Link Example 1.A Dynamic Linking - Example

Young W. Lim

2019-01-21 Mon

Outline

1. Based on
2. background
    - readelf background
    - PIC background
3. example codes
4. examining dynamic linker path
    - dynamic linker path
    - segments of PIC
5. examining shared library function calls
    - shared library function calls
    - section headers, contents, and disassemblies
    - relocation informations
    - dynamic section informations
    - using gdb

Based on

1. https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-relocation-elf.html

Compling 32-bit program on 64-bit gcc

- `gcc -v`
- `gcc -m32 t.c`
- `sudo apt-get install gcc-multilib`
- `sudo apt-get install g++-multilib`
- `gcc-multilib`
- `g++-multilib`
- `gcc -m32`
- `objdump -m i386`

sections vs. segments

- sections provide information about how information is organized within a binary file
- segments describe to the program loader and the dynamic linker (the dynamic linker if the binary is dynamically linked) how a process image should be composed in virtual memory
- `readelf -SW -l <binary>` shows the difference between sections and segments

- `readelf -l (--program-headers, --segments)`
- `readelf -S (--section-headers, --sections)`

https://reverseengineering.stackexchange.com/questions/17258/elf-file-format-find

section header table

- information about sections is stored in the section header table
- to find information about sections in a binary, parse the section header table.
- the section header table is not required to be present in the binary
- the loader only uses segment information to accomplish process creation
- `.got` and `.got.plt` are examples of labels that describe sections and never segments.

https://reverseengineering.stackexchange.com/questions/17258/elf-file-format-find

segment header table

- an array of structures, each describing
  a segment or other information the system needs
  to prepare the program for execution
- An object file segment contains *one or more* sections
- Program headers are meaningful only for
  executable and shared object files

https://reverseengineering.stackexchange.com/questions/17258/elf-file-format-find

readelf --sections output columns

- sh_name: the name of the section.
- sh_type: categorizes the section's contents and semantics
- sh_flags: one-bit flags that describe miscellaneous attributes
- sh_addr: the address of the section's first byte
  in the memory image of a process,
- sh_offset: the byte offset from the beginning of the file
  to the first byte in the section.
- sh_size: the section's size in bytes
- sh_link: a section header table index link
- sh_info: extra information
- sh_addralign: address alignment constraints
- sh_entsize: a table of fixed-sized entries, such as a symbol table

https://reverseengineering.stackexchange.com/questions/17258/elf-file-format-find

readelf --segments output columns

- p_offset: the offset from the beginning of the file
- p_vaddr: the virtual address in memory
- p_paddr: reserved for the segment's physical address.
- p_filesz: the number of bytes in the file image of the segment.
- p_memsz: the number of bytes in the memory image of the segment.
- p_flags: a bit mask of flags relevant to the segment:
  - PF_X, PF_W, PF_R
  - A text segment commonly has the flags PF_X and PF_R.
  - A data segment commonly has PF_X, PF_W, and PF_R.
- p_align: the value to which the segments are aligned

https://reverseengineering.stackexchange.com/questions/17258/elf-file-format-find

