	<a href="#">man/ cs/</a>	0x01	unsigned int fd	const char *buf	size_t count	-	-	-
2	open	<a href="#">man/ cs/</a>	0x02	const char *filename	int flags	umode_t mode	-	-	-
3	close	<a href="#">man/ cs/</a>	0x03	unsigned int fd	-	-	-	-	-
4	stat	<a href="#">man/ cs/</a>	0x04	const char *filename	struct __old_kernel_stat *statbuf	-	-	-	-
5	fstat	<a href="#">man/ cs/</a>	0x05	unsigned int fd	struct __old_kernel_stat *statbuf	-	-	-	-
6	lstat	<a href="#">man/ cs/</a>	0x06	const char *filename	struct __old_kernel_stat *statbuf	-	-	-	-
7	poll	<a href="#">man/ cs/</a>	0x07	struct pollfd *ufds	unsigned int nfds	int timeout	-	-	-
8	lseek	<a href="#">man/ cs/</a>	0x08	unsigned int fd	off_t offset	unsigned int whence	-	-	-
9	mmap	<a href="#">man/ cs/</a>	0x09	?	?	?	?	?	?
10	mprotect	<a href="#">man/ cs/</a>	0x0a	unsigned long start	size_t len	unsigned long prot	-	-	-
11	munmap	<a href="#">man/ cs/</a>	0x0b	unsigned long addr	size_t len	-	-	-	-
12	brk	<a href="#">man/ cs/</a>	0x0c	unsigned long brk	-	-	-	-	-
13	rt_sigaction	<a href="#">man/ cs/</a>	0x0d	int	const struct sigaction *	struct sigaction *	size_t	-	-

[https://chromium.googlesource.com/chromiumos/docs/+master/constants/syscalls.md#x86-32\\_bit](https://chromium.googlesource.com/chromiumos/docs/+master/constants/syscalls.md#x86-32_bit)



# Making a System Call in x86\_64 (64-bit) Assembly

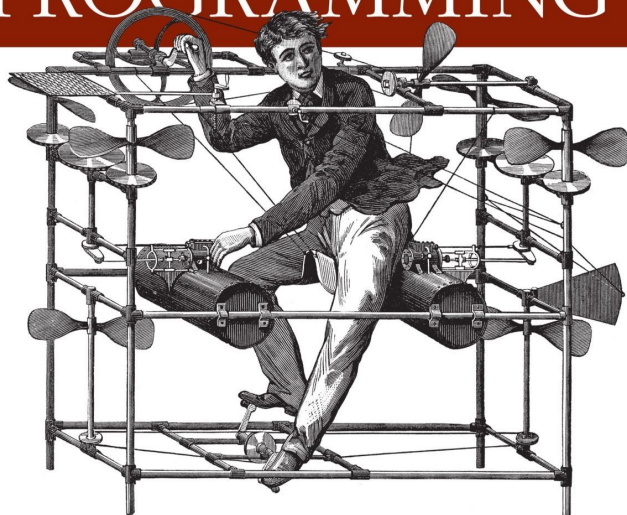
NR	syscall name	references	%rax	arg0 (%rdi)	arg1 (%rsi)	arg2 (%rdx)	arg3 (%r10)	arg4 (%r8)	arg5 (%r9)
59	execve	<a href="#">man/ cs/</a>	0x3b	const char *filename	const char *const *argv	const char *const *envp	-	-	-

```
push rax
xor rdx, rdx
xor rsi, rsi
mov rbx, '/bin//sh'
push rbx
push rsp
pop rdi
mov al, 59
syscall
```

SYSTEM AND LIBRARY CALLS EVERY PROGRAMMER NEEDS TO KNOW

# LINUX

## SYSTEM PROGRAMMING



O'REILLY®

ROBERT LOVE

# **Background Knowledge: Piping**

# Channels of Communication for Linux Process

Every process in Linux has three initial, standard channels of communication:

- Standard Input (stdin, fd=0) is the channel through which the process takes input. For example, your shell uses Standard Input to read the commands that you input.
- Standard Output (stdout, fd=1) is the channel through which processes output normal data, such as the flag when it is printed to you in previous challenges or the output of utilities such as `ls`.
- Standard Error (stderr, fd=2) is the channel through which processes output error details. For example, if you mistype a command, the shell will output, over standard error, that this command does not exist.

