CS 4910: Intro to Computer Security

Software Security I: Background Knowledge

Instructor: Xi Tan

Updates

- Lab 3:
  - Buffer-Overflow Attack (Set-UID Version)
  - Deadline: 5/05
- Homework 4
  - Deadline: 04/23

- Research Paper:
  - Deadline: ~~04/14~~ 4/21

Review

- Previous topics
  - CIA
  - Authentication
  - Access control
  - Database security
  - Malicious software
  - Network security

Software Security

Agenda

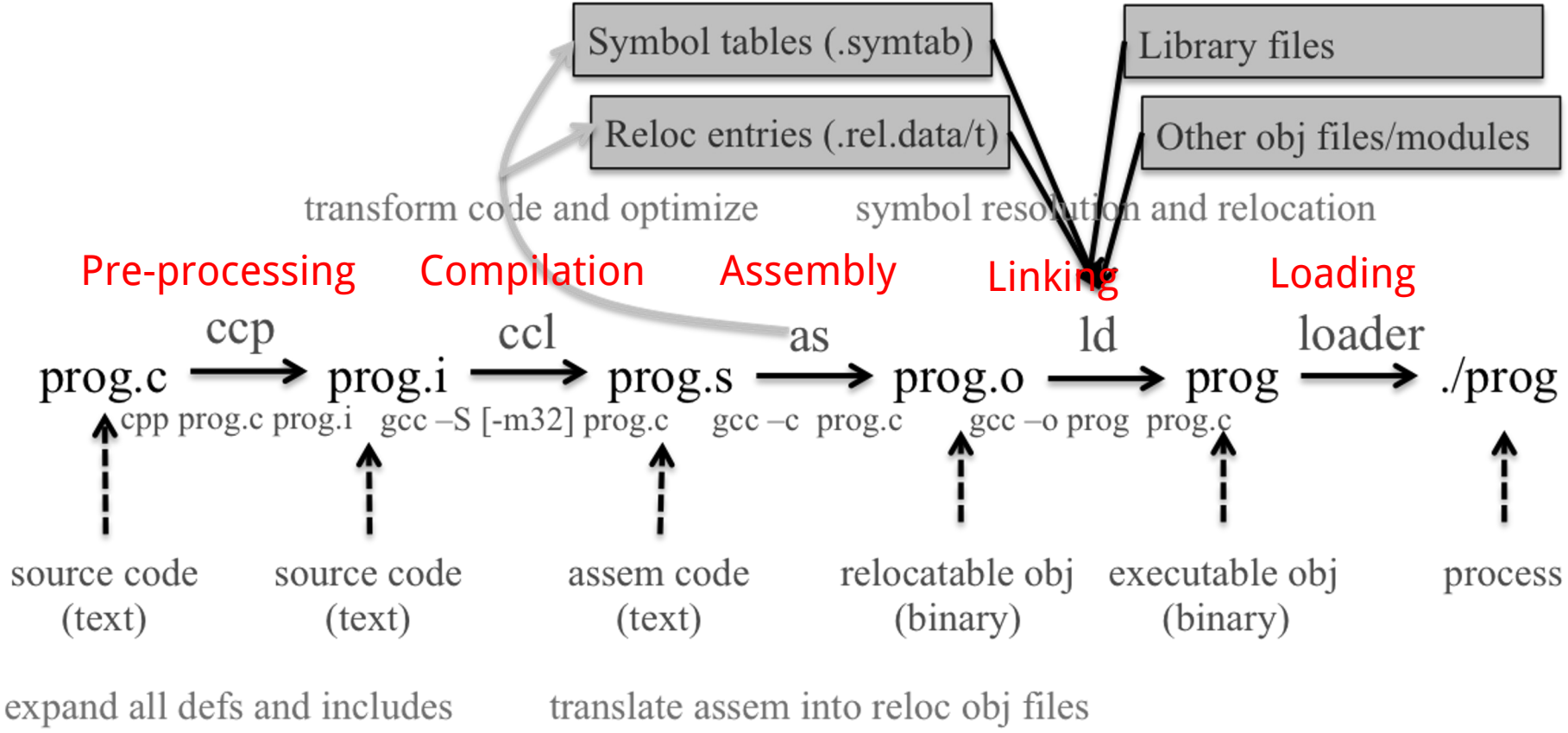
- Background
- Buffer-overflow attack
- Buffer-overflow defense

Today

- Background knowledge
  - Compiler, linker, loader
  - x86 and x86-64 architectures and ISA
  - Set-UID programs

Compiler, linker, and loader

From a C program to a process



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.

