CSE 610 Special Topics:
System Security - Attack and Defense for Binaries

Instructor: Dr. Ziming Zhao

Location: Online

Time: Monday, 5:20 PM - 8:10 PM

First off, Logistics!

Turn on camera if possible
Classes are recorded and released publicly

Have a notebook in front of you
From the second class, have the hacking environment ready

Webpage: https://zzm7000.github.io/teaching/2020fallcse610/index.html

Virtual machine: https://www.dropbox.com/s/38udm6klh4jo7nx/CSE610VM.zip?dl=0

Feel free to interrupt me and ask questions
Eat or drink if you need

Instructor

Dr. Ziming Zhao
Assistant Professor, CSE
Director, CyberspAce seCuriTy and forensIcs Lab (CactiLab)

Email: zimingzh@buffalo.edu
http://zzm7000.github.io
http://cactilab.github.io

Office: 338B Davis Hall / Online
Office hours: By appointment

Students - UB CSE 610 Graduates (3 credits)

Graduate Students (Master, PhD) who take this as CSE 610 (3-credit)

Graduate students who take 3-credit class will be invited to slack ***cacti-workspace***, ***#ubcse610private-fall2020***

Students - UB Undergraduate (No credit)

Join the slack ***cacti-workspace***, ***#ubcse610systemsecurity-fall2020***

Treat this as an opening hacking seminar. No string attached.

Course Goals

To provide you with good understanding of the **theories**, **principles**, **techniques** and **tools** used for software and system hacking and hardening.

You will study, in-depth, binary reverse engineering, vulnerability classes, vulnerability analysis, exploit/shellcode development, defensive solutions, etc. to understand how to crack and protect **native** software.

You will get your hands dirty.

Quick Poll

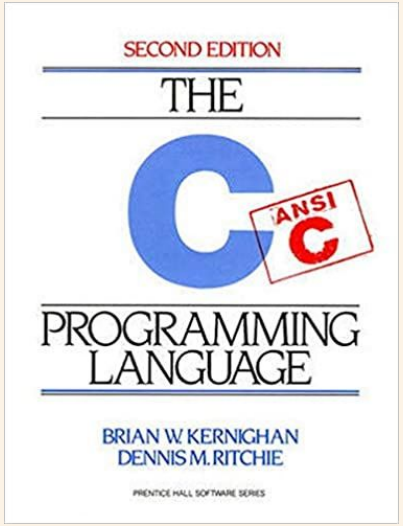
1. Which year of undergraduate and graduate you are in?

2. Did you take any security class before?
3. Did you take the “operating system” class?
4. Do you consider yourself a *nix user?
5. Do you have any hacking experience (binary, web, etc.)?

Today's Agenda

1. Class overview and logistics
2. Background knowledge
    a. Compiler, linker, loader
    b. x86 and x86-64 architectures and ISA
    c. Linux file permissions
    d. Set-UID programs
    e. Memory map of a Linux process
    f. System calls
    g. Environment and Shell variables
    h. Basic reverse engineering

Prerequisites



The real prerequisite:
The C Programming Language

Classes that will help you understand this class:
CSE 521 Operating Systems

Other skills:

Reverse engineering (Using objdump, IDA Pro, Ghidra, etc.)

Debugging (GDB, pwngdb)

Google, reading, self-learning, getting hands dirty

8 Topics

Binary attack and defense using x86 and x86-64 as examples. Discover **vulnerabilities**. Develop **exploits**. Memory corruption attacks (1 - 7).

1. Stack-based buffer overflow (2 session)
2. Defenses against stack-based buffer overflow (2)
3. Shellcode development (2)
4. Format string vulnerabilities (1)
5. Heap-based buffer overflow (1)
6. Integer overflow (1)
7. Return-oriented programming (2)
8. Cache side-channel attack, meltdown, spectre (2)

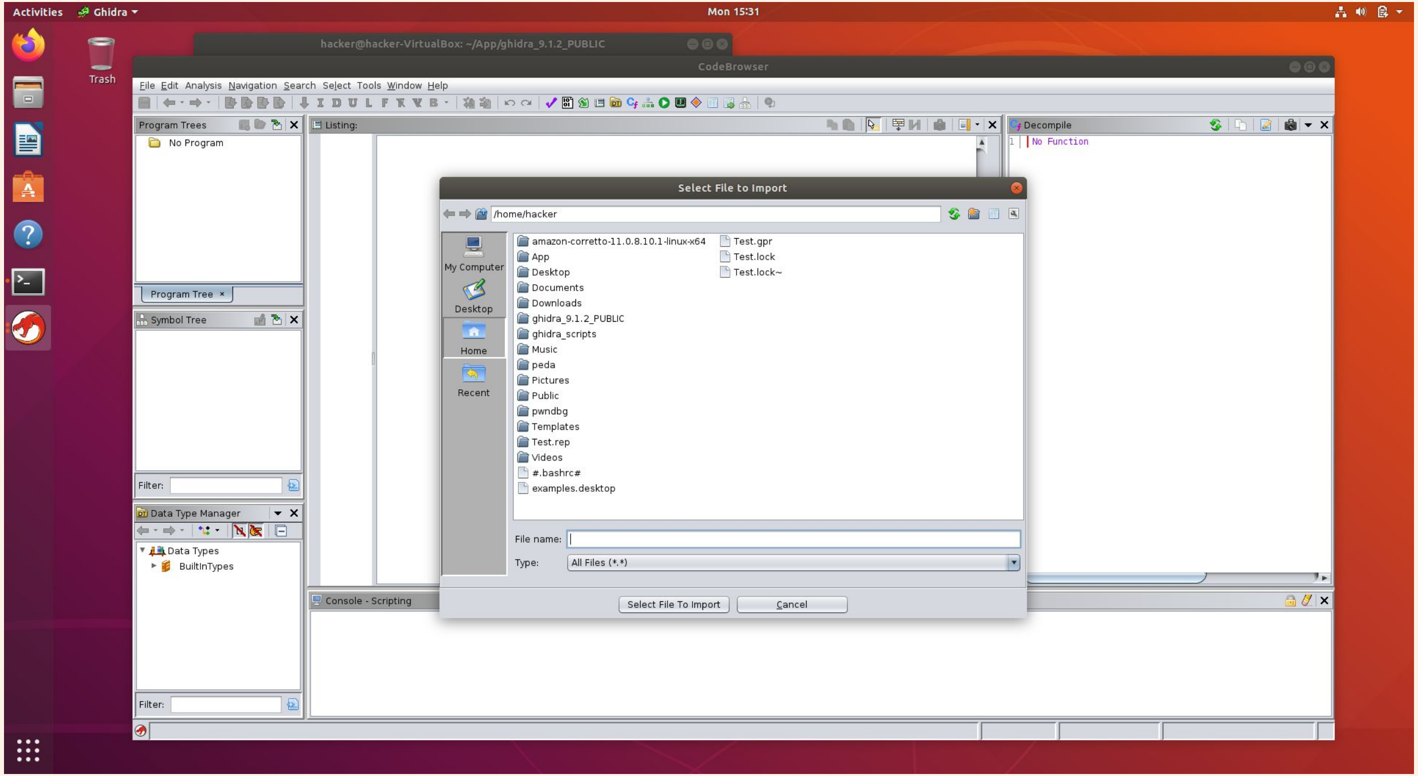
The Hacking Environment

Intel x86
x86-64, a.k.a amd64
Linux (Ubuntu)

Pwngdb
GDB peda
NSA Ghidra

The VM

User: hacker pwd: rekcah link:



Homework

Reading: whitepaper, paper, blog, etc.
Hands-on: hacking, debugging, etc.

Submit before the next class on UBLearns. We will discuss homework at the beginning of each class.

30% penalty if you submit within 10 mins after class starts.
0 points after 10 mins.

Hacking Assignment Rules

- For each hacking assignment, you will submit your exploit, a simple write-up, and screenshots to show it works
  - Simple write-up:
    - Briefly describe how you solve the challenge
    - Mention who you worked with if any in the write-up

- Discussion is encouraged. But, you cannot share your code, exploits, write-ups to your classmates or post them online.

Exams

Open-book; Asynchronous?; Written midterm and final

In-class CTF

In the last class. 1.5 - 2 hours.

Grades

Area	No. Items	Points per Item	Points for Area
Exams			200
Midterm		100	
Final		100	
Homework (Submit answers or reports at Blackboard)	14	45	630
In-class CTF	1	200	200
Attendance	14	2	28
Total			1058

Table 2: Grades Breakdown

Points	Grade
930 -	A
900 - 930	A-
870 - 900	B+
830 - 870	B
800 - 830	B-
770 - 800	C+
700 - 770	C
670 - 700	D+
600 - 670	D
0 - 600	F
Academic Dishonesty	>F<

Table 3: Final Letter Grades

Academic Integrity

- Discussion is encourage. But, you cannot share your code, exploits to your classmates or post them online.

- The university, college, and department policies against academic dishonesty will be strictly enforced. To understand your responsibilities as a student read: UB Student Code of Conduct.

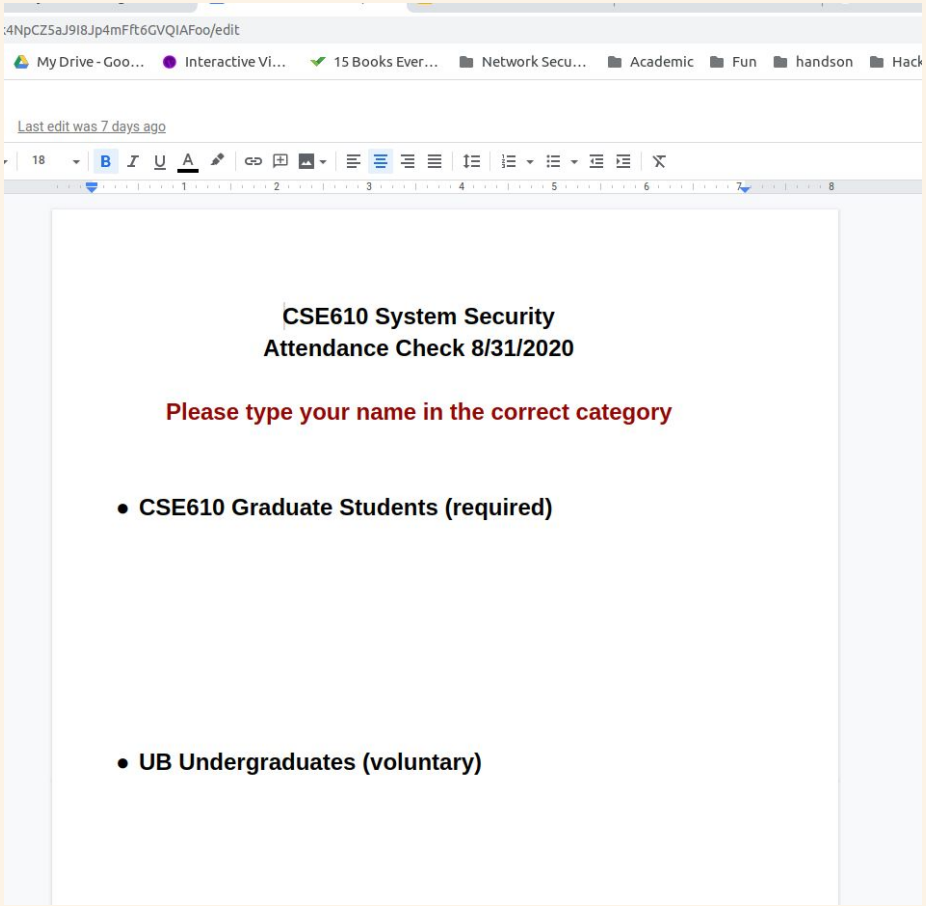
- Plagiarism or any form of cheating in homework, assignments, labs, or exams is subject to serious academic penalty.

- Any violation of the academic integrity policy will result in a 0 on the homework, lab or assignment, and even an **F** or **>F<** on the final grade. And, the violation will be reported to the Dean's office.