# Examples

## Redirecting output > or 1>

*echo hi > asdf      echo hi 1> asdf*

## Appending output >>

*echo hi >> asdf*

## Redirecting errors 2>

*/challenge/run 2> errors.log*

## Redirecting input <

*rev < messagefile*

# Channels of Communication for Linux Process

- Process can also take input from command line arguments

*ls -al*

*cat /flag*

*cat 1.txt 2.txt 3.txt*

# Pipe

The | (pipe) operator. Standard output from the command to the left of the pipe will be connected to (piped into) the standard input of the command to the right of the pipe.

*echo hello-world | wc -c*

# **Background Knowledge: Environment and Shell Variables**



# Environment and Shell Variables

Environment and Shell variables are a set of dynamic **named values**, stored within the system that are used by applications launched in shells.

KEY=value

KEY="Some other value"

KEY=value1:value2

The names of the variables are case-sensitive (UPPER CASE).

Multiple values must be separated by the colon **:** character.

There is no space around the equals **=** symbol.

# Environment and Shell Variables

Environment variables are variables that are available **system-wide** and are **inherited** by all spawned child processes and shells.

Shell variables are variables that apply only to the **current shell instance**. Each shell such as zsh and bash, has its own set of internal shell variables.

# Common Environment Variables

USER - The current logged in user.

HOME - The home directory of the current user.

EDITOR - The default file editor to be used. This is the editor that will be used when you type edit in your terminal.

SHELL - The path of the current user's shell, such as bash or zsh.

LOGNAME - The name of the current user.

PATH - A list of directories to be searched when executing commands.

LANG - The current locales settings.

TERM - The current terminal emulation.

MAIL - Location of where the current user's mail is stored.

# Commands

*env* – The command allows you to run another program in a custom environment without modifying the current one. When used without an argument it will print a list of the current environment variables.

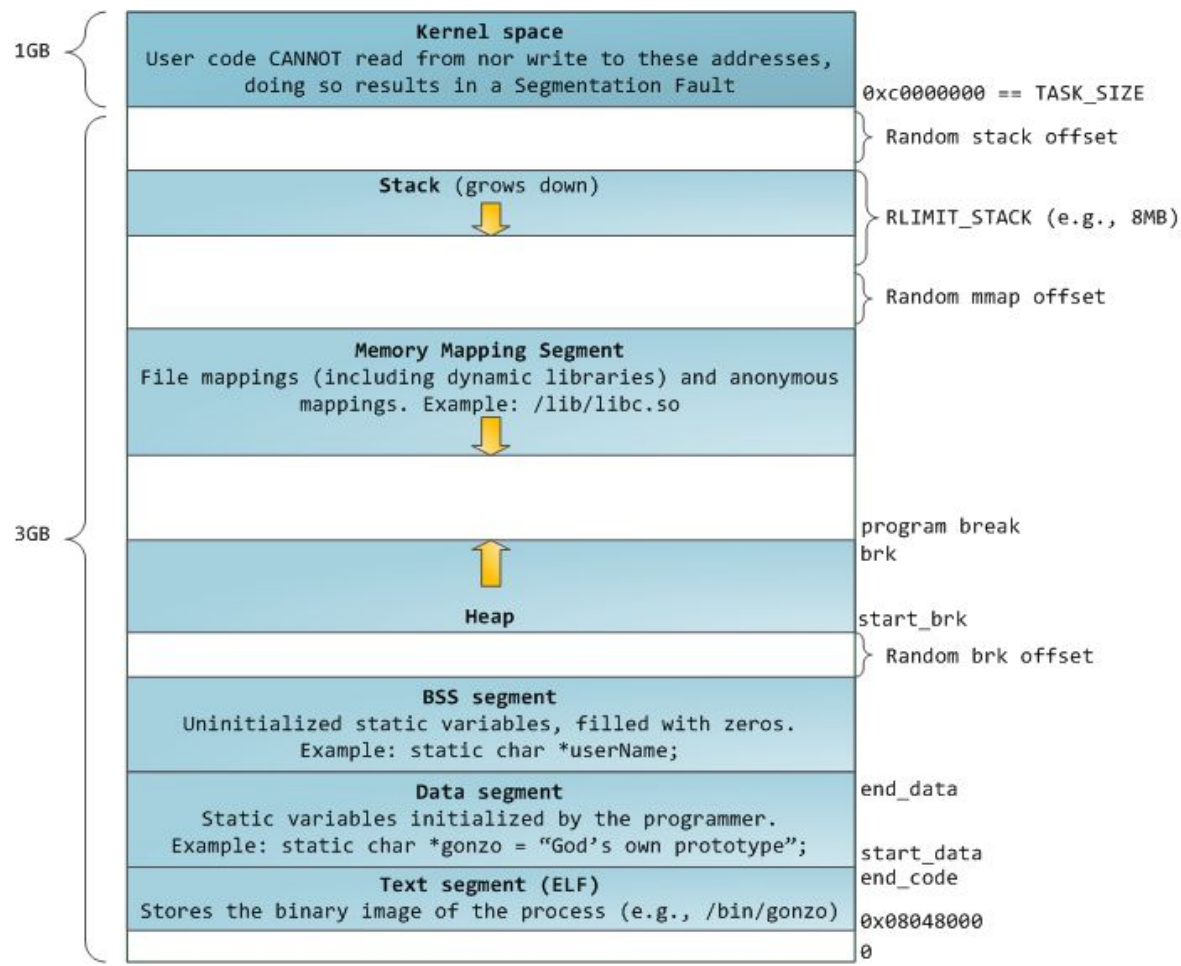
*printenv* – The command prints all or the specified environment variables.