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
```

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.

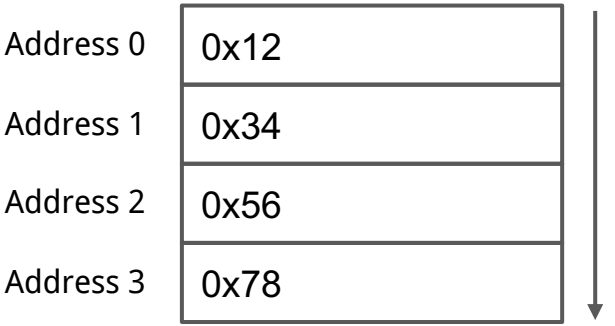
Endianness

- 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

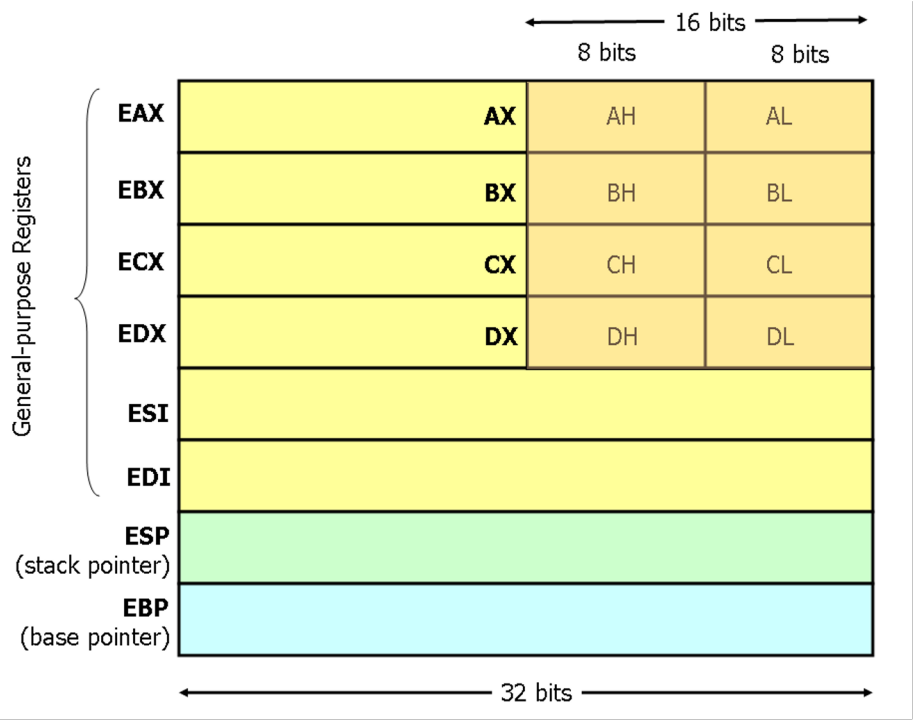
Value: 0x12345678

Base Registers

There are

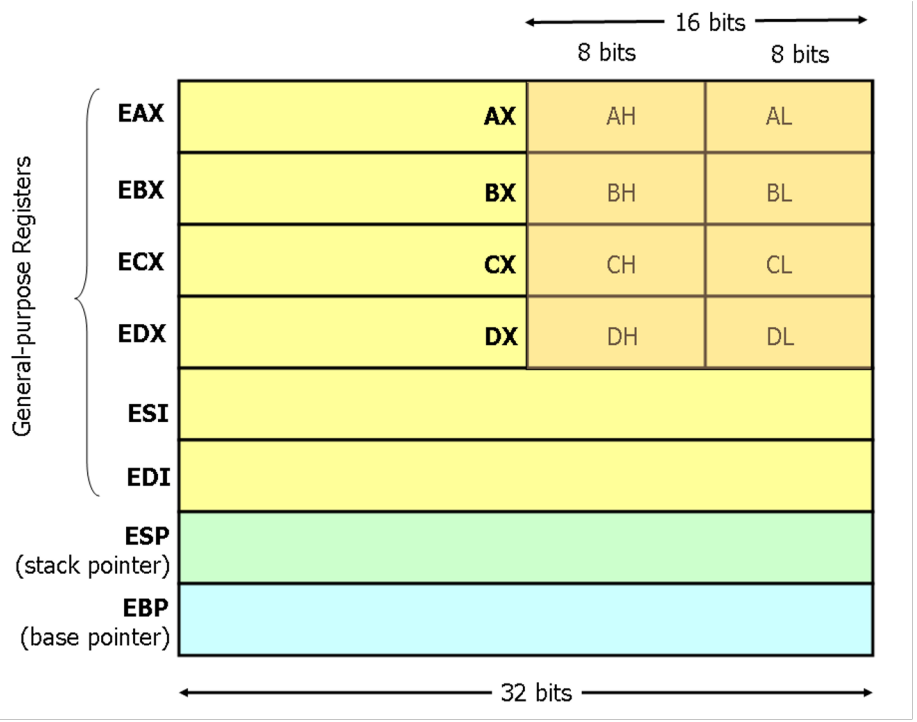
- Eight 32-bit “general-purpose” registers,
- One 32-bit EFLAGS register,
- One 32-bit instruction pointer register (eip), and
- Other special-purpose registers.

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.

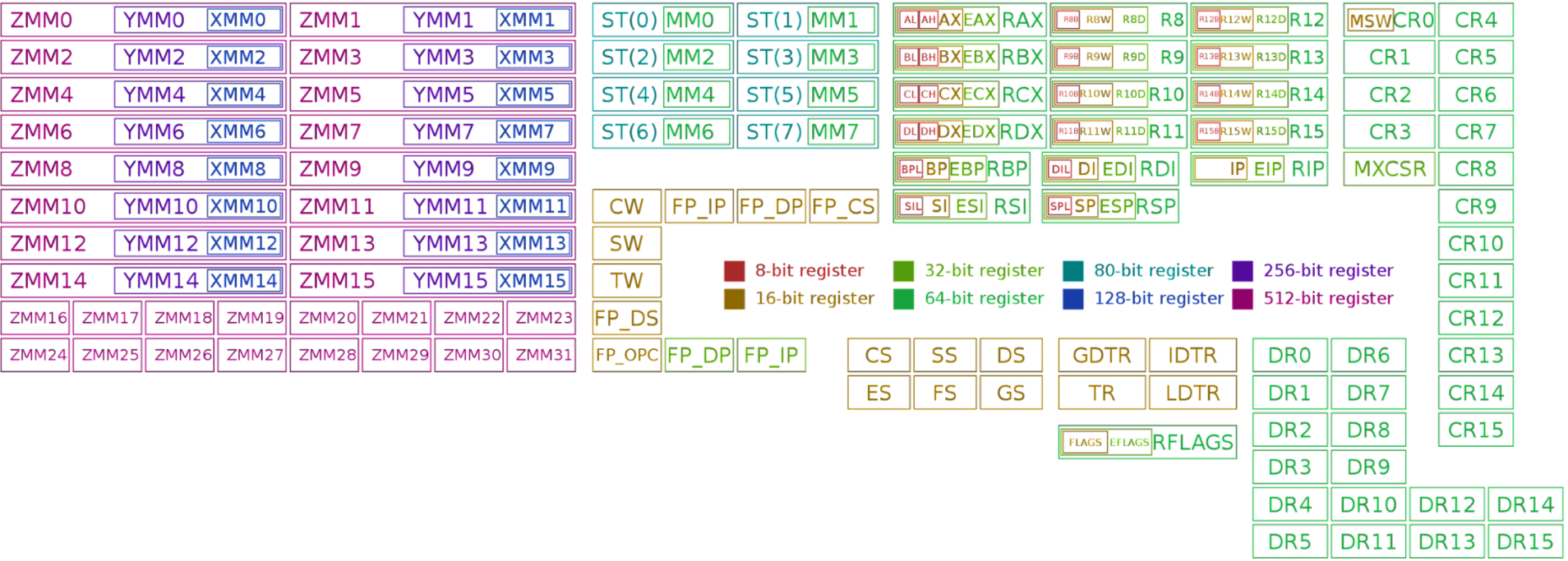