Ethical Hacking

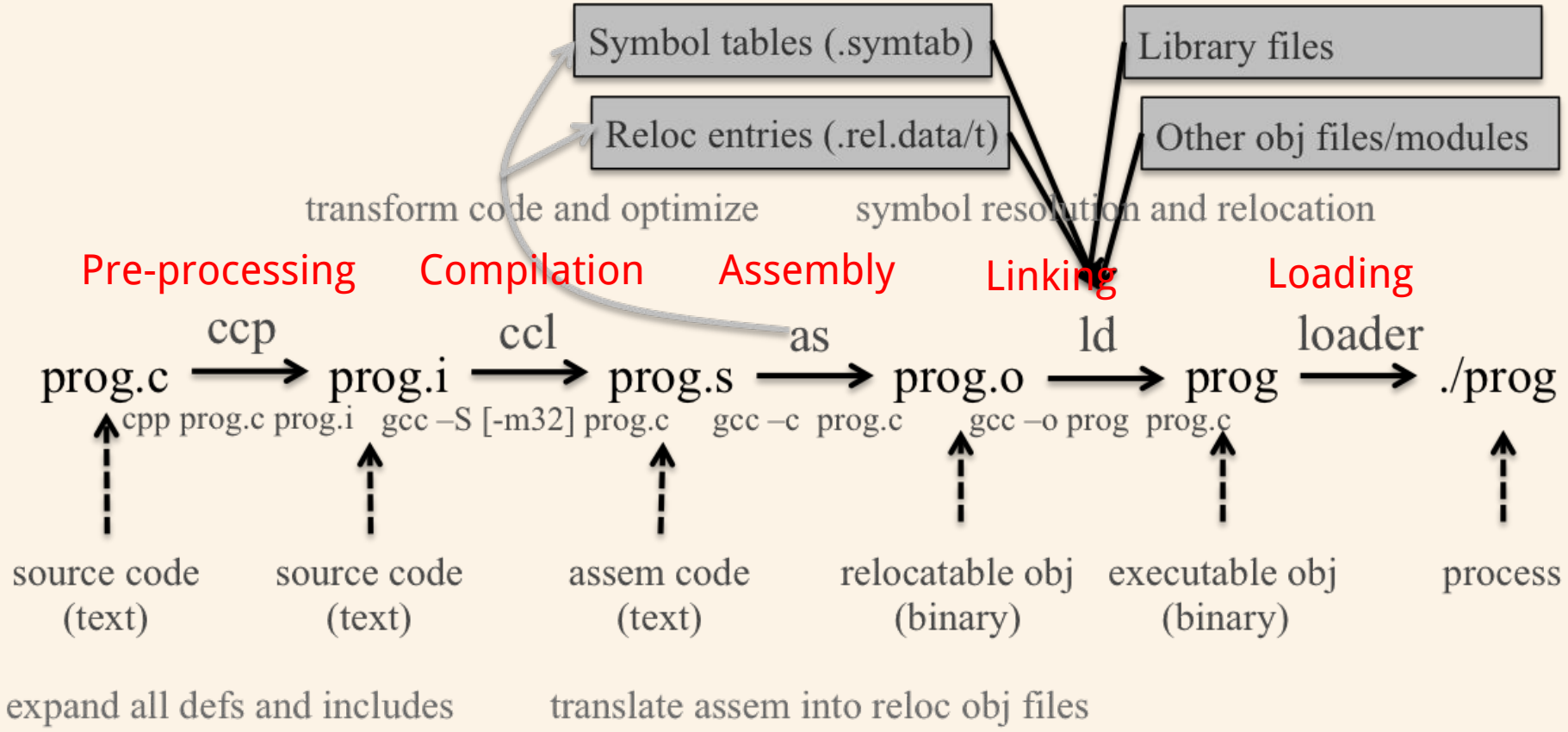
- Do not attempt to violate the law.
- If you discover real-world vulnerabilities using the knowledge you learn from this class, report the vulnerabilities responsibly.

Attendance Check



Background Knowledge: Compiler, linker and loader

From a C program to a process



Loader, e.g. Handler of execve() in Linux

1. Validation (permissions, memory requirements etc.)
2. Copying the program image from the disk into main memory
3. Copying the command-line arguments on the stack
4. Initializing registers (e.g., the stack pointer)
5. Jumping to the program entry point (_start)

Compiling a C program behind the scene (code/add)

add.c

```
#include "add.h"

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

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>

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

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

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

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

```
gcc -Wall -save-temps -O2 add.c main.c -o add64
```

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.

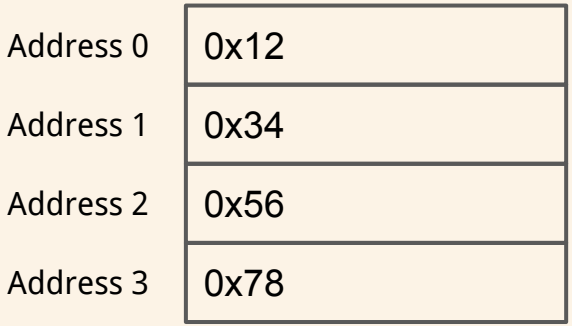
Endianness

- 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

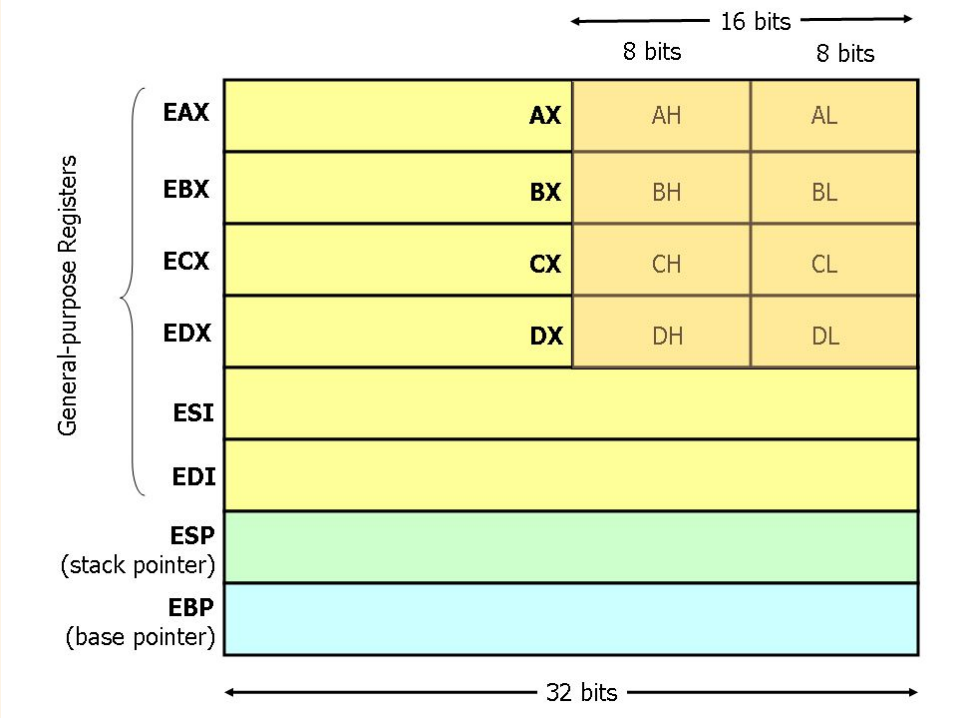


Base Registers

There are

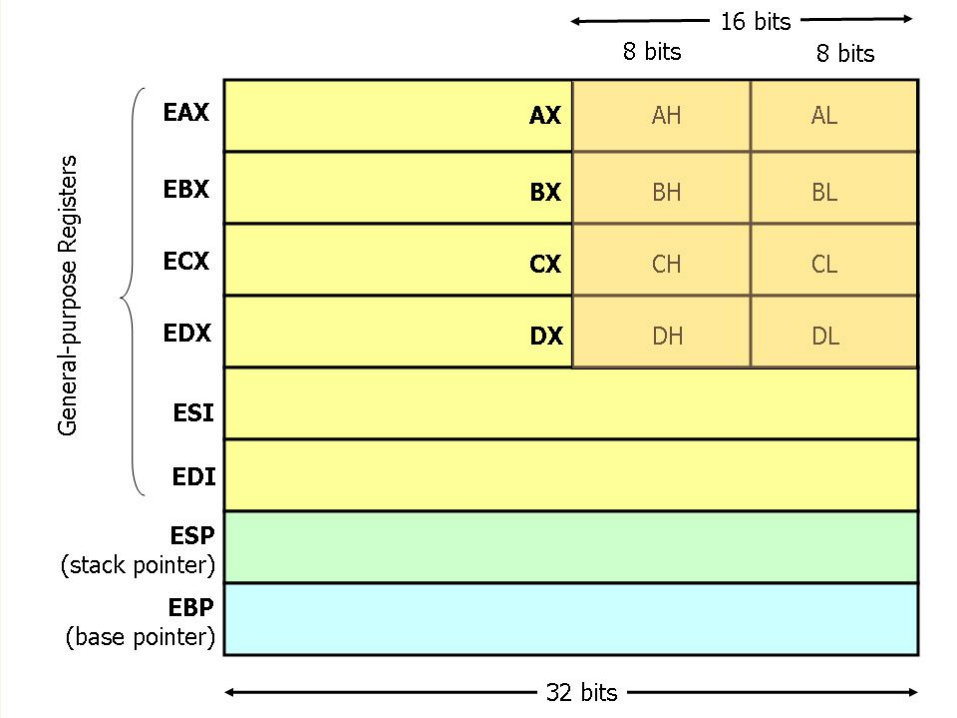
- Eight 32-bit “general-purpose” registers,
- One 32-bit EFLAGS register,
- One 32-bit instruction pointer register (eip), and
- Other special-purpose registers.

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.

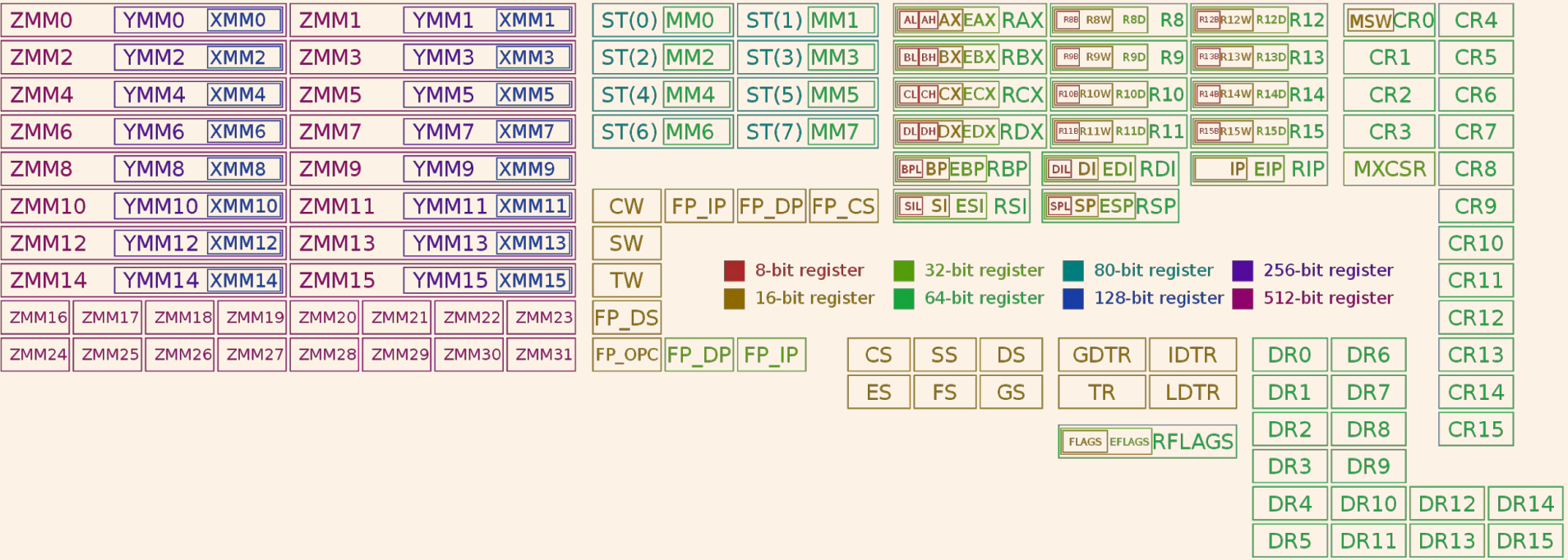
We will be interested in, at most, the bits

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.
Register eip holds the address of the **next** instruction to be executed.

Registers on x86 and amd64



https://en.wikipedia.org/wiki/X86

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>

AT&T 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), 4+var(,1), or (%eax,%ebx,1))
<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 1) to destination (operand 2)

mov (%ebx), %eax Load 4 bytes from the memory address in EBX into EAX.

mov -4(%esi), %eax Move 4 bytes at memory address ESI + (-4) into EAX. */

mov %cl, (%esi,%eax,1) Move the contents of CL into the byte at address ESI+EAX*1.

mov (%esi,%ebx,4), %edx Move the 4 bytes of data at address ESI+4*EBX into EDX.

Addressing Memory

The size prefixes b, w, l, q (x86-64) indicate sizes of 1, 2, 4, and 8 (x86-64) bytes respectively.

mov $2, (%ebx) isn't this ambiguous? We can have a default.

movb $2, (%ebx) Move 2 into the single byte at the address stored in EBX.

movw $2, (%ebx) Move the 16-bit integer representation of 2 into the 2 bytes starting at the address in EBX.

movl $2, (%ebx) 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 <con>, <reg>
mov <con>, <mem>

Examples
mov %ebx, %eax — copy the value in EBX into EAX
movb $5, var(,1) — store the value 5 into the byte at location var

Data Movement Instructions

push — Push on stack

Syntax
push <reg32>
push <mem>
push <con32>

Examples
push %eax — push eax on 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.

Data Movement Instructions

lea — Load effective address; used for quick calculation

Syntax
lea <mem>, <reg32>

Examples
lea (%ebx,%esi,8), %edi — the quantity EBX+8*ESI is placed in EDI.

Arithmetic and Logic Instructions

add $10, %eax — EAX is set to EAX + 10
addb $10, (%eax) — add 10 to the single byte stored at memory address stored in EAX

sub %ah, %al — AL is set to AL - AH
sub $216, %eax — subtract 216 from the value stored in EAX

dec %eax — subtract one from the contents of EAX

imul (%ebx), %eax — multiply the contents of EAX by the 32-bit contents of the memory at location EBX. Store the result in EAX.

shr %cl, %ebx — Store in EBX the floor of result of dividing the value of EBX by 2n 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>

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
cmpb $10, (%ebx)
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 %esp, %ebp
Sub #imm, %esp
```