readelf -r output columns

- Offset is the offset where the symbol value should go
- Info tells us two things
  - the type (depends on the arch)
  - the symbol index in the symtab
- Type - type of the symbol according to the ABI
- Sym value is the addend to be added to the symbol resolution
- Sym name and addend - a pretty printing of the symbol name + addend.

```
 Offset     Info    Type            Sym.Value  Sym. Name
00001ff4  00000406 R_386_GLOB_DAT    00000000   __gmon_start__
00001fe4  00000107 R_386_JUMP_SLOT   00000000   doAlmostNothing
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

PLT and GOT

- GOT (Global Offset Table) . . . `.data` . . . RW
- PLT (Procedure Linkage Table) . . . `.text` . . . RO

- in shared libraries,
  PC-Relative or absolute relocation is not used
- the call/access will have to be done via the PLT/GOT

`https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca`

access through PLT/GOT

- .got and .got.plt will be loaded in RW memory pages due to the security limitations
- their entries will be filled at runtime:
  - at program startup for global variables (.got)
  - on the first call to a function (.got.plt)

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

.got, .got.plt, .plt, .plt.got (1)

- text segment
  - Read only
  - .plt, .plt.got (the plt for the got)
- data segement
  - Read Write
  - .got, .got.plt (the got for the plt)
-
  ```
  LOAD          R E   .plt .plt.got .text
  LOAD          RW    .got .got.plt .data .bss
  ```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

.got, .got.plt, .plt, .plt.got (2)

- .got
  This is the GOT, or Global Offset Table. This is the actual table of offsets as filled in by the linker for external symbols.
- .plt
  This is the PLT, or Procedure Linkage Table. These are stubs that look up the addresses in the .got.plt section, and either jump to the right address, or trigger the code in the linker to look up the address. (If the address has not been filled in to .got.plt yet.)
- .got.plt
  This is the GOT for the PLT. It contains the target addresses (after they have been looked up) or an address back in the .plt to trigger the lookup. Classically, this data was part of the .got section.
- .plt.got
  It seems like they wanted every combination of PLT and GOT! unknown purpose. . .

https://systemoverlord.com/2017/03/19/got-and-plt-for-pwning.html

.rel.dyn vs. .rel.plt

- almost all the relocation type for .rel.dyn :
  R_386_GLOB_DAT (global variables)
- all the relocation type for .rel.plt :
  R_386_JUMP_SLOT (branch relocation) all

https://stackoverflow.com/questions/11676472/what-is-the-difference-between-got-a

.symtab vs. .dynsym

- the symbol table .symtab
  contain references for all symbols used during static link editing
- the symbol table .dynsym
  contain only those symbols needed for dynamic linking.

https://stackoverflow.com/questions/11676472/what-is-the-difference-between-got-a

the gnu assembler as suffix (1)

- Usually, all absolute symbol values must be located in a table, the global offset table, leaving the code position-independent;
  - independent of values of global symbols
  - independent of the address of the code

- The suffix modifies the value of the symbol into
  1. an index into the got
  2. a PC-relative value
  3. a value relative to the start of the got

`http://www.fdi.ucm.es/profesor/mendias/PSyD/docs/as.pdf`

the gnu assembler as suffix (2)

- every symbol use in code or a read-only section *must* have a PIC suffix for a useful shared library
- these constructs *must* not be used

with an additive constant offset
as is usually allowed (i.e. no 4 as in symbol + 4)

- This restriction is checked at link-time, not at assembly-time

http://www.fdi.ucm.es/profesor/mendias/PSyD/docs/as.pdf

as suffix : GOT

- attaching `:GOT` suffix to a symbol in an instruction causes the symbol (`extsym`) to be entered into the got
- the value (`extsym:GOT`) is a 32-bit index for that symbol into the got (`.data`) (an entry of the got)
- the name of the relocation is 'R_CRIS_32_GOT'.
- `move.d [$r0+extsym:GOT],$r9`

http://www.fdi.ucm.es/profesor/mendias/PSyD/docs/as.pdf

as suffix : PLT (1)

- `:PLT` suffix is used for function symbols.
- this creates a plt, an array of code stubs,
  at the time the shared object is created or linked against
  together with the corresponding got entry
- each entry of plt (a code stub) is associated with
  the got entry
- the value `fnname:PLT` is a pc-relative offset
  to the corresponding stub code in the plt (`.text`)

`http://www.fdi.ucm.es/profesor/mendias/PSyD/docs/as.pdf`

as suffix : PLT (2)

- the run-time symbol resolver will be called to look up and set the value of the symbol the first time the function is called (at latest; depending environment variables).
- It is only safe to leave the symbol unresolved this way if all references are function calls.
- the name of the relocation is 'R_CRIS_32_PLT_PCREL'
- `add.d fnname:PLT,$pc`

http://www.fdi.ucm.es/profesor/mendias/PSyD/docs/as.pdf

as suffix : PLTG

- Like PLT
- but the value `fnname:PLTG` is relative to the beginning of the got
- not a pc-relative offset
- the relocation is 'R_CRIS_32_PLT_GOTREL'.
- `move.d fnname:PLTG,$r3`

http://www.fdi.ucm.es/profesor/mendias/PSyD/docs/as.pdf

as suffix : GOTPLT

- similar to PLT
- the value of the symbol (`fnname:GOTPLT`) is a 32-bit index into the got (`.data`)
- a mix between the effect of the GOT and the PLT suffix;
- the difference to GOT is that
  - there will be a plt entry created
  - the symbol is assumed to be a function entry
  - will be resolved by the run-time resolver as with PLT
- The relocation is 'R_CRIS_32_GOTPLT'
- `jsr [$r0+fnname:GOTPLT]`

`http://www.fdi.ucm.es/profesor/mendias/PSyD/docs/as.pdf`

nmain.c