We will be interested in, at most, the bits

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 (program counter). 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>

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)
(read as MOVE FROM x to y)

mov [eax], ebx	Load 4 bytes from the EBX to the memory address 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

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.

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 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> # direct jump
jmp <reg32> # indirect jump

Example
jmp begin — Jump to the instruction labeled begin.

Control Flow Instructions

jcondition — Conditional jump

Syntax

je (jump when equal)

jne (jump when not equal)

jz (jump when last result was zero)

jg (jump when greater than)

jge (jump when greater than or equal to)

jl (jump when less than)

jle (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

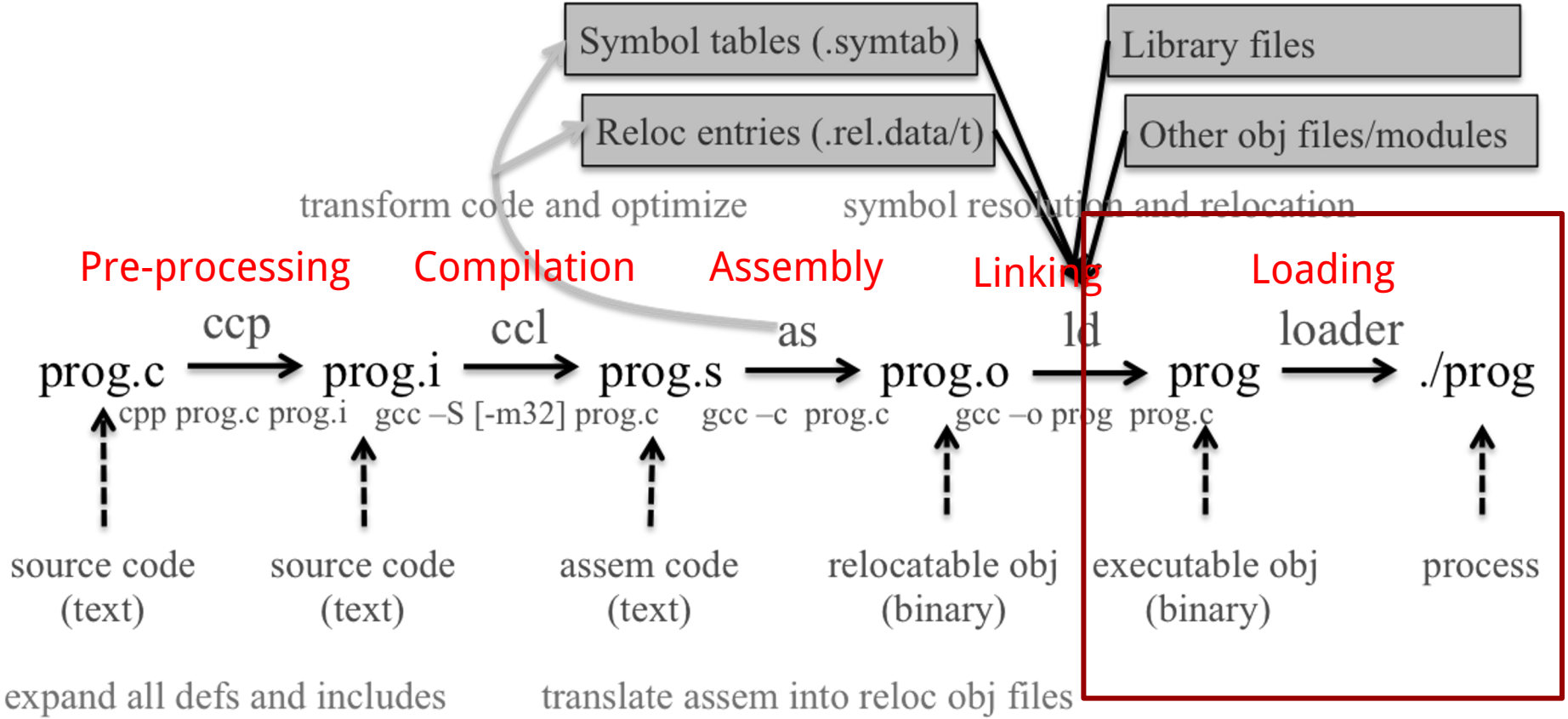
Stack Instructions

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

Equivalent to:

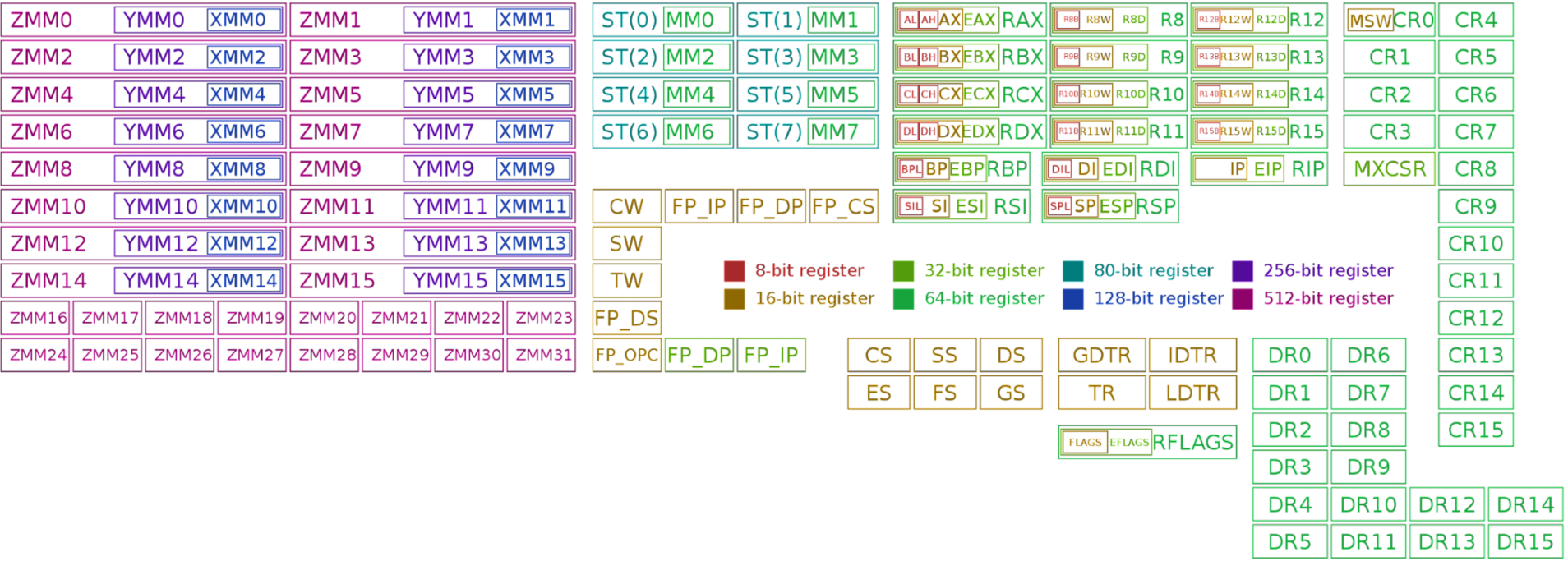
```
mov esp, ebp
pop ebp
```

From a C program to a process



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