Stack Instructions

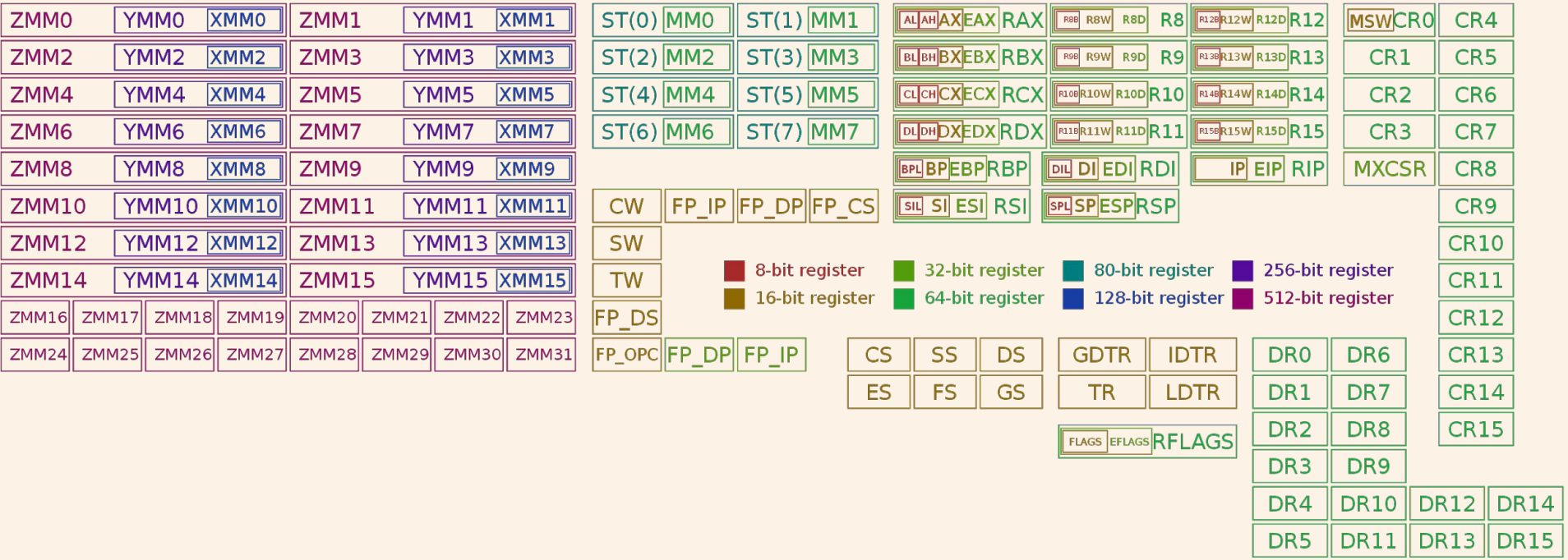
leave — Releases the function frame set up by an earlier ENTER instruction.

Equivalent to:

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

Background Knowledge: amd64 architecture

Registers on x86 and x86-64



https://en.wikipedia.org/wiki/X86

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;
}
```

```
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));
}
```

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

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

x86 vs. x86-64 (code/ladd)

x86

```
00000640 <ladd>:
 640:  8b 44 24 04        mov    0x4(%esp),%eax
 644:  8b 50 04           mov    0x4(%eax),%edx
 647:  8b 00              mov    (%eax),%eax
 649:  03 44 24 08        add    0x8(%esp),%eax
 64d:  13 54 24 0c        adc    0xc(%esp),%edx
 651:  c3                 ret
```

x86-64