*set* – The command sets or unsets shell variables. When used without an argument it will print a list of all variables including environment and shell variables, and shell functions.

*unset* – The command deletes shell and environment variables.

*export* – The command sets environment variables

The environment variables live towards the top of the stack, together with command line arguments.

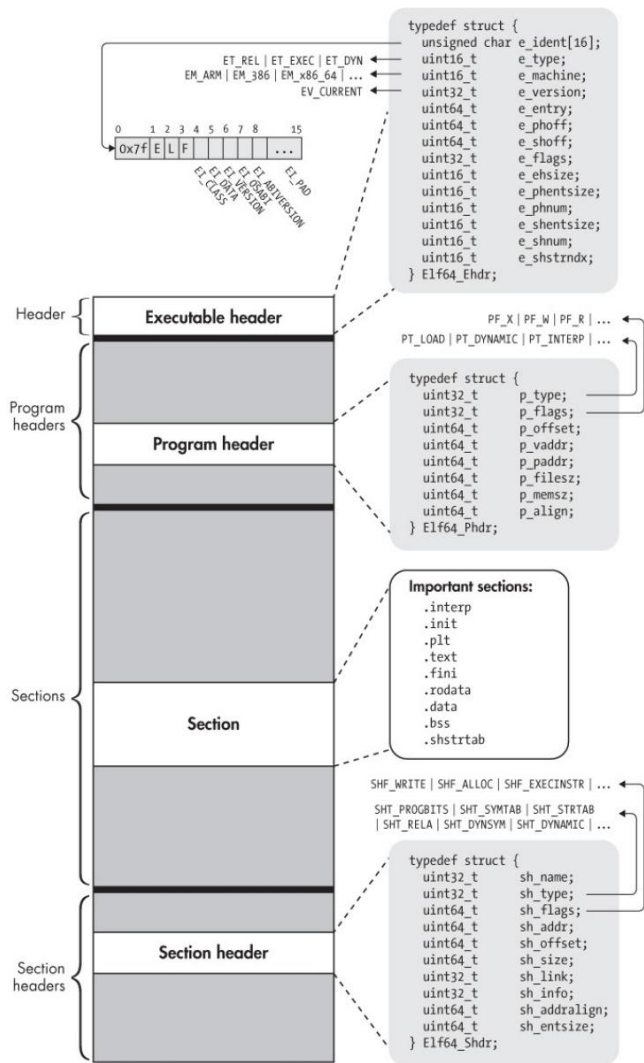


# **Background Knowledge: Executable and Linkable Format (ELF)**

# ELF Files

The **Executable and Linkable Format (ELF)** is a common standard file format for *executable files, object code, shared libraries, and core dumps*. Filename extension *none, .axf, .bin, .elf, .o, .prx, .puff, .ko, .mod* and *.so*

Contains the program and its data. Describes how the program should be loaded (program/segment headers). Contains metadata describing program components (section headers).



- Executable (a.out), object files (.o), shared libraries (.a), even core dumps.
- Four *types* of components: an **executable header**, a series of (optional) **program headers**, a number of **sections**, and a series of (optional) **section headers**, one per section.