```
000012c0 <ladd>:
   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

```
0000000000001220 <ladd>:
   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
```

Set-UID Programs

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.

```
-rwxr-xr-x  1 root root      170416 Nov 23  2022  ssh-add
-rwxr-sr-x  1 root _ssh      293304 Nov 23  2022  ssh-agent
-rwxr-xr-x  1 root root        1455 Nov 14  2022  ssh-argv0
-rwxr-xr-x  1 root root       12676 Feb 23  2022  ssh-copy-id
-rwxr-xr-x  1 root root      448960 Nov 23  2022  ssh-keygen
-rwxr-xr-x  1 root root      195008 Nov 23  2022  ssh-keyscan
-rwxr-xr-x  1 root root       80392 Feb  7  2022  stat
lrwxrwxrwx  1 root root           7 Feb  4  2022  static-sh -> busybox
-rwxr-xr-x  1 root root       43520 Feb  7  2022  stdbuf
-rwxr-xr-x  1 root root     1972848 Feb 16  2022  strace
-rwxr-xr-x  1 root root        1821 Feb 16  2021  strace-log-merge
-rwxr-xr-x  1 root root        7941 Oct  4  2022  streamzip
lrwxrwxrwx  1 root root          24 Nov  2  2022  strings -> x86_64-linux-gnu-strings
lrwxrwxrwx  1 root root          22 Nov  2  2022  strip -> x86_64-linux-gnu-strip
-rwxr-xr-x  1 root root       76288 Feb  7  2022  stty
-rwsr-xr-x  1 root root       55672 Feb 20  2022  su
-rwsr-xr-x  1 root root      232416 Apr  3  2023  sudo
lrwxrwxrwx  1 root root           4 Apr  3  2023  sudoedit -> sudo
-rwxr-xr-x  1 root root       89744 Apr  3  2023  sudoreplay
-rwxr-xr-x  1 root root       35232 Feb  7  2022  sum
-rwxr-xr-x  1 root root       35232 Feb  7  2022  sync
-rwxr-xr-x  1 root root     1119856 Mar 20  2023  systemctl
lrwxrwxrwx  1 root root          20 Mar 20  2023  systemd -> /lib/systemd/systemd
-rwxr-xr-x  1 root root     1809160 Mar 20  2023  systemd-analyze
-rwxr-xr-x  1 root root       18928 Mar 20  2023  systemd-ask-password
-rwxr-xr-x  1 root root       18816 Mar 20  2023  systemd-cat
-rwxr-xr-x  1 root root       23016 Mar 20  2023  systemd-cgls
-rwxr-xr-x  1 root root       39312 Mar 20  2023  systemd-cgtop
```

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);
}
```

ELF Binary Files Stack

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).

Commnad `file`

```
t@tancy-win ~ file /bin/ls
/bin/ls: ELF 64-bit LSB pie executable, x86-64, version 1 (SYSV), dynamically linked, interp
reter /lib64/ld-linux-x86-64.so.2, BuildID[sha1]=897f49cafa98c11d63e619e7e40352f855249c13, f
or GNU/Linux 3.2.0, stripped
```

file /bin/ls

```
t@tancy-win ~ 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 (Position-Independent Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x6ab0
  Start of program headers:          64 (bytes into file)
  Start of section headers:          136224 (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:         31
  Section header string table index: 30