-
  ```
  // nothing.h ------------------------------------
  void doAlmostNothing();
  ```

-
  ```
  // nmain.c -------------------------------------
  #include "nothing.h"

  int main(int argc, const char *argv[])
  {
    doAlmostNothing();
    return 0;
  }
  ```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

doNothingStatic, doNothing, doAlmostNothing

-
  ```
  // nothing.c ------------------------------------
  static void doNothingStatic() {
  }

  void doNothing() {
  }

  void doAlmostNothing() {
    doNothingStatic();
    doNothing();
  }
  ```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

commands for the dynamic linking

-
  ```
  $ gcc -c -fPIC -m32 nothing.c
  $ gcc -shared -m32 -o libnothing.so nothing.o
  $ gcc -c -m32 nmain.c
  $ gcc -m32 -o nmain_dyn.out nmain.o ./libnothing.so
  ```
- `-Wall -g -O0`
  ```
  $ gcc -Wall -g -O0 -fPIC -c -m32 nothing.c -o nothing_pic.o
  $ gcc -shared -m32 -o libnothing.so nothing_pic.o
  $ gcc -Wall -g -O0 -c -m32 nmain.c
  $ gcc -m32 -o nmain_dyn.out nmain.o ./libnothing.so
  ```

nothing.c	⟶	nothing_pic.o
nothing_pic.o	⟶	libnothing.so
nmain.c	⟶	nmain.o
nmain.c libnothing.so	⟶	nmain_dyn.out

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

commands for examining shared library function calls

-
  ```
  $ gcc -Wall -g -O0 -fPIC -c -m32 nothing.c -o nothing_pic.o
  $ gcc -shared -m32 -o libnothing.so nothing_pic.o
  $ gcc -Wall -g -O0 -c -m32 nmain.c
  $ gcc -m32 -o nmain_dyn.out nmain.o ./libnothing.so

  $ readelf --segments nmain_dyn.out
  $ objdump -d -s nmain.out
  $ objdump -d -s nmain_dyn.out
  $ objdump -d -j .plt.got nmain_dyn.out
  $ objdump -d -j .plt.got nmain_dyn.out
  $ gdb ... disas, x/a 0x...., c
  $ cat /proc/<pid>/map
  ```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

dynamic linker for a shared code

- several programs would jump to the shared code
  in memory to execute this common code.
- the virtual memory system will hide the actual position
- the addresses of the shared code at runtime
- dynamic linker relocates the undefined symbols at runtime
- this special process is by the `glibc`

`https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca`

reference path to the dynamic linker

- An executable that depends upon shared libraries, holds a reference to the path toward the dynamic linker to use
- this path is stored in the .interp section of the executable elf file:

-
  ```
  $readelf -S nmain_dyn.out  ..... address = 154 (.interp Addr)
  $hexdump -C nmain_dyn.out  ..... path = /lib/ld-linux.so.2
  ```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

readelf -S (1)