# Executable Header

---

```
typedef struct {
    unsigned char e_ident[16]; /* Magic number and other info */ 0x7F ELF ..
    uint16_t e_type; /* Object file type Executable, obj, dynamic lib */
    uint16_t e_machine; /* Architecture x86-64, Arm */
    uint32_t e_version; /* Object file version */
    uint64_t e_entry; /* Entry point virtual address */
    uint64_t e_phoff; /* Program header table file offset */
    uint64_t e_shoff; /* Section header table file offset */
    uint32_t e_flags; /* Processor-specific flags */
    uint16_t e_ehsize; /* ELF header size in bytes */
    uint16_t e_phentsize; /* Program header table entry size */
    uint16_t e_phnum; /* Program header table entry count */
    uint16_t e_shentsize; /* Section header table entry size */
    uint16_t e_shnum; /* Section header table entry count */
    uint16_t e_shstrndx; /* Section header string table index */
} Elf64_Ehdr;
```

---

⋮ readelf -h a.out ⋮

```
→ add readelf -h /bin/ls
```

```
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:                 0x67d0
  Start of program headers:            64 (bytes into file)
  Start of section headers:            140224 (bytes into file)
  Flags:                               0x0
  Size of this header:                 64 (bytes)
  Size of program headers:             56 (bytes)
  Number of program headers:           13
  Size of section headers:             64 (bytes)
  Number of section headers:           30
  Section header string table index: 29
```

# Sections

The code and data in an ELF binary are logically divided into contiguous non-overlapping chunks called sections. The structure of each section varies depending on the contents.

The division into sections is intended to provide a convenient organization for use by the ***linker***.

# Section Header Format

```
typedef struct {
    uint32_t  sh_name;      /* Section name (string tbl index) */
    uint32_t  sh_type;      /* Section type */
    uint64_t  sh_flags;     /* Section flags */
    uint64_t  sh_addr;      /* Section virtual addr at execution */
    uint64_t  sh_offset;    /* Section file offset */
    uint64_t  sh_size;      /* Section size in bytes */
    uint32_t  sh_link;      /* Link to another section */
    uint32_t  sh_info;      /* Additional section information */
    uint64_t  sh_addralign; /* Section alignment */
    uint64_t  sh_entsize;   /* Entry size if section holds table */
} Elf64_Shdr;
```

SHF\_WRITE | SHF\_ALLOC | SHF\_EXECINSTR | ...  
SHT\_PROGBITS | SHT\_SYMTAB | SHT\_STRTAB  
| SHT\_RELA | SHT\_DYNSYM | SHT\_DYNAMIC | ...

```
typedef struct {
    uint32_t  sh_name;
    uint32_t  sh_type;
    uint64_t  sh_flags;
    uint64_t  sh_addr;
    uint64_t  sh_offset;
    uint64_t  sh_size;
    uint32_t  sh_link;
    uint32_t  sh_info;
    uint64_t  sh_addralign;
    uint64_t  sh_entsize;
} Elf64_Shdr;
```

Each section is described by its section header.

```
readelf -S a.out
```

# sh\_flags

SHF\_WRITE: the section is writable at runtime.

SHF\_ALLOC: the contents of the section are to be loaded into virtual memory when executing the binary.

SHF\_EXECINSTR: the section contains executable instructions.

SHF\_WRITE | SHF\_ALLOC | SHF\_EXECINSTR | ...  
SHT\_PROGBITS | SHT\_SYMTAB | SHT\_STRTAB  
| SHT\_RELA | SHT\_DYNSYM | SHT\_DYNAMIC | ...

```
typedef struct {  
    uint32_t    sh_name;  
    uint32_t    sh_type;  
    uint64_t    sh_flags;  
    uint64_t    sh_addr;  
    uint64_t    sh_offset;  
    uint64_t    sh_size;  
    uint32_t    sh_link;  
    uint32_t    sh_info;  
    uint64_t    sh_addralign;  
    uint64_t    sh_entsize;  
} Elf64_Shdr;
```

```
→ add readelf -S add
There are 31 section headers, starting at offset 0x385c:
```