```
0000000000000780 <ladd>:
 780:  48 8b 07           mov    (%rdi),%rax
 783:  48 01 f0           add    %rsi,%rax
 786:  c3                 retq
```

objdump -d ladd
objdump -d ladd64

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 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

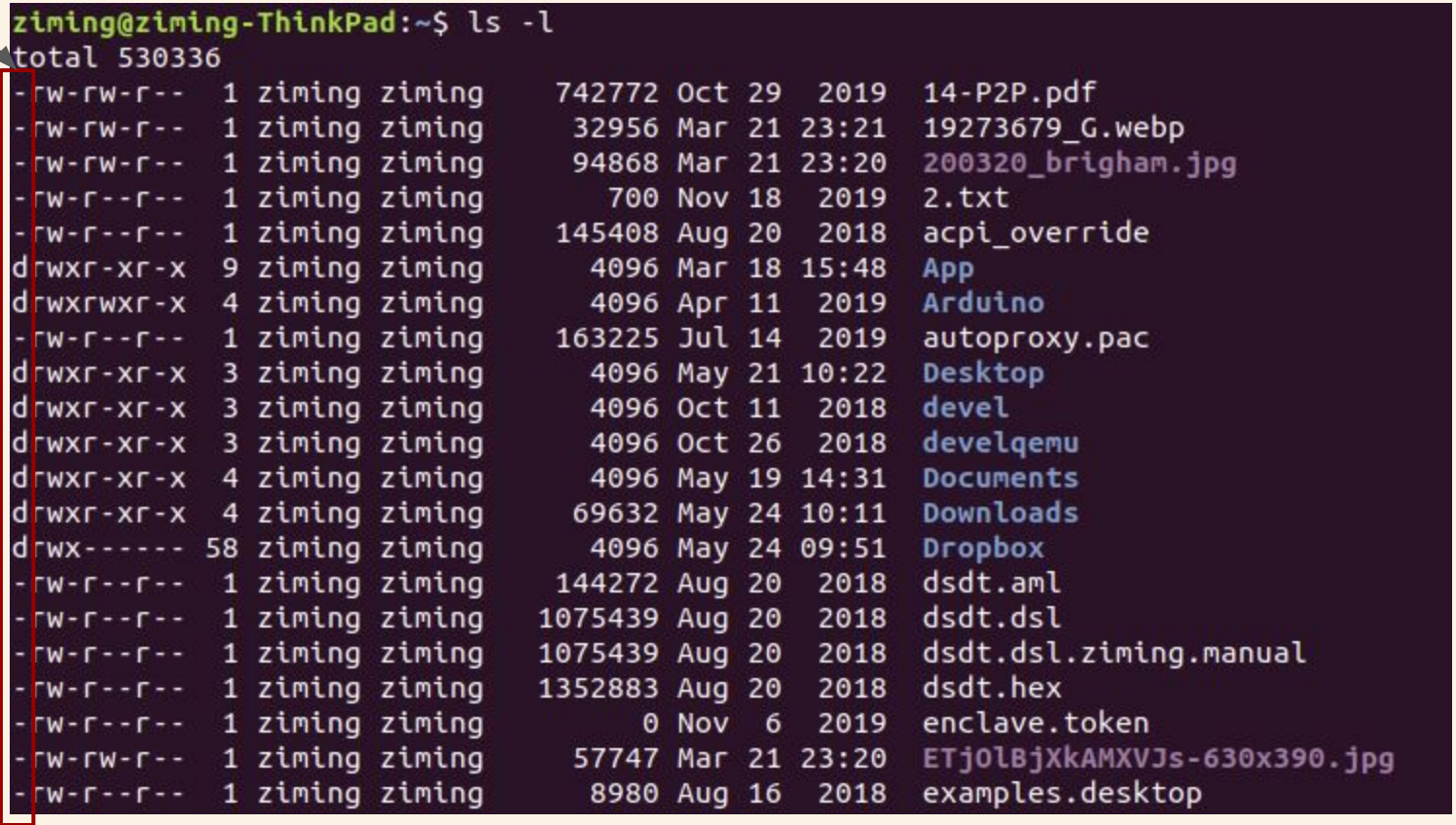
Each file or directory has three basic permission types defined for all the 3 user types:

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

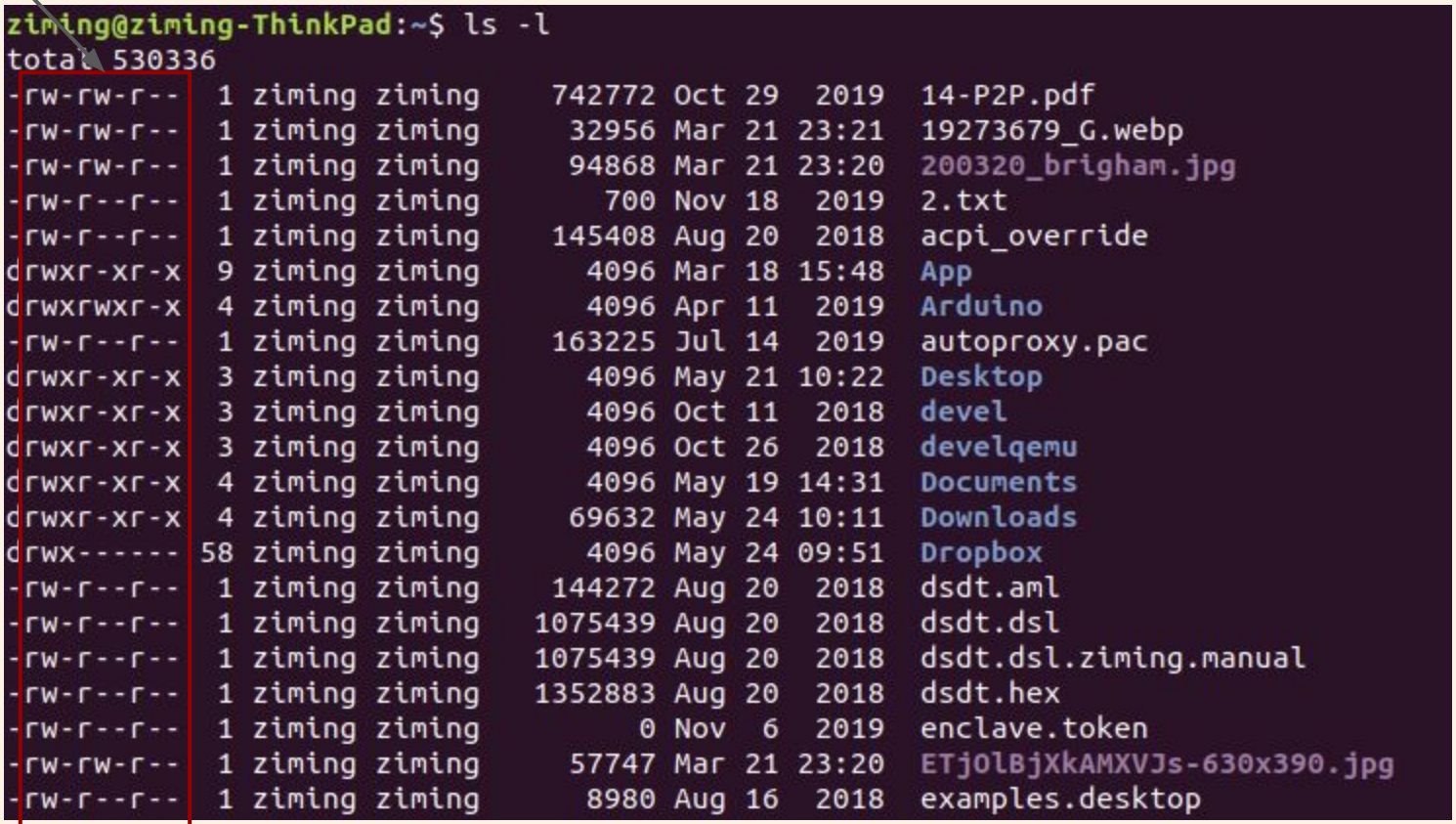
Write – The Write permissions refer to a user’s capability to write or modify a file or directory.

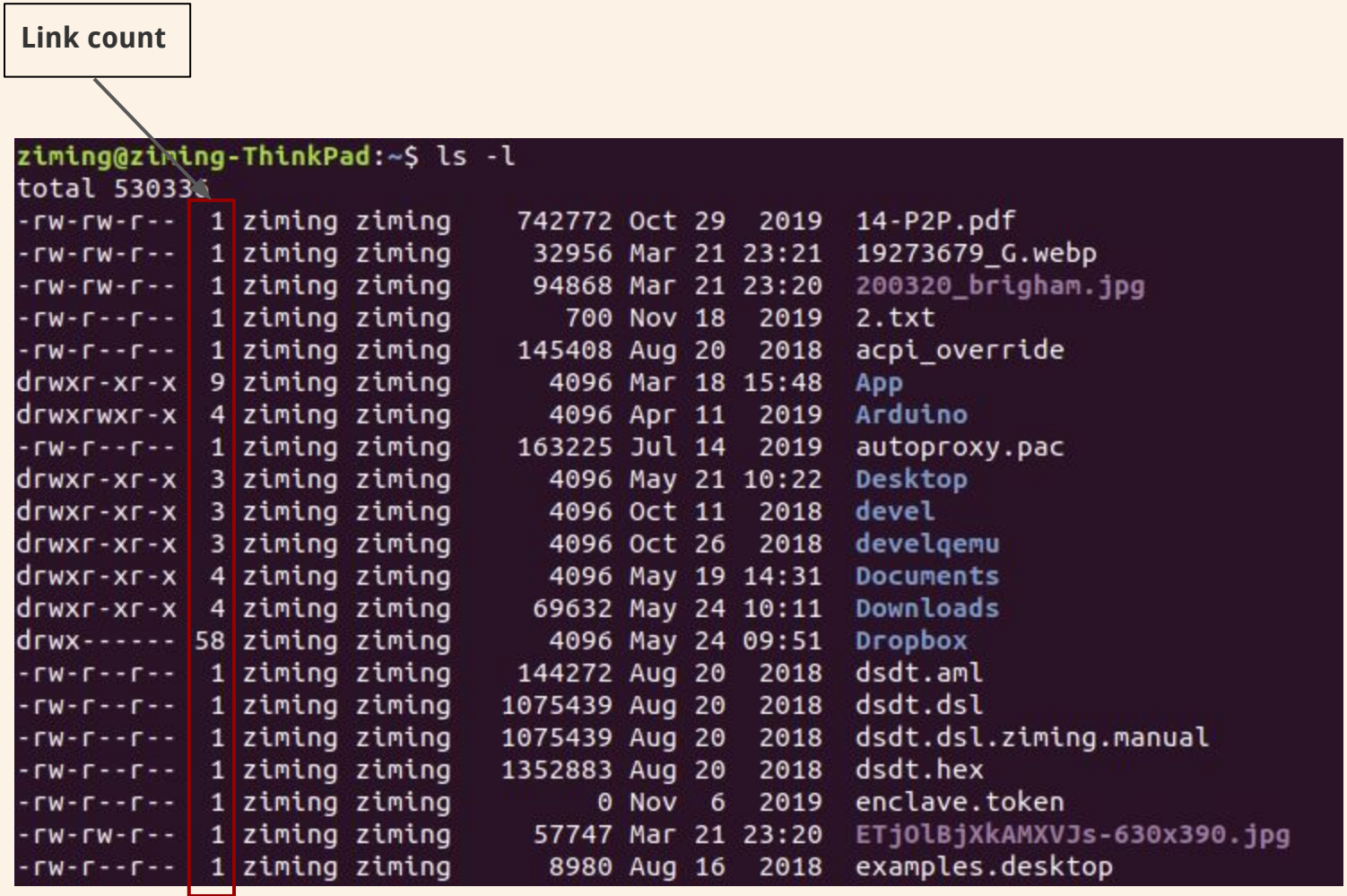
Execute – The Execute permission affects a user’s capability to execute a file or view the contents of a directory.

File type: First field in the output is file type. If the 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.

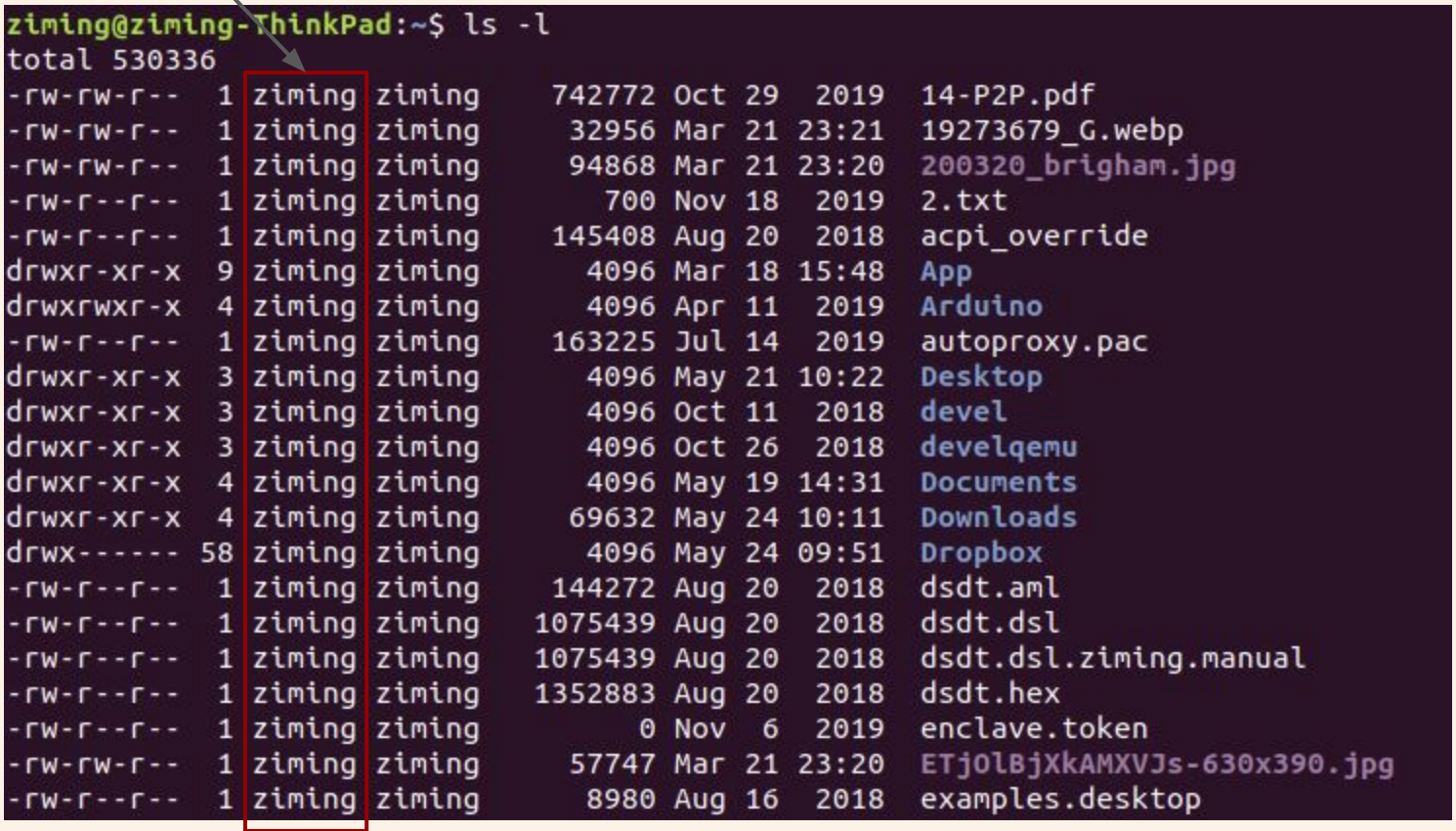


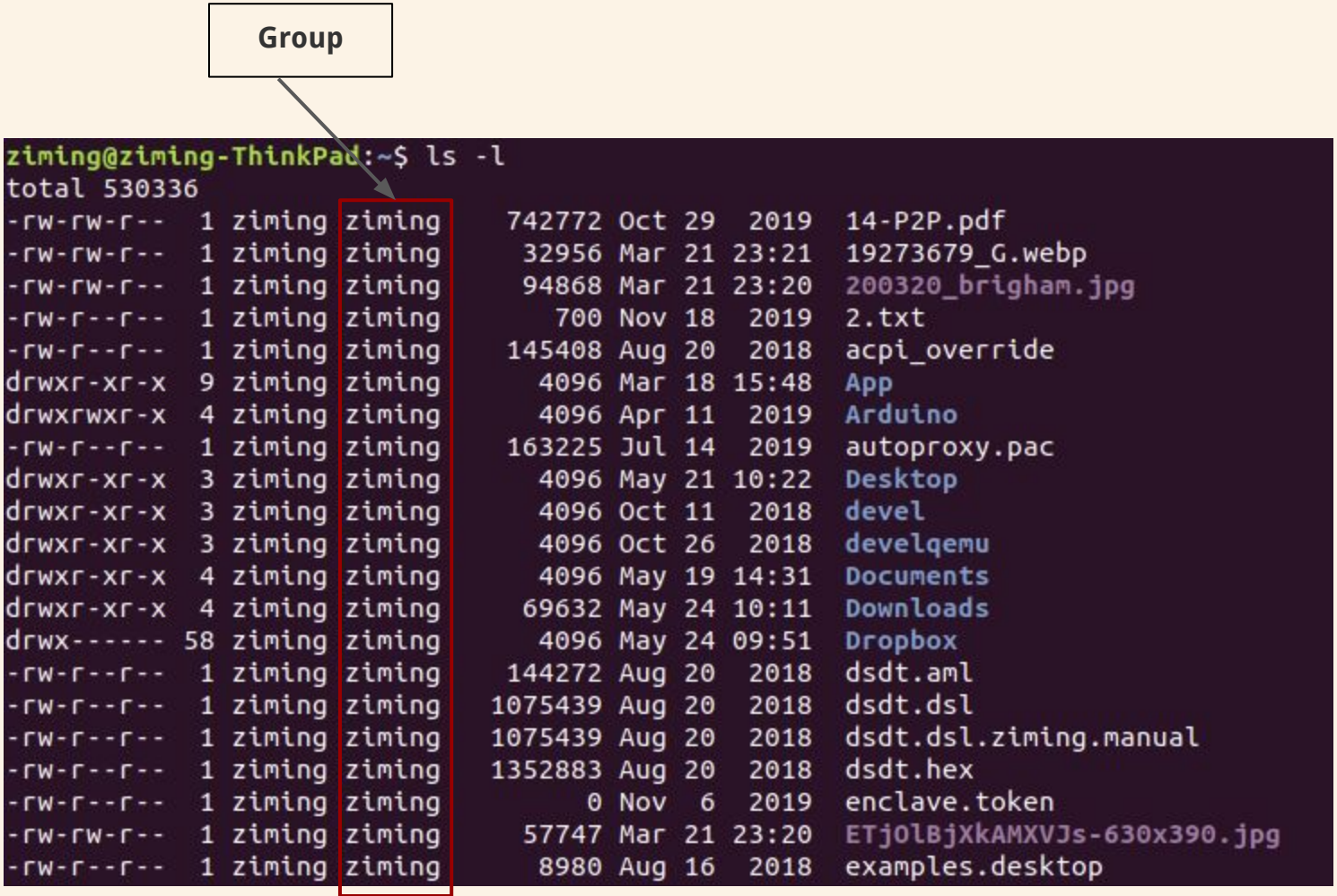
Permissions for owner, group, and others

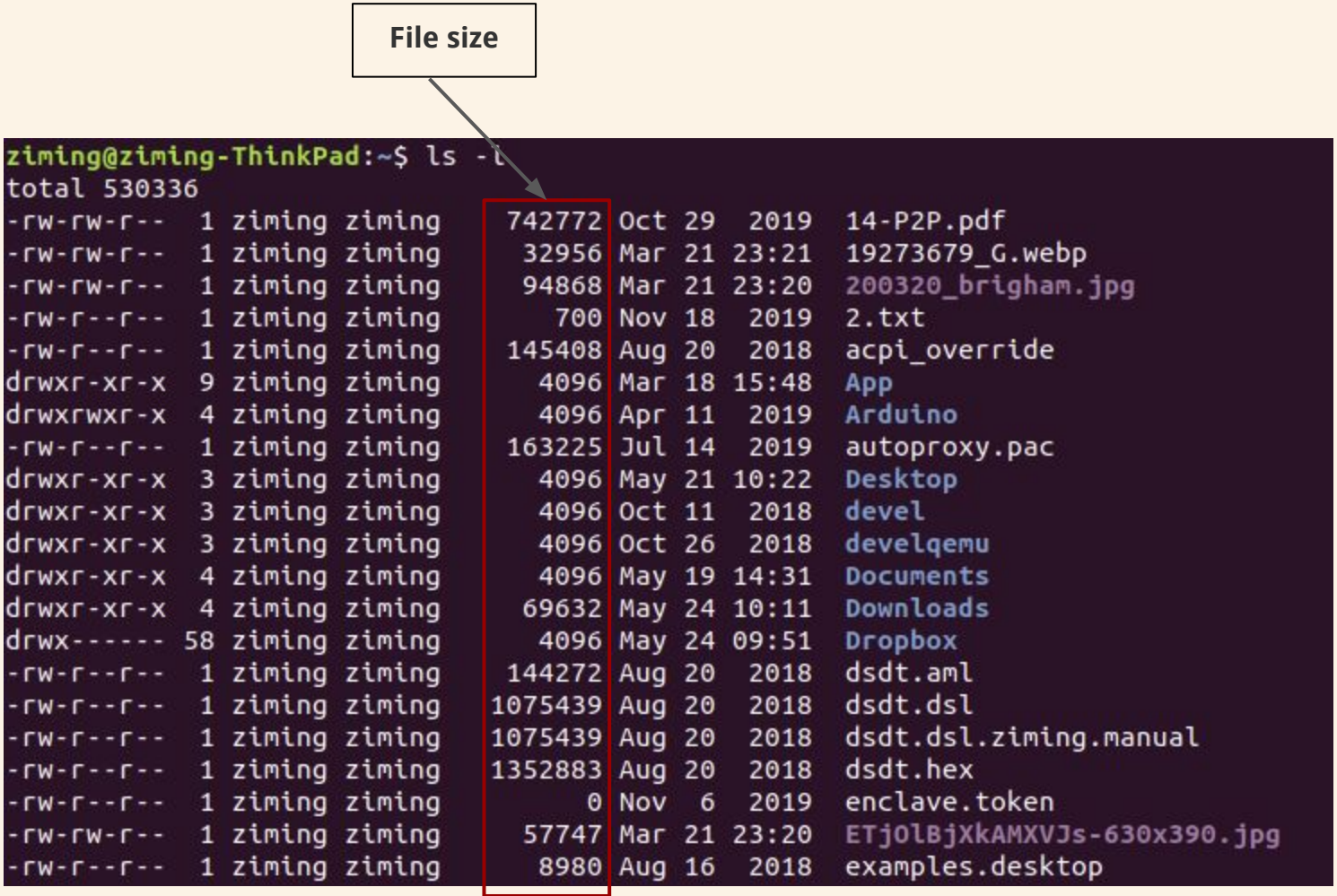


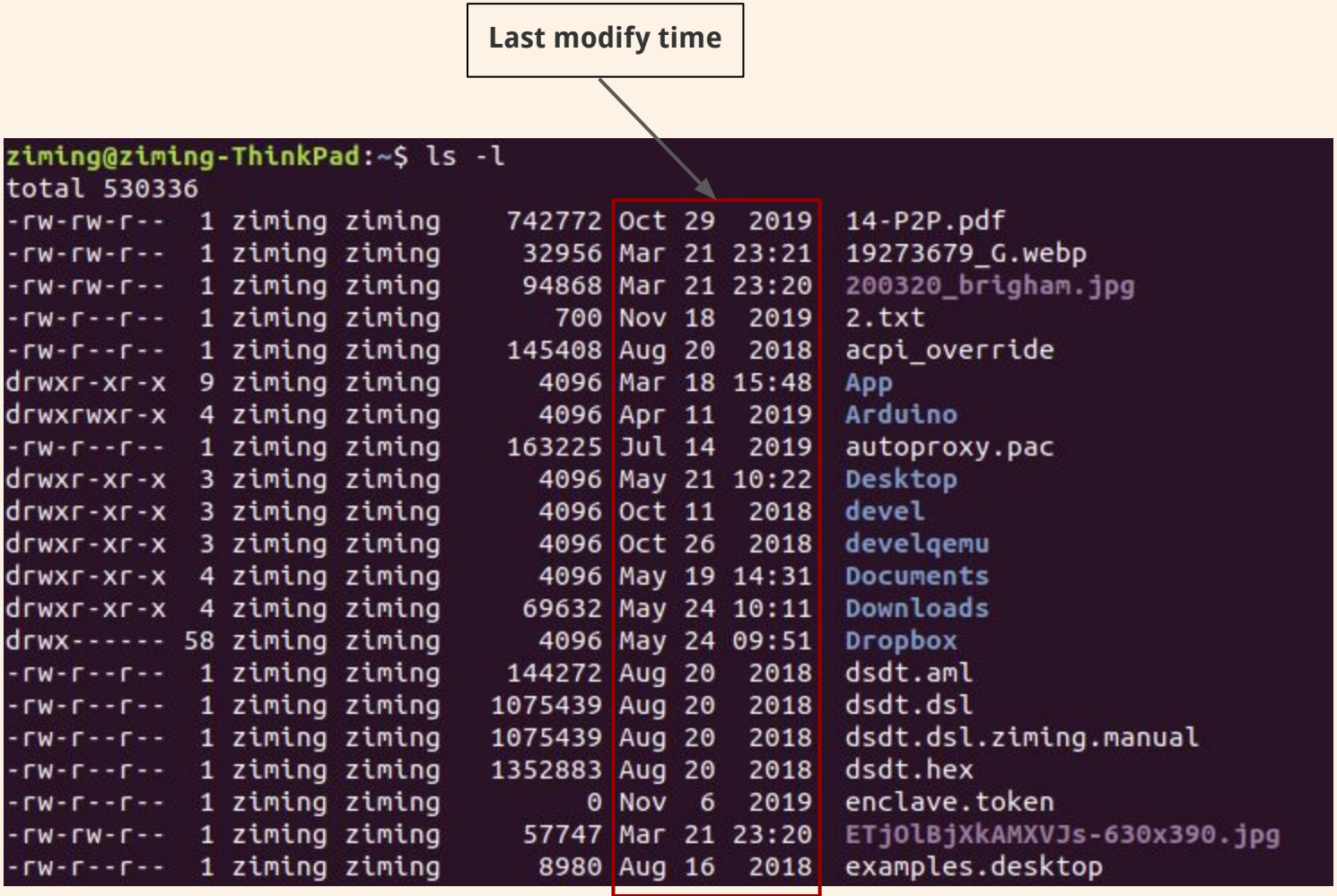

Link count
ziming@ziming-ThinkPad:~$ ls -l
total 530336
-rw-rw-r-- 1 ziming ziming 742772 Oct 29 2019 14-P2P.pdf
-rw-rw-r-- 1 ziming ziming 32956 Mar 21 23:21 19273679_G.webp
-rw-rw-r-- 1 ziming ziming 94868 Mar 21 23:20 200320_brigham.jpg
-rw-r--r-- 1 ziming ziming 700 Nov 18 2019 2.txt
-rw-r--r-- 1 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 ETjOlBjXkAMXVJs-630x390.jpg
-rw-r--r-- 1 ziming ziming 8980 Aug 16 2018 examples.desktop

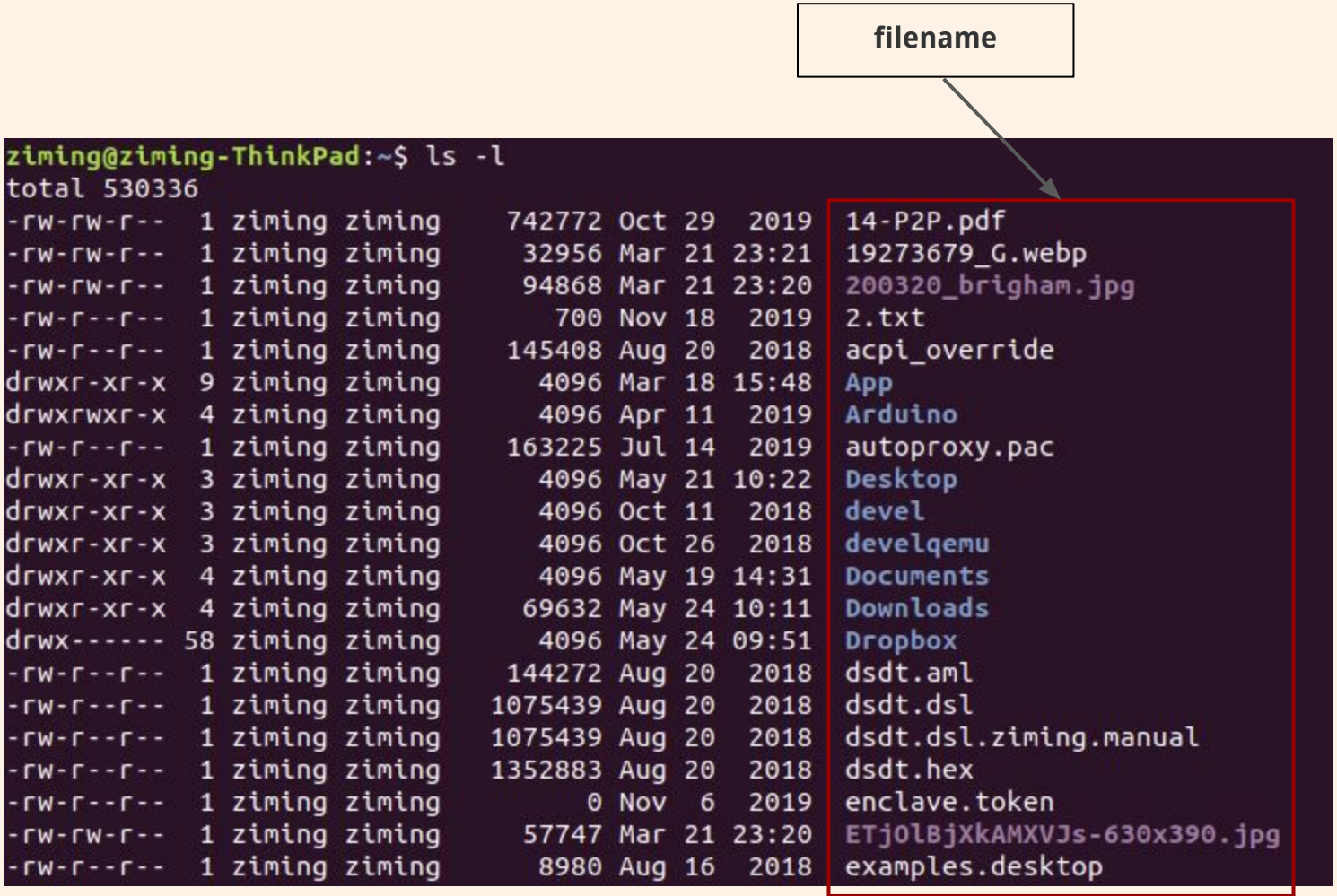
Owner: This field provide info about the creator of the file.




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 ETjOlBjXkAMXVJs-630x390.jpg
-rw-r--r-- 1 ziming ziming 8980 Aug 16 2018 examples.desktop