```
young@USys1:~$ readelf -S nmain_dyn.out
There are 29 section headers, starting at offset 0x17a8:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        00000154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            00000168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            00000188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        000001ac 0001ac 00003c 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          000001e8 0001e8 0000d0 10   A  6   1  4
  [ 6] .dynstr           STRTAB          000002b8 0002b8 0000da 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          00000392 000392 00001a 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         000003ac 0003ac 000030 00   A  6   1  4
  [ 9] .rel.dyn          REL             000003dc 0003dc 000040 08   A  5   0  4
  [10] .rel.plt          REL             0000041c 00041c 000010 08  AI  5  22  4
  [11] .init             PROGBITS        0000042c 00042c 000023 00  AX  0   0  4
  [12] .plt              PROGBITS        00000450 000450 000030 04  AX  0   0 16
  [13] .plt.got          PROGBITS        00000480 000480 000010 08  AX  0   0  8
  [14] .text             PROGBITS        00000490 000490 0001d2 00  AX  0   0 16
  [15] .fini             PROGBITS        00000664 000664 000014 00  AX  0   0  4
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

readelf -S (2)

```
  [16] .rodata           PROGBITS        00000678 000678 000008 00   A  0   0  4
  [17] .eh_frame_hdr     PROGBITS        00000680 000680 00003c 00   A  0   0  4
  [18] .eh_frame         PROGBITS        000006bc 0006bc 0000fc 00   A  0   0  4
  [19] .init_array       INIT_ARRAY      00001ed0 000ed0 000004 04  WA  0   0  4
  [20] .fini_array       FINI_ARRAY      00001ed4 000ed4 000004 04  WA  0   0  4
  [21] .dynamic          DYNAMIC         00001ed8 000ed8 000100 08  WA  6   0  4
  [22] .got              PROGBITS        00001fd8 000fd8 000028 04  WA  0   0  4
  [23] .data             PROGBITS        00002000 001000 000008 00  WA  0   0  4
  [24] .bss              NOBITS          00002008 001008 000004 00  WA  0   0  1
  [25] .comment          PROGBITS        00000000 001008 00002a 01  MS  0   0  1
  [26] .symtab           SYMTAB          00000000 001034 000430 10     27  43  4
  [27] .strtab           STRTAB          00000000 001464 000248 00      0   0  1
  [28] .shstrtab         STRTAB          00000000 0016ac 0000fc 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  p (processor specific)
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

hexdump -C

- readelf -S

```
[Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
[ 1] .interp           PROGBITS        00000154 000154 000013 00   A  0   0  1
```

Addr = 154

- hexdump -C nmain_dyn.out

```
...
00000140  d0 1e 00 00 30 01 00 00  30 01 00 00 04 00 00 00  |....0...0.......|
00000150  01 00 00 00 2f 6c 69 62  2f 6c 64 2d 6c 69 6e 75  |..../lib/ld-linu|
00000160  78 2e 73 6f 2e 32 00 00  04 00 00 00 10 00 00 00  |x.so.2..........|
...
```

/lib/ld-linux.so.2

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

libnothing.so segment headers summary

```
LOAD          R E  00 .dynsym .dynstr .rel.dyn .rel.plt .plt .plt.got .text
LOAD          RW   01 .dynamic .got .got.plt .data .bss
DYNAMIC       RW   02 .dynamic
NOTE          R    03
GNU_EH_FRAME  R    04
GNU_STACK     RW   05
GNU_RELRO     R    06 .dynamic .got
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

nmain_dyn.out segment headers summary

```
Type         VirtAddr   Flg
PHDR         0x00000034 R   00
INTERP       0x00000154 R   01  .interp
                            02  .interp .dynsym .dynstr .rel.dyn .rel.plt
LOAD         0x00000000 R E     .init .plt .plt.got .text
LOAD         0x00001ed0 RW  03  .got .data .bss
DYNAMIC      0x00001ed8 RW  04  .dynamic
NOTE         0x00000168 R   05
GNU_EH_FRAME 0x00000680 R   06
GNU_STACK    0x00000000 RW  07
GNU_RELRO    0x00001ed0 R   08  .dynamic .got
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

readelf -segments libnothing.so (1)