#### Section Headers:

[Nr]	Name	Type	Addr	Off	Size	ES	Flg	Lk	Inf	Al
[ 0]		NULL	00000000	000000	000000	00		0	0	0
[ 1]	.interp	PROGBITS	000001b4	0001b4	000013	00	A	0	0	1
[ 2]	.note.gnu.build-i	NOTE	000001c8	0001c8	000024	00	A	0	0	4
[ 3]	.note.gnu.propert	NOTE	000001ec	0001ec	00001c	00	A	0	0	4
[ 4]	.note.ABI-tag	NOTE	00000208	000208	000020	00	A	0	0	4
[ 5]	.gnu.hash	GNU_HASH	00000228	000228	000020	04	A	6	0	4
[ 6]	.dynsym	DYSYM	00000248	000248	0000a0	10	A	7	1	4
[ 7]	.dynstr	STRTAB	000002e8	0002e8	0000bb	00	A	0	0	1
[ 8]	.gnu.version	VERSYM	000003a4	0003a4	000014	02	A	6	0	2
[ 9]	.gnu.version_r	VERNEED	000003b8	0003b8	000040	00	A	7	1	4
[10]	.rel.dyn	REL	000003f8	0003f8	000040	08	A	6	0	4
[11]	.rel.plt	REL	00000438	000438	000020	08	AI	6	24	4
[12]	.init	PROGBITS	00001000	001000	000024	00	AX	0	0	4
[13]	.plt	PROGBITS	00001030	001030	000050	04	AX	0	0	16
[14]	.plt.got	PROGBITS	00001080	001080	000010	10	AX	0	0	16
[15]	.plt.sec	PROGBITS	00001090	001090	000040	10	AX	0	0	16
[16]	.text	PROGBITS	000010d0	0010d0	000259	00	AX	0	0	16
[17]	.fini	PROGBITS	0000132c	00132c	000018	00	AX	0	0	4
[18]	.rodata	PROGBITS	00002000	002000	000025	00	A	0	0	4
[19]	.eh_frame_hdr	PROGBITS	00002028	002028	000054	00	A	0	0	4
[20]	.eh_frame	PROGBITS	0000207c	00207c	00014c	00	A	0	0	4
[21]	.init_array	INIT_ARRAY	00003ed0	002ed0	000004	04	WA	0	0	4
[22]	.fini_array	FINI_ARRAY	00003ed4	002ed4	000004	04	WA	0	0	4
[23]	.dynamic	DYNAMIC	00003ed8	002ed8	0000f8	08	WA	7	0	4
[24]	.got	PROGBITS	00003fd0	002fd0	000030	04	WA	0	0	4
[25]	.data	PROGBITS	00004000	003000	000008	00	WA	0	0	4
[26]	.bss	NOBITS	00004008	003008	000004	00	WA	0	0	1
[27]	.comment	PROGBITS	00000000	003008	00002a	01	MS	0	0	1
[28]	.symtab	SYMTAB	00000000	003034	000490	10		29	47	4
[29]	.strtab	STRTAB	00000000	0034c4	00027d	00		0	0	1
[30]	.shstrtab	STRTAB	00000000	003741	000118	00		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),  
p (processor specific)

readelf -S a.out



# Sections

.init: executable code that performs initialization tasks and needs to run before any other code in the binary is executed.

.fini: code that runs after the main program completes.

.text: where the main code of the program resides.

# Sections

.rodata section, which stands for “read-only data,” is dedicated to storing constant values. Because it stores constant values, .rodata is not writable.

The default values of initialized variables are stored in the .data section, which is marked as writable since the values of variables may change at runtime.

the .bss section reserves space for uninitialized variables. The name historically stands for “block started by symbol,” referring to the reserving of blocks of memory for (symbolic) variables.



# Lazy Binding (.plt, .got, .got.plt Sections)

**Binding at Load Time:** When a binary is loaded into a process for execution, the dynamic linker resolves references to functions located in shared libraries. The addresses of shared functions were not known at compile time.

**In reality - Lazy Binding:** many of the relocations are typically not done right away when the binary is loaded but are deferred until the first reference to the unresolved location is actually made.