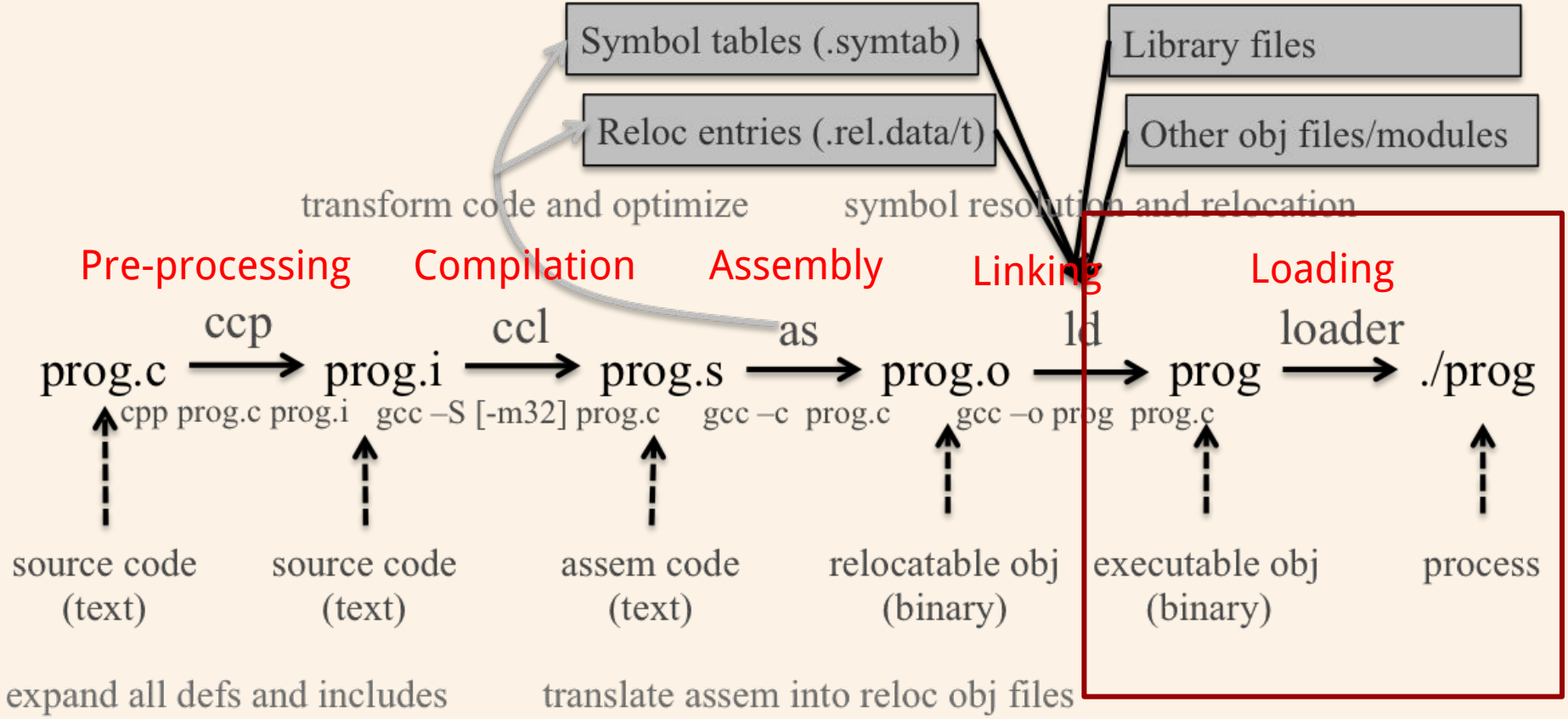

File size
ziming@ziming-ThinkPad:~$ ls -l
total 530336
-rw-rw-r-- 1 ziming ziming 742772 Oct 29 2019 14-P2P.pdf
-rw-rw-r-- 1 ziming ziming 32956 Mar 21 23:21 19273679_G.webp
-rw-rw-r-- 1 ziming ziming 94868 Mar 21 23:20 200320_brigham.jpg
-rw-r--r-- 1 ziming ziming 700 Nov 18 2019 2.txt
-rw-r--r-- 1 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 ETjOlBjXkAMXVJs-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 ETjOlBjXkAMXVJs-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 ETjOlBjXkAMXVJs-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 bzcat
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       6 Jul  4  2019 bzfgrep ->
-rwxr-xr-x  1 root root    3642 Jul  4  2019 bzgrep
-rwxr-xr-x  3 root root   34888 Jul  4  2019 bzip2
-rwxr-xr-x  1 root root   14328 Jul  4  2019 bzip2recov
lrwxrwxrwx  1 root root       6 Jul  4  2019 bzless ->
-rwxr-xr-x  1 root root    1297 Jul  4  2019 bzmore
-rwxr-xr-x  1 root root   35064 Jan 18  2018 cat
-rwxr-xr-x  1 root root   14328 Apr 21  2017 chacl
-rwxr-xr-x  1 root root   63672 Jan 18  2018 chgrp
-rwxr-xr-x  1 root root   59608 Jan 18  2018 chmod
-rwxr-xr-x  1 root root   67768 Jan 18  2018 chown
-rwxr-xr-x  1 root root   10312 Jan 22  2018 chvt
-rwxr-xr-x  1 root root  141528 Jan 18  2018 cp
-rwxr-xr-x  1 root root  157224 Nov  5  2019 cpio
-rwxr-xr-x  1 root root  121432 Jan 25  2018 dash
-rwxr-xr-x  1 root root  100568 Jan 18  2018 date
-rwxr-xr-x  1 root root   76000 Jan 18  2018 dd
-rwxr-xr-x  1 root root   84776 Jan 18  2018 df
-rwxr-xr-x  1 root root  133792 Jan 18  2018 dir
-rwxr-xr-x  1 root root   72000 Mar  5 12:23 dmesg
```

```
-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 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 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
```