```
young@USys1:~$ readelf --segments libnothing.so

Elf file type is DYN (Shared object file)
Entry point 0x360
There are 7 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  LOAD           0x000000 0x00000000 0x00000000 0x005c0 0x005c0 R E 0x1000
  LOAD           0x000f28 0x00001f28 0x00001f28 0x000ec 0x000f0 RW  0x1000
  DYNAMIC        0x000f30 0x00001f30 0x00001f30 0x000c0 0x000c0 RW  0x4
  NOTE           0x000114 0x00000114 0x00000114 0x00024 0x00024 R   0x4
  GNU_EH_FRAME   0x0004b8 0x000004b8 0x000004b8 0x0003c 0x0003c R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x000f28 0x00001f28 0x00001f28 0x000d8 0x000d8 R   0x1
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

readelf -segments libnothing.so (2)

```
Section to Segment mapping:
 Segment Sections...
  00     .note.gnu.build-id .gnu.hash .dynsym .dynstr .rel.dyn
         .rel.plt .init .plt .plt.got .text .fini .eh_frame_hdr .eh_frame
  01     .init_array .fini_array .dynamic .got .got.plt .data .bss
  02     .dynamic
  03     .note.gnu.build-id
  04     .eh_frame_hdr
  05
  06     .init_array .fini_array .dynamic .got
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

readelf -segments nmain_dyn.out (1)

```
young@USys1:~$ readelf --segments nmain.out

Elf file type is DYN (Shared object file)
Entry point 0x490
There are 9 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x00000034 0x00000034 0x00120 0x00120 R   0x4
  INTERP         0x000154 0x00000154 0x00000154 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x00000000 0x00000000 0x007b8 0x007b8 R E 0x1000
  LOAD           0x000ed0 0x00001ed0 0x00001ed0 0x00138 0x0013c RW  0x1000
  DYNAMIC        0x000ed8 0x00001ed8 0x00001ed8 0x00100 0x00100 RW  0x4
  NOTE           0x000168 0x00000168 0x00000168 0x00044 0x00044 R   0x4
  GNU_EH_FRAME   0x000680 0x00000680 0x00000680 0x0003c 0x0003c R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x10
  GNU_RELRO      0x000ed0 0x00001ed0 0x00001ed0 0x00130 0x00130 R   0x1
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

readelf -segments nmain_dyn.out (2)

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

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

function call to doAlmostNothing

-
  ```
  main +--> doAlmostNothing +--> doNothingStatic
                            +--> doNothing
  ```

    - does not jump directly to the function
      but to an intermediary code linked to the PLT
      (doAlmostNothing `@plt`)

`https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca`

static linking vs. dynamic linking

- the statically linked executable

```
objdump -d -s nmain.out

000005cd <main>:
...
508:   e8 30 00 00 00          call   53d <doAlmostNothing>

0000053d <doAlmostNothing>:
...
```

- the dynamically linked executable

```
objdump -d -s nmain_dyn.out

000005cd <main>:
...
5e8:   e8 73 fe ff ff          call   460 <doAlmostNothing@plt>

00000460 <doAlmostNothing@plt>:  -- .plt entry
...
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

1. main calls <doAlmostNothing@plt>

- <main>

```
5dc:  call   5f9 <__x86.get_pc_thunk.ax>  ;; push $0x5e1
                                          ;; jmp  0x5f9
5e1:  add    $0x19f7,%eax                 ;; $0x19f7+$0x5e1= $0x1fd8
                                          ;; mov $0x1fdb,%eax
5e6:  mov    %eax,%ebx                    ;; mov $0x1fdb,%ebx
5e8:  call   460 <doAlmostNothing@plt>    ;; push $0x05ed
                                          ;; jmp 0x460
```

- <__x86.get_pc_thunk.ax>

```
5f9:  mov    (%esp),%eax                 ;; mov $0x5e1,%eax
5fc:  ret
```

- .got section address at 0x1fd8

```
[22] .got              PROGBITS        00001fd8 000fd8 000028 04  WA  0   0  4
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