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.pr[...] NOTE             0000000000000338  00000338
       0000000000000030  0000000000000000   A       0     0     8
  [ 3] .note.gnu.bu[...] NOTE             0000000000000368  00000368
       0000000000000024  0000000000000000   A       0     0     4
  [ 4] .note.ABI-tag     NOTE             000000000000038c  0000038c
       0000000000000020  0000000000000000   A       0     0     4
  [ 5] .gnu.hash         GNU_HASH         00000000000003b0  000003b0
       000000000000004c  0000000000000000   A       6     0     8
  [ 6] .dynsym           DYNSYM           0000000000000400  00000400
       0000000000000b88  0000000000000018   A       7     1     8
  [ 7] .dynstr           STRTAB           0000000000000f88  00000f88
       00000000000005a6  0000000000000000   A       0     0     1
  [ 8] .gnu.version      VERSYM           000000000000152e  0000152e
       00000000000000f6  0000000000000002   A       6     0     2
  [ 9] .gnu.version_r    VERNEED          0000000000001628  00001628
       00000000000000c0  0000000000000000   A       7     2     8
  [10] .rela.dyn         RELA             00000000000016e8  000016e8
       00000000000013e0  0000000000000018   A       6     0     8
  [11] .rela.plt         RELA             0000000000002ac8  00002ac8
       0000000000000960  0000000000000018  AI       6    25     8
  [12] .init             PROGBITS         0000000000004000  00004000
       0000000000000025  0000000000000000  AX       0     0     4
  [13] .plt              PROGBITS         0000000000004030  00004030
       0000000000000650  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.

Memory Map of a Linux Process

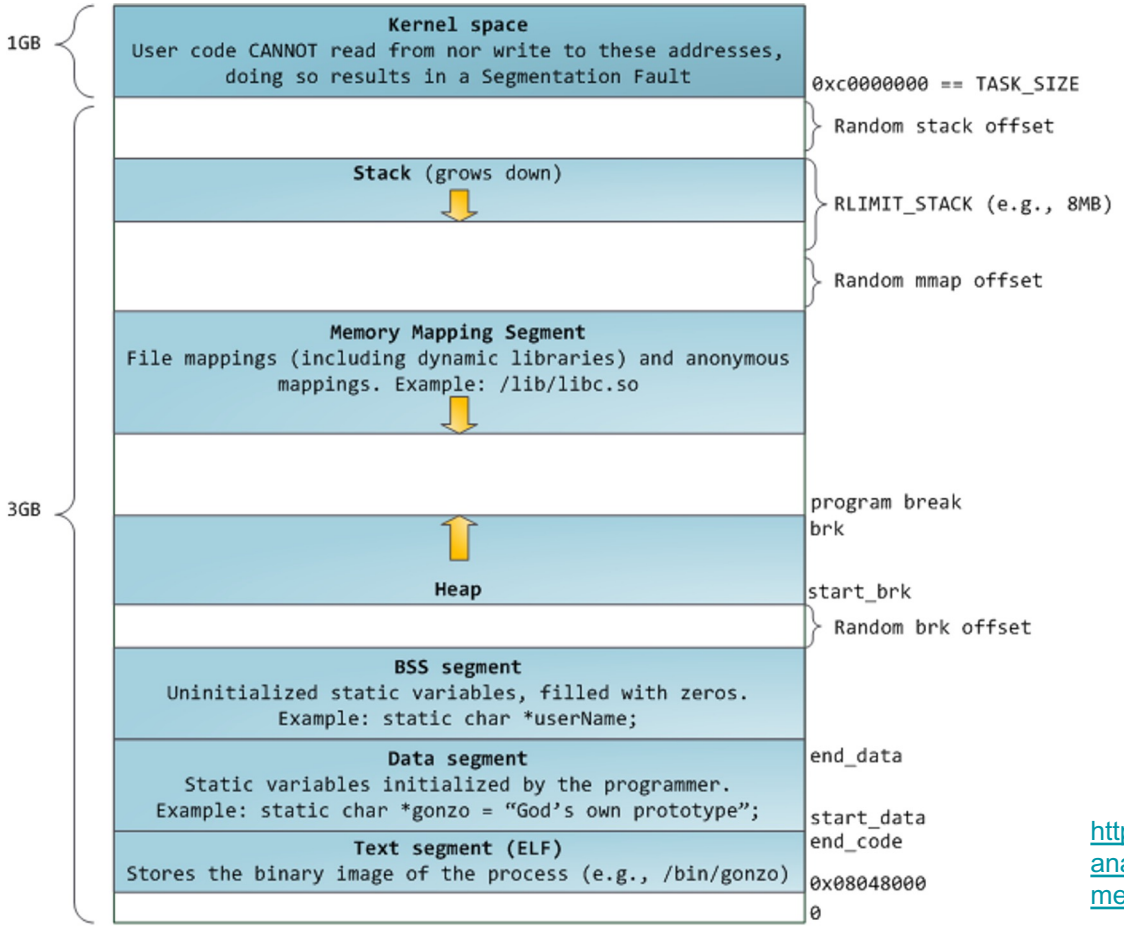
Memory Map of Linux Process (32 bit)