Example: code/rdsecret

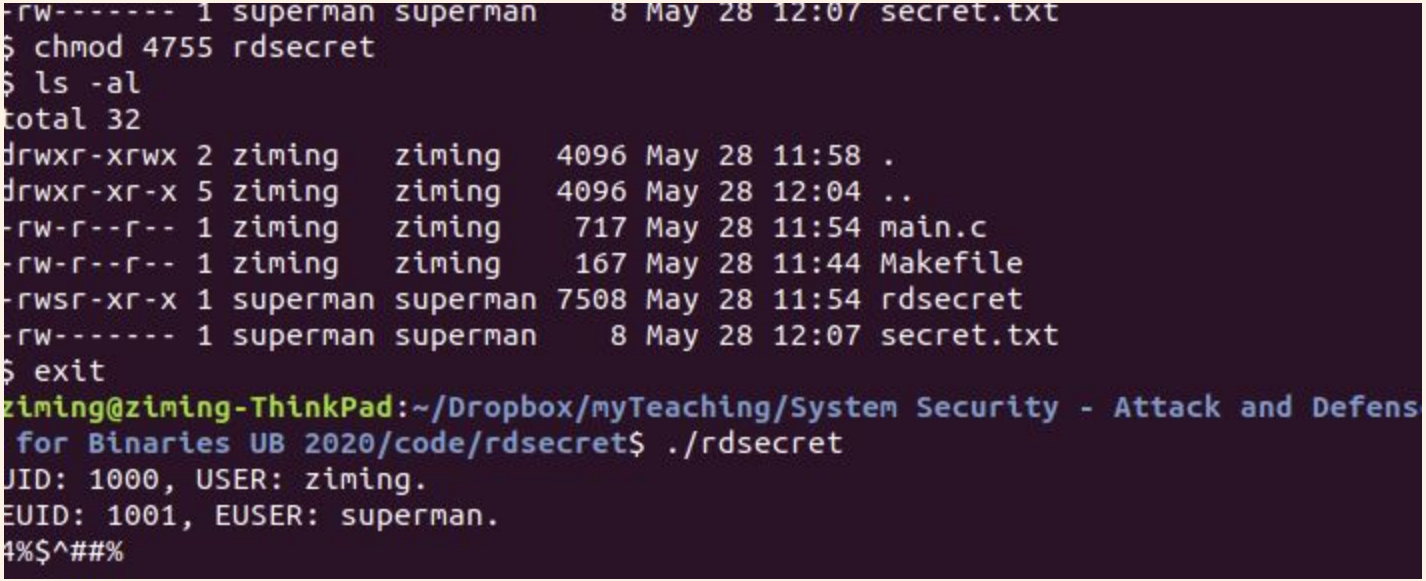
main.c

```
#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);
   }

   // open the file
   fp = fopen("secret.txt", "r");
   if (fp == NULL)
   {
           printf("Can't read the secret!\n");
           return(1);
   }

   fread(buffer, 99, 1, fp);
   printf("%s\n", buffer);
   fclose(fp);

   return(0);
}
```

Demo

```
-rw-r--r-- 1 ziming   ziming    167 May 28 11:44 Makefile
-rwxr-xr-x 1 ziming   ziming   7508 May 28 11:54 rdsecret
ziming@ziming-ThinkPad:~/Dropbox/myTeaching/System Security - Attack and Defense
ziming@ziming-ThinkPad:~/Dropbox/myTeaching/System Security - Attack and Defense
 for Binaries UB 2020/code/rdsecret$ su superman
Password:
$ echo 4%$^##% > secret.txt
$ chmod 600 secret.txt
$ ls -al
total 32
drwxr-xrwx 2 ziming   ziming   4096 May 28 11:58 .
drwxr-xr-x 5 ziming   ziming   4096 May 28 12:04 ..
-rw-r--r-- 1 ziming   ziming    717 May 28 11:54 main.c
-rw-r--r-- 1 ziming   ziming    167 May 28 11:44 Makefile
-rwxr-xr-x 1 ziming   ziming   7508 May 28 11:54 rdsecret
-rw------- 1 superman superman    8 May 28 12:07 secret.txt
$ cat secret.txt
4%$^##%
$ ./rdsecret
UID: 1001, USER: superman.
EUID: 1001, EUSER: superman.
4%$^##%
```


ziming@ziming-ThinkPad:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/rdsecret$ ls -al
total 32
drwxr-xrwx 2 ziming ziming 4096 May 28 11:58 .
drwxr-xr-x 5 ziming ziming 4096 May 28 12:04 ..
-rw-r--r-- 1 ziming ziming 717 May 28 11:54 main.c
-rw-r--r-- 1 ziming ziming 167 May 28 11:44 Makefile
-rwxr-xr-x 1 superman superman 7508 May 28 11:54 rdsecret
-rw------- 1 superman superman 8 May 28 12:07 secret.txt
ziming@ziming-ThinkPad:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/rdsecret$ ./rdsecret
UID: 1000, USER: ziming.
EUID: 1000, EUSER: ziming.
Can't read the secret!
ziming@ziming-ThinkPad:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/rdsecret$ cat secret.txt
cat: secret.txt: Permission denied
ziming@ziming-ThinkPad:~/Dropbox/myTeaching/System Security - Attack and Defense for Binaries UB 2020/code/rdsecret$


-rw------- 1 superman superman 8 May 28 12:07 secret.txt
$ chmod 4755 rdsecret
$ ls -al
total 32
drwxr-xrwx 2 ziming ziming 4096 May 28 11:58 .
drwxr-xr-x 5 ziming ziming 4096 May 28 12:04 ..
-rw-r--r-- 1 ziming ziming 717 May 28 11:54 main.c
-rw-r--r-- 1 ziming ziming 167 May 28 11:44 Makefile
-rwsr-xr-x 1 superman superman 7508 May 28 11:54 rdsecret
-rw------- 1 superman superman 8 May 28 12:07 secret.txt
$ exit
ziming@ziming-ThinkPad:~/Dropbox/myTeaching/System Security - Attack and Defens
for Binaries UB 2020/code/rdsecret$ ./rdsecret
UID: 1000, USER: ziming.
EUID: 1001, EUSER: superman.
4%$^##%

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

```
ziming@ziming-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 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]                   NULL             0000000000000000  00000000
       0000000000000000  0000000000000000           0     0     0
  [ 1] .interp           PROGBITS         0000000000000318  00000318
       000000000000001c  0000000000000000   A       0     0     1
  [ 2] .note.gnu.propert NOTE             0000000000000338  00000338
       0000000000000020  0000000000000000   A       0     0     8
  [ 3] .note.gnu.build-i NOTE             0000000000000358  00000358
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .note.ABI-tag     NOTE             000000000000037c  0000037c
       0000000000000020  0000000000000000   A       0     0     4
  [ 5] .gnu.hash         GNU_HASH         00000000000003a0  000003a0
       00000000000000e4  0000000000000000   A       6     0     8
  [ 6] .dynsym           DYNSYM           0000000000000488  00000488
       0000000000000d08  0000000000000018   A       7     1     8
  [ 7] .dynstr           STRTAB           0000000000001190  00001190
       000000000000064c  0000000000000000   A       0     0     1
  [ 8] .gnu.version      VERSYM           00000000000017dc  000017dc
       0000000000000116  0000000000000002   A       6     0     2
  [ 9] .gnu.version_r    VERNEED          00000000000018f8  000018f8
       0000000000000070  0000000000000000   A       7     1     8
  [10] .rela.dyn         RELA             0000000000001968  00001968
       0000000000001350  0000000000000018   A       6     0     8
  [11] .rela.plt         RELA             0000000000002cb8  00002cb8
       00000000000009f0  0000000000000018  AI       6    25     8
  [12] .init             PROGBITS         0000000000004000  00004000
       000000000000001b  0000000000000000  AX       0     0     4
  [13] .plt              PROGBITS         0000000000004020  00004020
       00000000000006b0  0000000000000010  AX       0     0     16
```