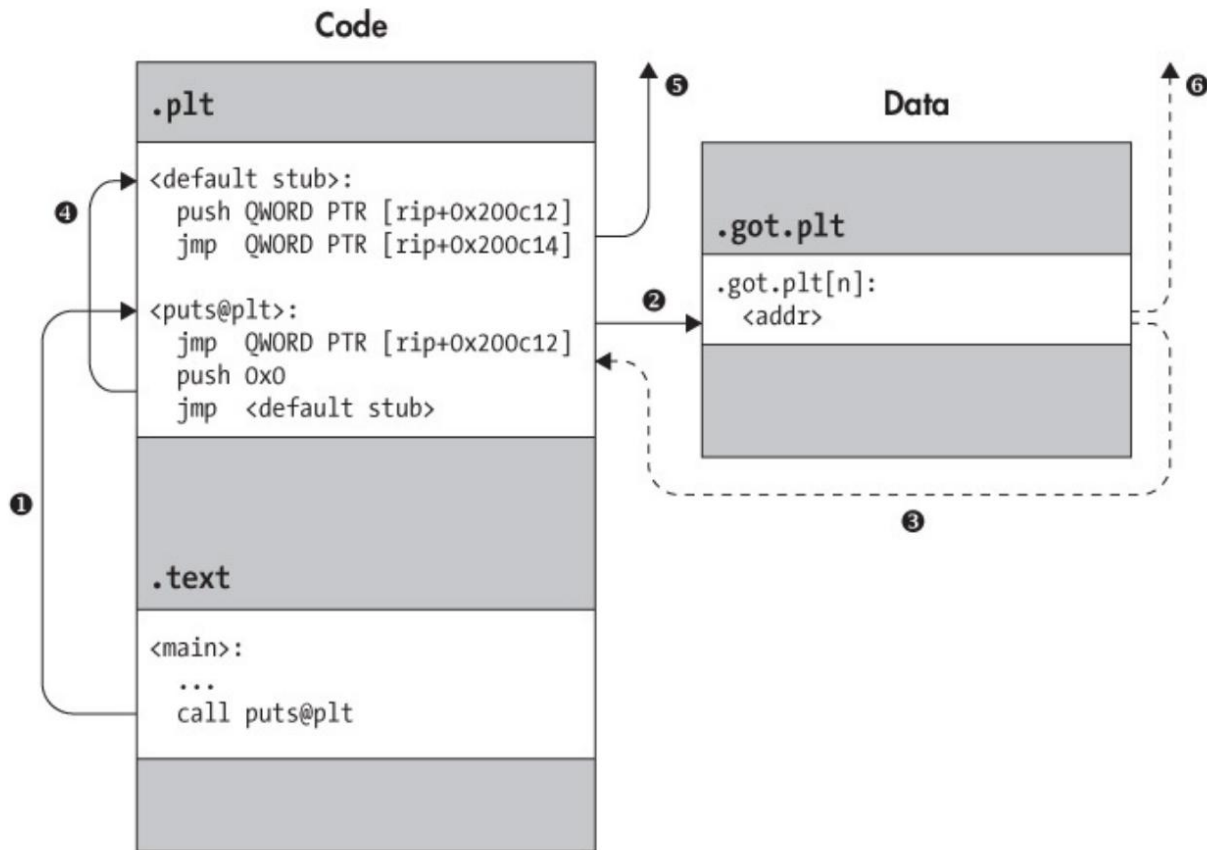
# Lazy Binding (.plt, .got, .got.plt Sections)

Lazy binding in Linux ELF binaries is implemented with the help of two special sections, called the Procedure Linkage Table ( .plt ) and the Global Offset Table ( .got ).

.plt is a code section that contains executable code. The PLT consists entirely of stubs of a well-defined format, dedicated to directing calls from the .text section to the appropriate library location.

.got.plt is a data section.