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/

/proc/pid_of_process/maps

Example processmap.c

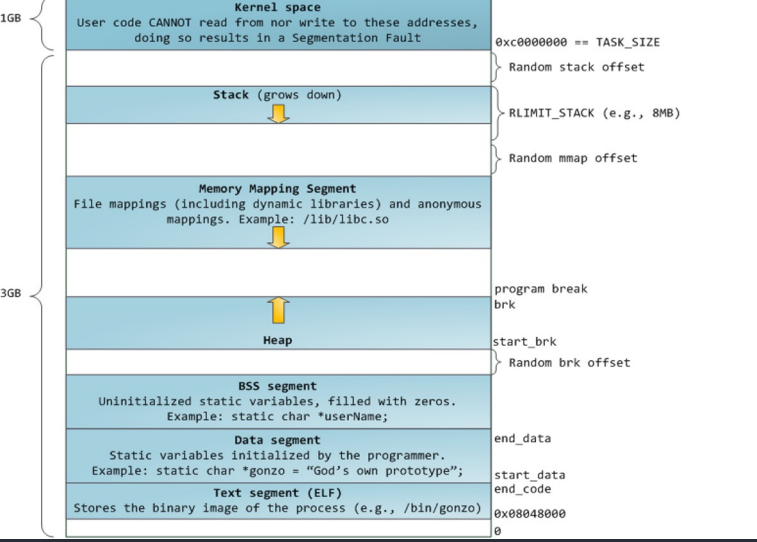
```
#include <stdio.h>
#include <stdlib.h>

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

cat /proc/pid/maps

pmap -X pid

pmap -X `pidof pm`




```
→  pm pmap -X 3592
3592:   ./pm_32
 Address Perm   Offset Device            Inode Size  Rss  Pss Referenced Anonymous LazyFree ShmemPmdMapped FilePmdMapped Shared_Hugetlb Private_Hugetlb Swap SwapPss Locked THPeligible Mapping
56591000 r--p 00000000  00:30 19703248369808013    4    4    4          4         0        0               0             0              0               0    0       0      0           0 pm_32
56592000 r-xp 00001000  00:30 19703248369808013    4    4    4          4         0        0               0             0              0               0    0       0      0           0 pm_32
56593000 r--p 00002000  00:30 19703248369808013    4    0    0          0         0        0               0             0              0               0    0       0      0           0 pm_32
56594000 r--p 00002000  00:30 19703248369808013    4    4    4          4         4        0               0             0              0               0    0       0      0           0 pm_32
56595000 rw-p 00003000  00:30 19703248369808013    4    4    4          4         4        0               0             0              0               0    0       0      0           0 pm_32
57901000 rw-p 00000000  00:00                0  136    4    4          4         4        0               0             0              0               0    0       0      0           0 [heap]
f7ca9000 r--p 00000000  08:10            35094  128  128  128        128         0        0               0             0              0               0    0       0      0           0 libc.so.6
f7cc9000 r-xp 00020000  08:10            35094 1528  732  732        732         0        0               0             0              0               0    0       0      0           0 libc.so.6
f7e47000 r--p 0019e000  08:10            35094  532   64   64         64         0        0               0             0              0               0    0       0      0           0 libc.so.6
f7ecc000 ---p 00223000  08:10            35094    4    0    0          0         0        0               0             0              0               0    0       0      0           0 libc.so.6
f7ecd000 r--p 00223000  08:10            35094    8    8    8          8         8        0               0             0              0               0    0       0      0           0 libc.so.6
f7ecf000 rw-p 00225000  08:10            35094    4    4    4          4         4        0               0             0              0               0    0       0      0           0 libc.so.6
f7ed0000 rw-p 00000000  00:00                0   40   16   16         16        16        0               0             0              0               0    0       0      0           0
f7edf000 rw-p 00000000  00:00                0    8    8    8          8         8        0               0             0              0               0    0       0      0           0
f7ee1000 r--p 00000000  00:00                0   16    0    0          0         0        0               0             0              0               0    0       0      0           0 [vvar]
f7ee5000 r-xp 00000000  00:00                0    8    4    4          4         0        0               0             0              0               0    0       0      0           0 [vdso]
f7ee7000 r--p 00000000  08:10            35077    4    4    4          4         0        0               0             0              0               0    0       0      0           0 ld-linux.so.2
f7ee8000 r-xp 00001000  08:10            35077  148  148  148        148         0        0               0             0              0               0    0       0      0           0 ld-linux.so.2
f7f0d000 r--p 00026000  08:10            35077   60   60   60         60         0        0               0             0              0               0    0       0      0           0 ld-linux.so.2
f7f1c000 r--p 00034000  08:10            35077    8    8    8          8         8        0               0             0              0               0    0       0      0           0 ld-linux.so.2
f7f1e000 rw-p 00036000  08:10            35077    4    4    4          4         4        0               0             0              0               0    0       0      0           0 ld-linux.so.2
ff99b000 rw-p 00000000  00:00                0  132   12   12         12        12        0               0             0              0               0    0       0      0           0 [stack]
                                             ==== ==== ==== ========== ========= ======== =============== ============= ============== =============== ==== ======= ====== ===========
                                             2788 1220 1220       1220        72        0               0             0              0               0    0       0      0           0 KB
```

Memory Map of Linux Process (64 bit system)