2. indirect function call through PLT

- the dynamically linked executable
- at main, call   460 <doAlmostNothing@plt>
- .plt starts at 450
- the first entry PLT[0] starts 450
- the second entry PLT[1] starts 460
- the first instruction of PLT[1] jumps to GOT[3]
- PLT[1]

```
00000460 <doAlmostNothing@plt>:
 460: ff a3 0c 00 00 00       jmp    *0xc(%ebx)
 466: 68 00 00 00 00          push   $0x0
 46b: e9 e0 ff ff ff          jmp    450 <.plt>
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

3. jump to *GOT[3] instruction at PLT[1]

- 1st instruction of PLT[1]

```
00000460 <doAlmostNothing@plt>:
 460: jmp    *0xc(%ebx)     ;; jmp  *($0xc + $0x1fd8) -- (12=3*4)
                            ;; jmp  *GOT[3] (=0x466)
                            ;; jump to the 2nd inst at PLT[1]
```

- *GOT[3] : lazy binding address for doAlmostNothing
  dynamic linker will overwrite the correct address at *GOT[3]

- GOT : disassembly of section .got

```
00001fd8 <_GLOBAL_OFFSET_TABLE_>:
  1fd8:     d8 1e 00 00   ;; 1fd8 + 0
            00 00 00 00   ;; 1fd8 + 4
            00 00 00 00   ;; 1fd8 + 8
            66 04 00 00   ;; 1fd8 + c ---> 0x466
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

4. push 0x0 and jump to &PLT[0] instruction at PLT[1]

- 1st instruction of PLT[1]

```
00000460 <doAlmostNothing@plt>:
 460: jmp    *0xc(%ebx)     ;; jmp  *($0xc + $0x1fd8) -- (12=3*4)
                            ;; jmp  *GOT[3] (=0x466)  ---->
                            ;; jump to the 2nd inst at PLT[1]
```

- 2nd and 3rd instucitons of PLT[1]

```
466: push   $0x0           ;; push ID 0 for doAlmostNothing  <--- 0x466
46b: jmp    450 <.plt>     ;; jmp to &PLT[0]
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

5. push &GOT[1] instruction at PLT[0]

- PLT[0]

```
450: pushl   0x4(%ebx)       ;; push $0x1fd8+4 = $0x1fdc = &GOT[1]
```

- &GOT[1] = 0x1fdc
- *GOT[1] = 0x0000 :
  info for the dynamic linker, indentifying nmain.o module

- GOT : disassembly of section .got

```
1fd8:     d8 1e 00 00   ;; 1fd8 + 0 ---> address of .dynamic section
1fdc:     00 00 00 00   ;; 1fd8 + 4 ---> identifying info
1fe0:     00 00 00 00   ;; 1fd8 + 8 ---> entry point in dynamic linker
1fe4:     66 04 00 00   ;; 1fd8 + c ---> 0x466 = &PLT[1]
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

6. jump to *GOT[2] instruction at PLT[0]

- PLT[0]

```
00000450 <.plt>:
 450: pushl  0x4(%ebx)      ;; push $0x1fd8+4 = $0x1fdc = &GOT[1]
 456: jmp    *0x8(%ebx)     ;; jump *($0x1fd8+8) = *($0x1fe0) = *GOT[2]
 45c: add    %al,(%eax)
      ...
```

- GOT[2] contains an entry point into the lazy binding code of the dynamic linker
- GOT : disassembly of section .got

```
00001fd8 <_GLOBAL_OFFSET_TABLE_>:
  1fd8:     d8 1e 00 00    ;; 1fd8 + 0 ---> address of .dynamic section
  1fdc:     00 00 00 00    ;; 1fd8 + 4 ---> identifying info
  1fe0:     00 00 00 00    ;; 1fd8 + 8 ---> entry point in dynamic linker
  1fe4:     66 04 00 00    ;; 1fd8 + c ---> 0x466 = &PLT[1]
```

- 0x1fdc = &GOT[1]