# Dynamically Resolving a Library Function Using the PLT



# Example: Debug misc/lazyb

```
0028| 0xffffc63c --> 0xf7dcee5 (<_libc_start_main+245>: add esp,0x10)
Legend: code, data, rodata, value
0x56556070 in main ()
gdb-peda$ si
[-----registers-----]
EAX: 0x5655701e ("Second call to printf.")
EBX: 0x56559000 --> 0x3efc
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7f99000 --> 0x1eadc
EDI: 0xf7f99000 --> 0x1eadc
EBP: 0xffffc638 --> 0x0
ESP: 0xfffffc3c ("fbUV\036pUV\344\306\377\377\354\306\377\377\345aUVP\306\377\377")
EIP: 0x56556074 (<puts@plt>: endbr32)
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGM trap INTERRUPT direction overflow)
[-----code-----]
0x56556060 <__cxa_finalize@plt>: endbr32
0x56556064 <__cxa_finalize@plt+4>: jmp DWORD PTR [ebx+0x10]
0x5655606a <__cxa_finalize@plt+10>: nop WORD PTR [eax+eax*1+0x0]
=> 0x56556070 <puts@plt>: endbr32
0x56556074 <puts@plt+4>: jmp DWORD PTR [ebx+0xc]
0x5655607a <puts@plt+10>: nop WORD PTR [eax+eax*1+0x0]
0x56556080 <__libc_start_main@plt>: endbr32
0x56556084 <__libc_start_main@plt+4>: jmp DWORD PTR [ebx+0x10]
[-----stack-----]
0000| 0xffffc61c ("fbUV\036pUV\344\306\377\377\354\306\377\377\345aUVP\306\377\377")
0004| 0xfffffc20 --> 0x5655701e ("Second call to printf.")
0008| 0xfffffc24 --> 0xfffffc64 --> 0xffffc03 ("home/z/lnng/Dropbox/myTeaching/Software Security UB 2021 Fall/code/lazybinding/lazyb")
0012| 0xfffffc28 --> 0xfffffc6c --> 0xffffc09 ("COLORTERM=truecolor")
0016| 0xfffffc2c --> 0x56556105 (<main+24>: add ebx,0x2e1b)
0020| 0xfffffc30 --> 0xfffffc50 --> 0x1
0024| 0xfffffc34 --> 0x0
0028| 0xfffffc38 --> 0x0
Legend: code, data, rodata, value
0x56556070 in puts@plt ()
gdb-peda$
[-----registers-----]
EAX: 0x5655701e ("Second call to printf.")
EBX: 0x56559000 --> 0x3efc
ECX: 0xffffffff
EDX: 0xffffffff
ESI: 0xf7f99000 --> 0x1eadc
EDI: 0xf7f99000 --> 0x1eadc
EBP: 0xfffffc38 --> 0x0
ESP: 0xfffffc3c ("fbUV\036pUV\344\306\377\377\354\306\377\377\345aUVP\306\377\377")
EIP: 0x56556074 (<puts@plt+4>: jmp DWORD PTR [ebx+0xc])
EFLAGS: 0x296 (carry PARITY ADJUST zero SIGM trap INTERRUPT direction overflow)
[-----code-----]
0x56556064 <__cxa_finalize@plt+4>: jmp DWORD PTR [ebx+0x10]
0x5655606a <__cxa_finalize@plt+10>: nop WORD PTR [eax+eax*1+0x0]
0x56556070 <puts@plt>: endbr32
=> 0x56556074 <puts@plt+4>: jmp DWORD PTR [ebx+0xc]
| 0x5655607a <puts@plt+10>: nop WORD PTR [eax+eax*1+0x0]
| 0x56556080 <__libc_start_main@plt>: endbr32
| 0x56556084 <__libc_start_main@plt+4>: jmp DWORD PTR [ebx+0x10]
| 0x5655608a <__libc_start_main@plt+10>: nop WORD PTR [eax+eax*1+0x0]
|> 0xf7e1fc00 <__GI__IO_puts>: endbr32
0xf7e1fc04 <__GI__IO_puts+4>: push ebp
0xf7e1fc05 <__GI__IO_puts+5>: mov ebp,esp
0xf7e1fc07 <__GI__IO_puts+7>: push edi
JUMP is taken
[-----stack-----]
0000| 0xffffc61c ("fbUV\036pUV\344\306\377\377\354\306\377\377\345aUVP\306\377\377")
0004| 0xfffffc20 --> 0x5655701e ("Second call to printf.")
0008| 0xfffffc24 --> 0xfffffc64 --> 0xffffc03 ("home/z/lnng/Dropbox/myTeaching/Software Security UB 2021 Fall/code/lazybinding/lazyb")
0012| 0xfffffc28 --> 0xfffffc6c --> 0xffffc09 ("COLORTERM=truecolor")
0016| 0xfffffc2c --> 0x56556105 (<main+24>: add ebx,0x2e1b)
0020| 0xfffffc30 --> 0xfffffc50 --> 0x1
0024| 0xfffffc34 --> 0x0
0028| 0xfffffc38 --> 0x0
Legend: code, data, rodata, value
0x56556074 in puts@plt ()
gdb-peda$
[0] 0:gdb*
```

GDB Cheatsheet:

<https://darkdust.net/files/GDB%20Cheat%20Sheet.pdf>

# **Section View (Section Header) vs. Segment View (Program Header)**

The program header table provides a segment view of the binary, as opposed to the section view provided by the section header table.

The section view of an ELF binary is meant for static linking purposes.

The segment view is used by the operating system and dynamic linker when loading an ELF into a process for execution to locate the relevant code and data and decide what to load into virtual memory.

Segments are simply a bunch of sections bundled together.

# Program Header Format

---

```
typedef struct {
    uint32_t  p_type;      /* Segment type          */
    uint32_t  p_flags;     /* Segment flags          */
    uint64_t  p_offset;    /* Segment file offset    */
    uint64_t  p_vaddr;     /* Segment virtual address */
    uint64_t  p_paddr;     /* Segment physical address */
    uint64_t  p_filesz;    /* Segment size in file   */
    uint64_t  p_memsz;     /* Segment size in memory  */
    uint64_t  p_align;     /* Segment alignment      */
} Elf64_Phdr;
```

---

Each section is described by its section header.

```
readelf -l a.out
```

```
PT_LOAD | PT_DYNAMIC | PT_INTERP | ...
PF_X | PF_W | PF_R | ...

typedef struct {
    uint32_t  p_type;
    uint32_t  p_flags;
    uint64_t  p_offset;
    uint64_t  p_vaddr;
    uint64_t  p_paddr;
    uint64_t  p_filesz;
    uint64_t  p_memsz;
    uint64_t  p_align;
} Elf64_Phdr;
```