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

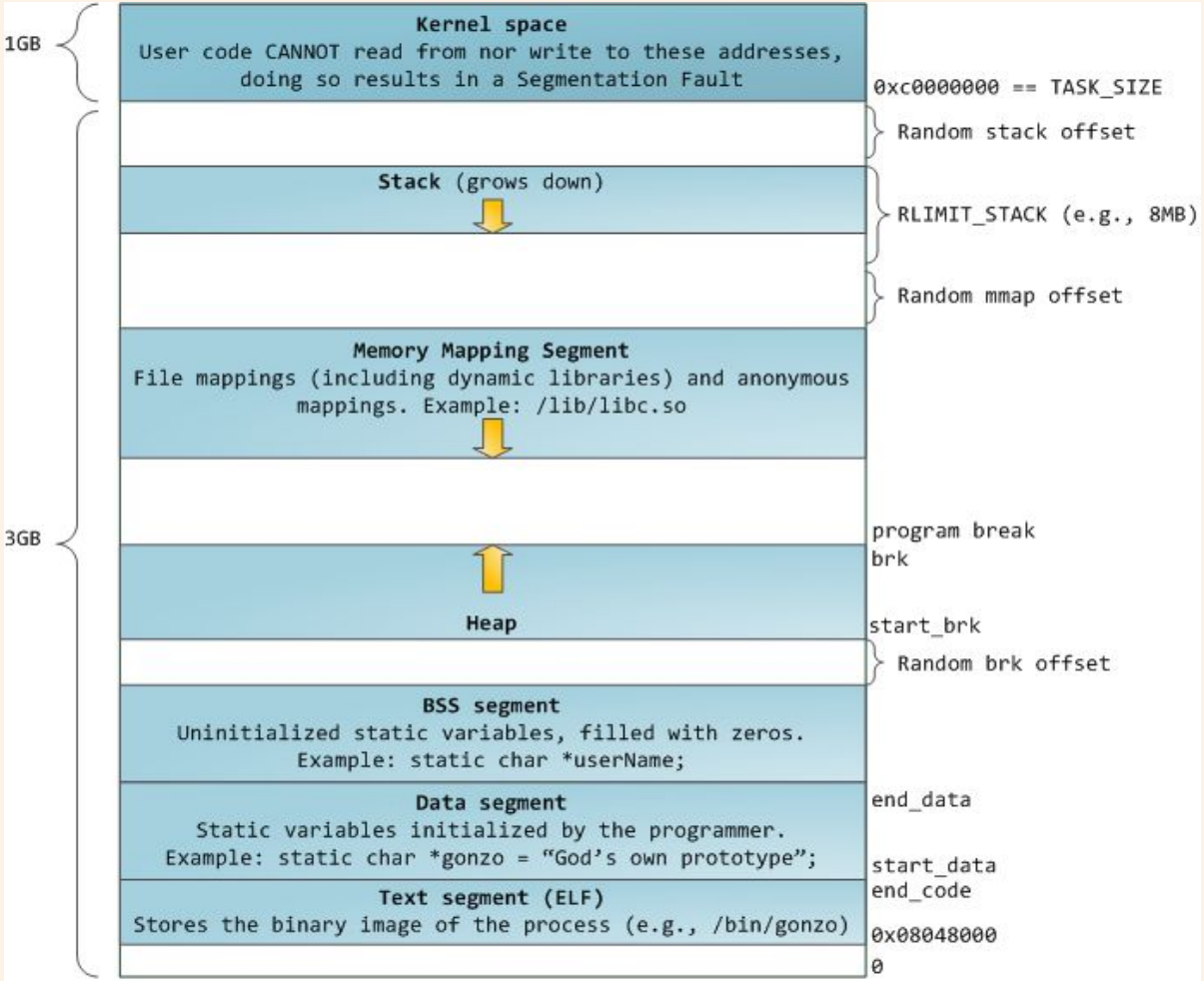
Memory Map of Linux Process (32 bit)

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/post/anatomy-of-a-program-in-memory/

NULL Pointer in C/C++

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

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

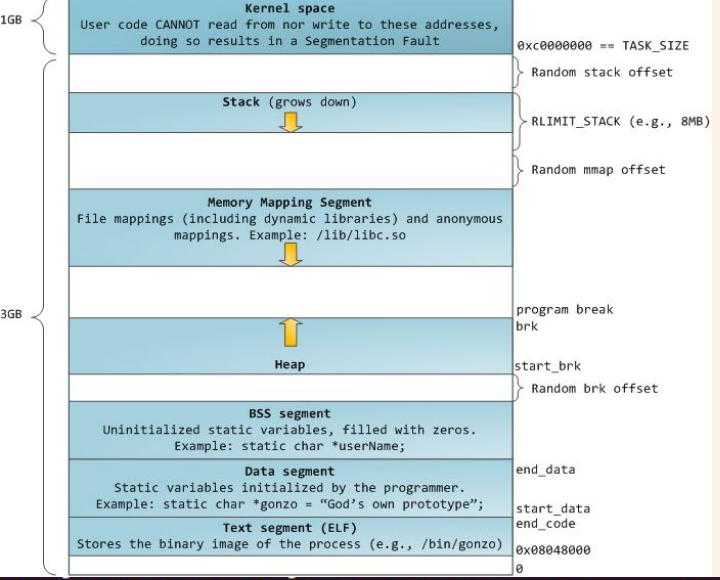
```
#define NULL ((char *)0)
#define NULL 0

//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 libc-2.27.so
f7f4b000 rw-p 001d7000 103:02  2883591    4   4   4          4         4        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
f7f75000 rw-p 00000000  00:00        0    8   8   8          8         8        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         0        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 ld-2.27.so
f7fa3000 rw-p 00026000 103:02  2883587    4   4   4          4         4        0              0              0              0    0       0      0 ld-2.27.so
ffef3000 rw-p 00000000  00:00        0  132  12  12         12        12        0              0              0              0    0       0      0 [stack]
                               ==== === === ========= ======== ======= ============= ============== =============== ==== ======= ======
                               2372 988 988       988       60       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]
ffffffffff600000 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 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. 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, 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