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

__x86.get_pc_thunk.ax

-
  ```
  000005f9 <__x86.get_pc_thunk.ax>:
   5f9: 8b 04 24                  mov    (%esp),%eax
   5fc: c3                        ret
   5fd: 66 90                     xchg   %ax,%ax
   5ff: 90                        nop
  ```
- called at main to store PC to %eax

  ```
  000005cd <main>:
   ...
   5dc: e8 18 00 00 00            call   5f9 <__x86.get_pc_thunk.ax>
   5e1: 05 f7 19 00 00            add    $0x19f7,%eax
   5e6: 89 c3                     mov    %eax,%ebx
   5e8: e8 73 fe ff ff            call   460 <doAlmostNothing@plt>
  ```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

main disassembly

- 000005cd <main>:

```
  5cd: 8d 4c 24 04             lea    0x4(%esp),%ecx
  5d1: 83 e4 f0                and    $0xfffffff0,%esp
  5d4: ff 71 fc                pushl  -0x4(%ecx)
  5d7: 55                      push   %ebp
  5d8: 89 e5                   mov    %esp,%ebp
  5da: 53                      push   %ebx
  5db: 51                      push   %ecx
  5dc: e8 18 00 00 00          call   5f9 <__x86.get_pc_thunk.ax>
  5e1: 05 f7 19 00 00          add    $0x19f7,%eax
  5e6: 89 c3                   mov    %eax,%ebx
  5e8: e8 73 fe ff ff          call   460 <doAlmostNothing@plt>
  5ed: b8 00 00 00 00          mov    $0x0,%eax
  5f2: 59                      pop    %ecx
  5f3: 5b                      pop    %ebx
  5f4: 5d                      pop    %ebp
  5f5: 8d 61 fc                lea    -0x4(%ecx),%esp
  5f8: c3                      ret
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

.got and .got.plt

- .got and .got.plt will be loaded in RW memory pages
- their entries will be filled at runtime:
  - at program startup for global variables (.got)
  - on the first call to a function (.got.plt)
- <<libnothing.so>>

```
LOAD           0x00000000 R E  00 .dynsym .dynstr .rel.dyn .rel.plt .plt .plt.g
LOAD           0x00001f28 RW   01 .dynamic .got .got.plt .data .bss
```

- <<nmain_dyn.out>>

```
LOAD          0x00000000 R E     .init .plt .plt.got .text
LOAD          0x00001ed0 RW   03  .got .data .bss
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

.plt, .plt.got, .got section headers

- `readelf --sections nmain_dyn.out`

```
Section Headers:
[Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al

[12] .plt              PROGBITS        00000450 000450 000030 04  AX  0   0 16
[13] .plt.got          PROGBITS        00000480 000480 000010 08  AX  0   0  8

[22] .got              PROGBITS        00001fd8 000fd8 000028 04  WA  0   0  4
```

`https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca`

.plt, .plt.got, .got section conents

- objdump -d -s nmain_dyn.out

```
Contents of section .plt:
 0450 ffb30400 0000ffa3 08000000 00000000  ................
 0460 ffa30c00 00006800 000000e9 e0ffffff  ......h.........
 0470 ffa31000 00006808 000000e9 d0ffffff  ......h.........
Contents of section .plt.got:
 0480 ffa31800 00006690 ffa31c00 00006690  ......f.......f.

Contents of section .got:
 1fd8 d81e0000 00000000 00000000 66040000  ............f...
 1fe8 76040000 00000000 00000000 00000000  v...............
 1ff8 cd050000 00000000                    ........
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

.plt section disassembly

```
objdump -d -s nmain_dyn.out

Disassembly of section .plt:

00000450 <.plt>:
 450:    ff b3 04 00 00 00        pushl   0x4(%ebx)
 456:    ff a3 08 00 00 00        jmp     *0x8(%ebx)
 45c:    00 00                    add     %al,(%eax)
         ...