Report bugs to [ncp@tj.cnki.net](mailto:ncp@tj.cnki.net) or [bugzilla@redhat.com](mailto:bugzilla@redhat.com)

→ add readelf -l add

Elf file type is DYN (Shared object file)

Entry point 0x1160

There are 12 program headers, starting at offset 52

Program Headers:

Type	Offset	VirtAddr	PhysAddr	FileSiz	MemSiz	Flg	Align
PHDR	0x000034	0x00000034	0x00000034	0x00180	0x00180	R	0x4
INTERP	0x0001b4	0x000001b4	0x000001b4	0x00013	0x00013	R	0x1
[Requesting program interpreter: /lib/ld-linux.so.2]							
LOAD	0x000000	0x00000000	0x00000000	0x00458	0x00458	R	0x1000
LOAD	0x001000	0x00001000	0x00001000	0x00344	0x00344	R E	0x1000
LOAD	0x002000	0x00002000	0x00002000	0x001c8	0x001c8	R	0x1000
LOAD	0x002ed0	0x00003ed0	0x00003ed0	0x00138	0x0013c	RW	0x1000
DYNAMIC	0x002ed8	0x00003ed8	0x00003ed8	0x000f8	0x000f8	RW	0x4
NOTE	0x0001c8	0x000001c8	0x000001c8	0x00060	0x00060	R	0x4
GNU_PROPERTY	0x0001ec	0x000001ec	0x000001ec	0x0001c	0x0001c	R	0x4
GNU_EH_FRAME	0x002028	0x00002028	0x00002028	0x00054	0x00054	R	0x4
GNU_STACK	0x000000	0x00000000	0x00000000	0x00000	0x00000	RW	0x10
GNU_RELRO	0x002ed0	0x00003ed0	0x00003ed0	0x00130	0x00130	R	0x1

Section to Segment mapping:

Segment Sections...

00	
01	.interp
02	.interp .note.gnu.build-id .note.gnu.property .note.ABI-tag .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt
03	.init .plt .plt.got .plt.sec .text .fini
04	.rodata .eh_frame_hdr .eh_frame
05	.init_array .fini_array .dynamic .got .data .bss
06	.dynamic
07	.note.gnu.build-id .note.gnu.property .note.ABI-tag
08	.note.gnu.property
09	.eh_frame_hdr
10	
11	.init_array .fini_array .dynamic .got

→ add

[0] 0:zsh\*

# **Background Knowledge: Manual Binary Analysis Tools**



# Tools for this class

file

readelf

strings

nm

objdump

GDB

[optional] IDA Pro

[optional] ghidra

[optional] Binary Ninja

# GDB Cheat Sheet

Start gdb using:

`gdb <binary>`

Pass initial commands for gdb through a file

`gdb <binary> -x <initfile>`

To start the program and breakpoint at main()

`start <argv>`

To start the program and breakpoint at \_start

`starti <argv>`

To run the program without breakpoint

`r <argv>`

Use another program's output as stdin in GDB:

`r <<< $(python2 -c "print '\x12\x34'*5")`

# GDB Cheat Sheet

Set breakpoint at address:

`b *0x80000000`

Set breakpoint at beginning of a function:

`b main`

....

`b <filename:line number>`

`b <line number>`

Disassemble 10 instructions from an address:

`x/10i 0x80000000`

Exam 15 dword (w) from an address; show hex (x):

`x/15wx 0x80000000`

Exam 3 qword (g) from an address; show hex (x):

`x/3gx 0x80000000`

# GDB Cheat Sheet

To show breakpoints

`info b`

To remove breakpoints

`clear <function name>`

`clear *<instruction address>`

`clear <filename:line number>`

`clear <line number>`

# GDB Cheat Sheet

Use “examine” or “x” command

`x/32xw <memory location>` to see memory contents at memory location, showing 32 hexadecimal words

`x/5s <memory location>` to show 5 strings (null terminated) at a particular memory location

`x/10i <memory location>` to show 10 instructions at particular memory location

See registers

`info reg`

Step an instruction

`si`

# GDB Script

Use “examine” or “x” command

`x/32xw <memory location>` to see memory contents at memory location, showing 32 hexadecimal words

`x/5s <memory location>` to show 5 strings (null terminated) at a particular memory location

`x/10i <memory location>` to show 10 instructions at particular memory location

See registers

`info reg`

Step an instruction

`si`

# Shell Cheat Sheet

Run a program and use another program's output as a parameter

program `$(python2 -c "print '\x12\x34'*5")`

# Reading

1. <https://iq.thc.org/how-does-linux-start-a-process>