```
ziming@ziming-ThinkPad:~$ strace ls
execve("/bin/ls", ["ls"], 0x7ffc1c069370 /* 56 vars */) = 0
brk(NULL)                               = 0x55c29ecbc000
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=153244, ...}) = 0
mmap(NULL, 153244, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f9ce52bd000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libselinux.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\20b\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=154832, ...}) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f9ce52bb000
mmap(NULL, 2259152, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f9ce4e94000
mprotect(0x7f9ce4eb9000, 2093056, PROT_NONE) = 0
mmap(0x7f9ce50b8000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x24000) = 0x7f9ce50b8000
mmap(0x7f9ce50ba000, 6352, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f9ce50ba000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\260\34\2\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0755, st_size=2030544, ...}) = 0
mmap(NULL, 4131552, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f9ce4aa3000
mprotect(0x7f9ce4c8a000, 2097152, PROT_NONE) = 0
mmap(0x7f9ce4e8a000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1e7000) = 0x7f9ce4e8a000
mmap(0x7f9ce4e90000, 15072, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f9ce4e90000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libpcre.so.3", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0 \25\0\0\0\0\0\0"..., 832) = 832
fstat(3, {st_mode=S_IFREG|0644, st_size=464824, ...}) = 0
mmap(NULL, 2560264, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f9ce4831000
mprotect(0x7f9ce48a1000, 2097152, PROT_NONE) = 0
mmap(0x7f9ce4aa1000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x70000) = 0x7f9ce4aa1000
close(3)                                = 0
access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT (No such file or directory)
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libdl.so.2", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\16\0\0\0\0\0\0"..., 832) = 832
```

Making a System Call in x86 Assembly

On x86/x86-64, most system calls rely on the software interrupt (the **int 0x80** instruction).

A software interrupt is caused either by an exceptional condition in the processor itself, or a special 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

%eax	Name	Source	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	kernel/exit.c	int	-	-	-	-
2	sys_fork	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
3	sys_read	fs/read_write.c	unsigned int	char *	size_t	-	-
4	sys_write	fs/read_write.c	unsigned int	const char *	size_t	-	-
5	sys_open	fs/open.c	const char *	int	int	-	-
6	sys_close	fs/open.c	unsigned int	-	-	-	-
7	sys_waitpid	kernel/exit.c	pid_t	unsigned int *	int	-	-
8	sys_creat	fs/open.c	const char *	int	-	-	-
9	sys_link	fs/namei.c	const char *	const char *	-	-	-
10	sys_unlink	fs/namei.c	const char *	-	-	-	-
11	sys_execve	arch/i386/kernel/process.c	struct pt_regs	-	-	-	-
12	sys_chdir	fs/open.c	const char *	-	-	-	-
13	sys_time	kernel/time.c	int *	-	-	-	-
14	sys_mknod	fs/namei.c	const char *	int	dev_t	-	-
15	sys_chmod	fs/open.c	const char *	mode_t	-	-	-
16	sys_lchown	fs/open.c	const char *	uid_t	gid_t	-	-
18	sys_stat	fs/stat.c	char *	struct __old_kernel_stat *	-	-	-
19	sys_lseek	fs/read_write.c	unsigned int	off_t	unsigned int	-	-
20	sys_getpid	kernel/sched.c	-	-	-	-	-
21	sys_mount	fs/super.c	char *	char *	char *	-	-
22	sys_oldumount	fs/super.c	char *	-	-	-	-

https://www.informatik.htw-dresden.de/~beck/ASM/syscall_list.html

Making a System Call in x86 Assembly

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

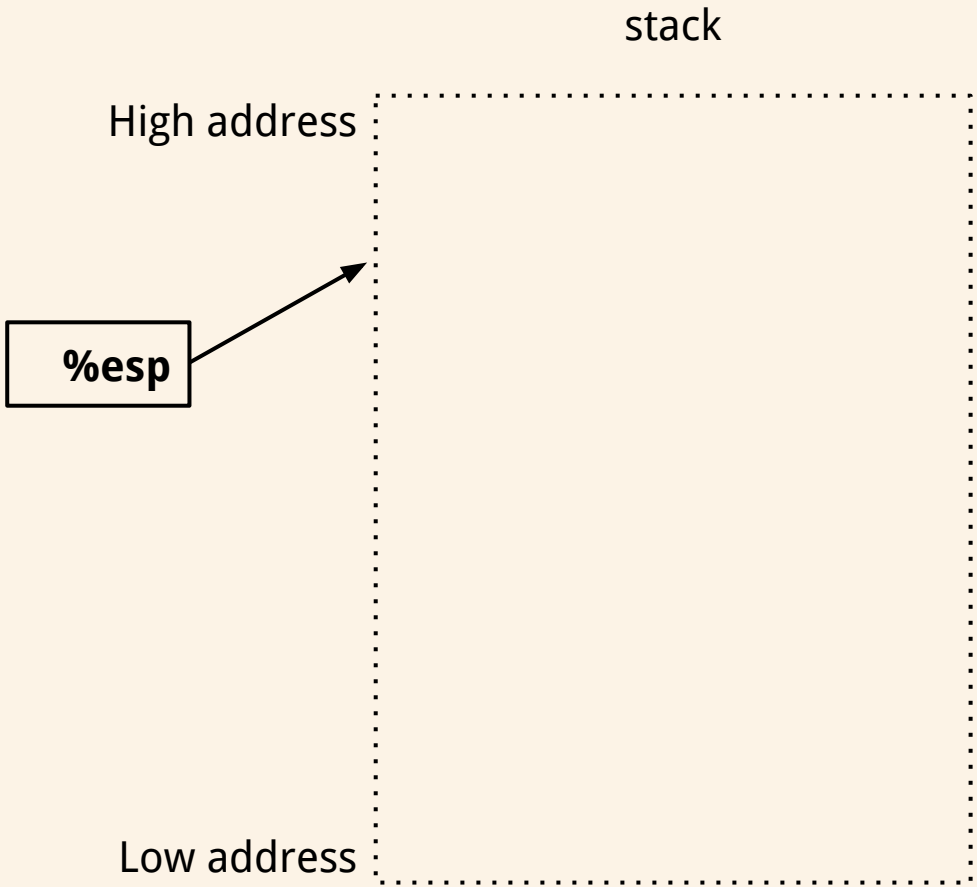
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of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

Source: www.LookupTables.com

http://shell-storm.org/shellcode/files/shellcode-827.php

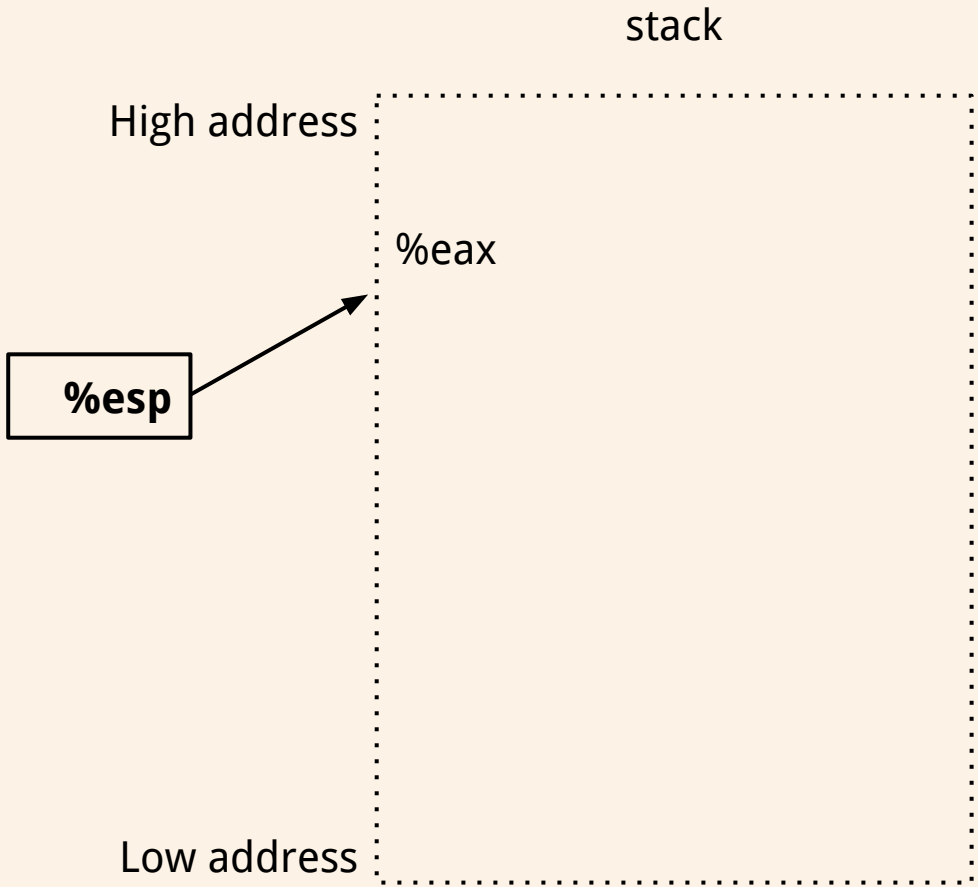
Making a System Call in x86 Assembly

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



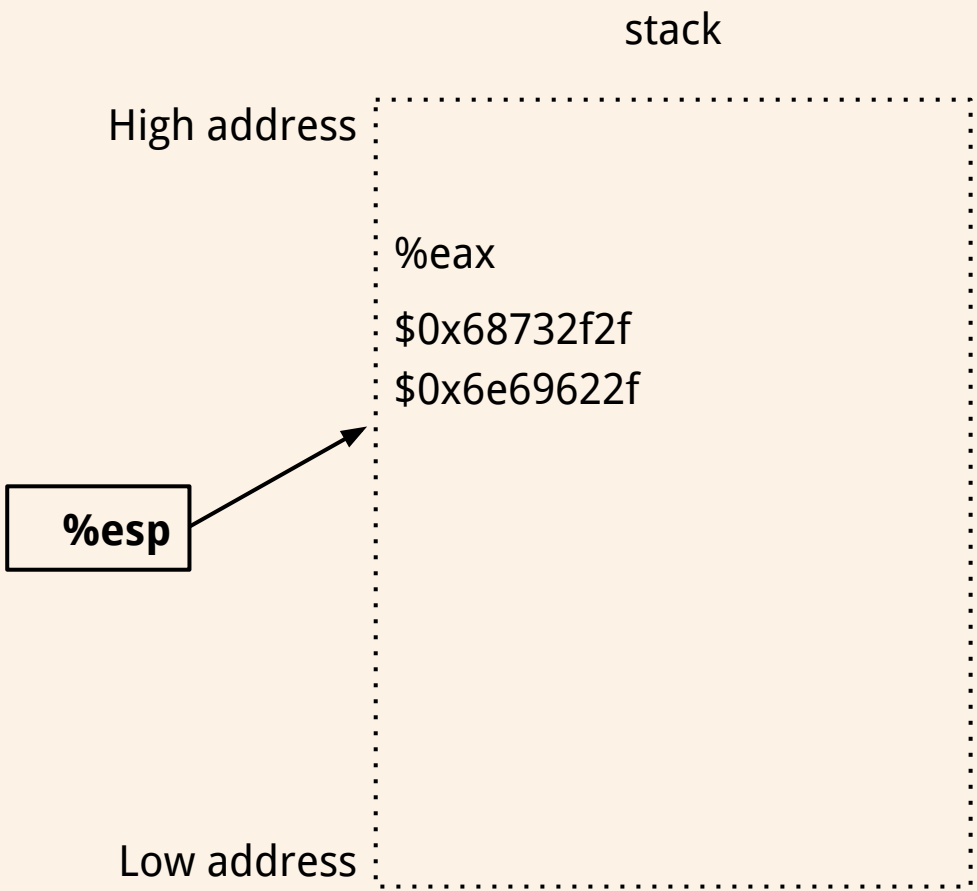
Making a System Call in x86 Assembly

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



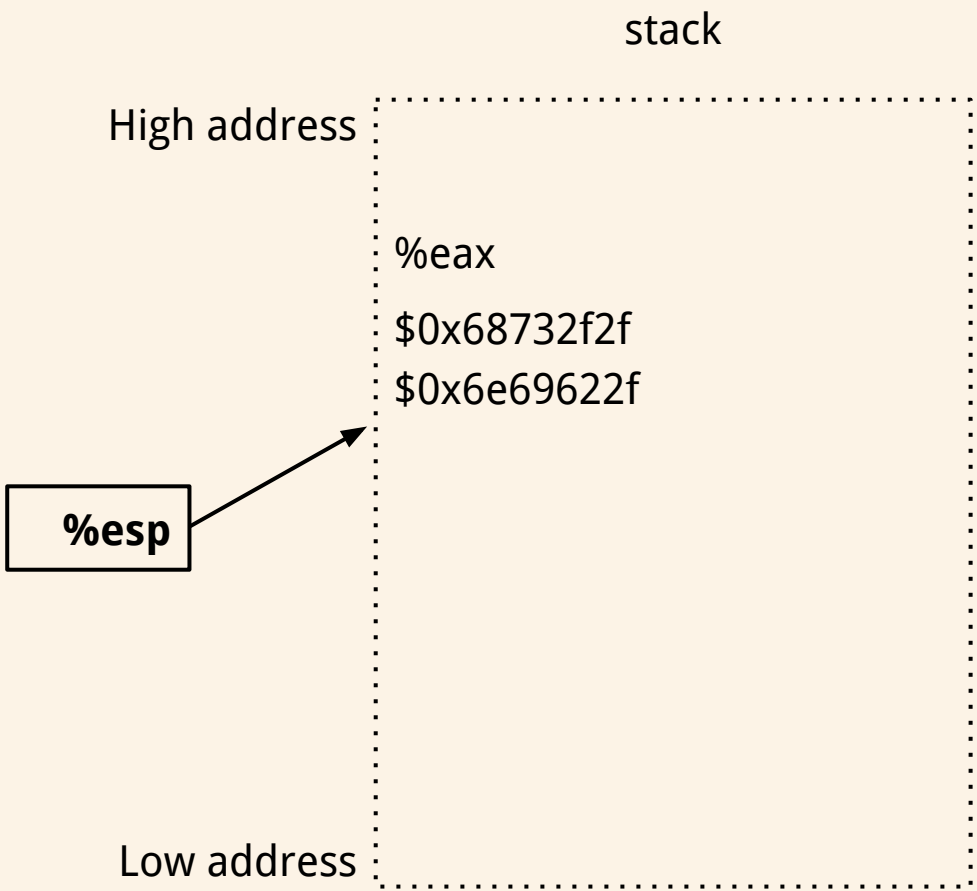
Making a System Call in x86 Assembly

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

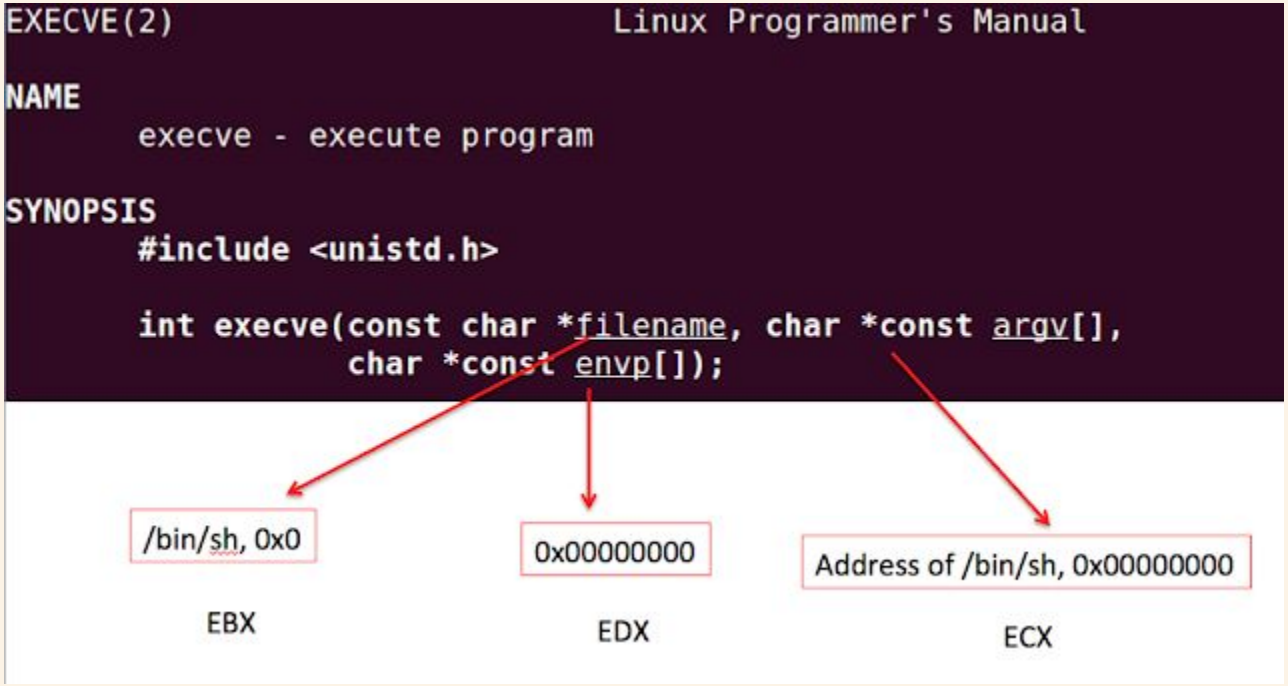


Making a System Call in x86 Assembly

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



Making a System Call in x86 Assembly



execve(“/bin/sh”, address of string “/bin/sh”, 0)

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.

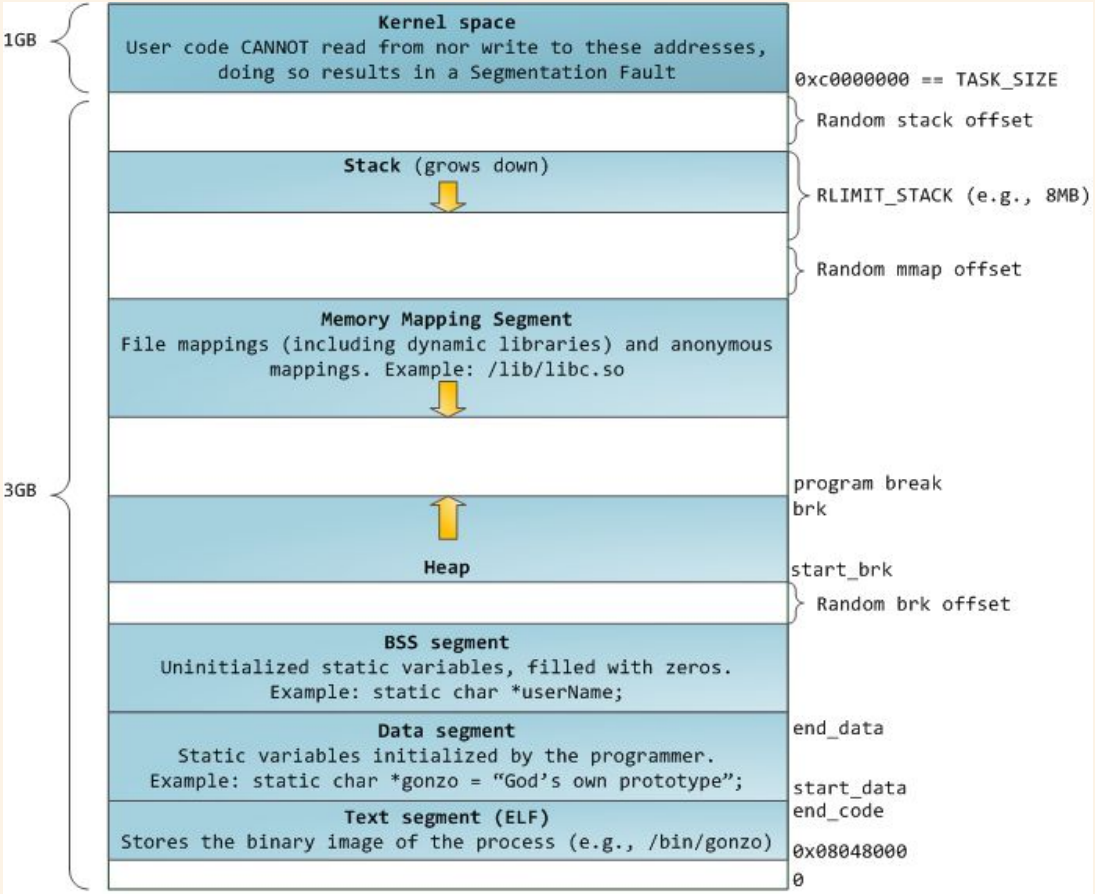
Common Environment Variables

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: Reverse Engineering Tools

Tools for Week-1

file
readelf
strings
nm
objdump
IDA Pro
ghidra

GDB Cheat Sheet

Start gdb using:
gdb <binary>
Pass initial commands for gdb through a file
gdb <binary> –x <initfile>

To start running the program
r <argv>
Use python output as stdin in GDB:
r <<< $(python -c "print '\x12\x34'*5")

Set breakpoint at address:
b *0x80000000
b main
Disassemble 10 instructions from an address:
x/10i 0x80000000

GDB Cheat Sheet

To put breakpoints (stop execution on a certain line)
b <function name>
b *<instruction address>
b <filename:line number>
b <line number>

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

Shell Cheat Sheet

Run a program and use another program’s output as a parameter
program $(python -c "print '\x12\x34'*5")

In-class Exercises

1. Homework-1

Dues

1. Homework-1