```
→  pm pmap -X 3500
3500:   ./processmap
     Address Perm   Offset Device           Inode Size  Rss Pss Referenced Anonymous LazyFree ShmemPmdMapped FilePmdMapped Shared_Hugetlb Private_Hugetlb Swap SwapPss Locked THPeligible Mapping
5629bed75000 r--p 00000000  00:30 433471464134466l4    4   4   4          0         0        0              0             0              0              0    0       0      0           0 processmap
5629bed76000 r-xp 00001000  00:30 433471464134466l4    4   4   4          4         0        0              0             0              0              0    0       0      0           0 processmap
5629bed77000 r--p 00002000  00:30 433471464134466l4    4   0   0          0         0        0              0             0              0              0    0       0      0           0 processmap
5629bed78000 r--p 00002000  00:30 433471464134466l4    4   4   4          4         4        0              0             0              0              0    0       0      0           0 processmap
5629bed79000 rw-p 00003000  00:30 433471464134466l4    4   4   4          4         4        0              0             0              0              0    0       0      0           0 processmap
5629c016a000 rw-p 00000000  00:00                0  132   4   4          4         4        0              0             0              0              0    0       0      0           0 [heap]
7f8997f0f000 rw-p 00000000  00:00                0   12   8   8          8         8        0              0             0              0              0    0       0      0           0
7f8997f12000 r--p 00000000  08:10            33247  160 160  40        160         0        0              0             0              0              0    0       0      0           0 libc.so.6
7f8997f3a000 r-xp 00028000  08:10            33247 1620 712 186        712         0        0              0             0              0              0    0       0      0           0 libc.so.6
7f89980cf000 r--p 001bd000  08:10            33247  352  64  16         64         0        0              0             0              0              0    0       0      0           0 libc.so.6
7f8998127000 r--p 00214000  08:10            33247   16  16  16         16        16        0              0             0              0              0    0       0      0           0 libc.so.6
7f899812b000 rw-p 00218000  08:10            33247    8   8   8          4         8        0              0             0              0              0    0       0      0           0 libc.so.6
7f899812d000 rw-p 00000000  00:00                0   52  20  20         16        20        0              0             0              0              0    0       0      0           0
7f899813f000 rw-p 00000000  00:00                0    8   4   4          4         4        0              0             0              0              0    0       0      0           0
7f8998141000 r--p 00000000  08:10            33244    8   8   2          8         0        0              0             0              0              0    0       0      0           0 ld-linux-x86-64.so.2
7f8998143000 r-xp 00002000  08:10            33244  168 168  42        168         0        0              0             0              0              0    0       0      0           0 ld-linux-x86-64.so.2
7f899816d000 r--p 0002c000  08:10            33244   44  40  10         40         0        0              0             0              0              0    0       0      0           0 ld-linux-x86-64.so.2
7f8998179000 r--p 00037000  08:10            33244    8   8   8          8         8        0              0             0              0              0    0       0      0           0 ld-linux-x86-64.so.2
7f899817b000 rw-p 00039000  08:10            33244    8   8   8          8         8        0              0             0              0              0    0       0      0           0 ld-linux-x86-64.so.2
7ffe5fc2d000 rw-p 00000000  00:00                0  132  16  16         16        16        0              0             0              0              0    0       0      0           0 [stack]
7ffe5fc92000 r--p 00000000  00:00                0   16   0   0          0         0        0              0             0              0              0    0       0      0           0 [vvar]
7ffe5fc96000 r-xp 00000000  00:00                0    4   4   0          4         0        0              0             0              0              0    0       0      0           0 [vdso]
                                                 ==== ==== === ========== ========= ======== ============== ============= ============== ============== ==== ======= ====== ===========
                                                 2768 1264 404       1252       100        0              0             0              0              0    0       0      0           0 KB
```