00000460 <doAlmostNothing@plt>:
 460:    ff a3 0c 00 00 00        jmp     *0xc(%ebx)
 466:    68 00 00 00 00           push    $0x0
 46b:    e9 e0 ff ff ff           jmp     450 <.plt>

00000470 <__libc_start_main@plt>:
 470:    ff a3 10 00 00 00        jmp     *0x10(%ebx)
 476:    68 08 00 00 00           push    $0x8
 47b:    e9 d0 ff ff ff           jmp     450 <.plt>
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

.plt.got section disassembly

```
 objdump -d -j .plt.got nmain_dyn.out

nmain_dyn.out:     file format elf32-i386

Disassembly of section .plt.got:

00000480 <__cxa_finalize@plt>:
 480:   ff a3 18 00 00 00       jmp    *0x18(%ebx)
 486:   66 90                   xchg   %ax,%ax

00000488 <__gmon_start__@plt>:
 488:   ff a3 1c 00 00 00       jmp    *0x1c(%ebx)
 48e:   66 90                   xchg   %ax,%ax
young@USys1:~$ objdump -d -s -j .got nmain_dyn.out
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

.got section disassembly

```
objdump -d -j .got nmain_dyn.out

nmain_dyn.out:     file format elf32-i386


Disassembly of section .got:

00001fd8 <_GLOBAL_OFFSET_TABLE_>:
    1fd8:       d8 1e 00 00 00 00 00 00 00 00 00 00 66 04 00 00     ............f..
    1fe8:       76 04 00 00 00 00 00 00 00 00 00 00 00 00 00 00     v..............
    1ff8:       cd 05 00 00 00 00 00 00                             ........

  https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca
```

.text section disassembly of doNothingStatic

```
objdump -d -s libnothing.so

Disassembly of section .text:
...

0000045d <doNothingStatic>:
 45d:   55                      push   %ebp
 45e:   89 e5                   mov    %esp,%ebp
 460:   e8 3b 00 00 00          call   4a0 <__x86.get_pc_thunk.ax>
 465:   05 9b 1b 00 00          add    $0x1b9b,%eax
 46a:   90                      nop
 46b:   5d                      pop    %ebp
 46c:   c3                      ret
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

.text section disassembly of doNothing

```
objdump -d -s libnothing.so

Disassembly of section .text:
...

0000046d <doNothing>:
 46d:   55                      push   %ebp
 46e:   89 e5                   mov    %esp,%ebp
 470:   e8 2b 00 00 00          call   4a0 <__x86.get_pc_thunk.ax>
 475:   05 8b 1b 00 00          add    $0x1b8b,%eax
 47a:   90                      nop
 47b:   5d                      pop    %ebp
 47c:   c3                      ret
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

.text section disassembly of doAlmostNothing

```
objdump -d -s libnothing.so

Disassembly of section .text:
...

0000047d <doAlmostNothing>:
 47d:   55                      push   %ebp
 47e:   89 e5                   mov    %esp,%ebp
 480:   53                      push   %ebx
 481:   83 ec 04                sub    $0x4,%esp
 484:   e8 d7 fe ff ff          call   360 <__x86.get_pc_thunk.bx>
 489:   81 c3 77 1b 00 00       add    $0x1b77,%ebx
 48f:   e8 c9 ff ff ff          call   45d <doNothingStatic>
 494:   e8 a7 fe ff ff          call   340 <doNothing@plt>
 499:   90                      nop
 49a:   83 c4 04                add    $0x4,%esp
 49d:   5b                      pop    %ebx
 49e:   5d                      pop    %ebp
 49f:   c3                      ret
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

.dynamic section disassembly

```
objdump - -j .dynamic nmain_dyn.out

nmain_dyn.out:     file format elf32-i386

Disassembly of section .dynamic:

00001ed8 <_DYNAMIC>:
    1ed8:       01 00 00 00 01 00 00 00 01 00 00 00 81 00 00 00     ...............
    1ee8:       0c 00 00 00 2c 04 00 00 0d 00 00 00 64 06 00 00     ....,.......d..
    1ef8:       19 00 00 00 d0 1e 00 00 1b 00 00 00 04 00 00 00     ...............
    1f08:       1a 00 00 00 d4 1e 00 00 1c 00 00 00 04 00 00 00     ...............
    1f18:       f5 fe ff 6f ac 01 00 00 05 00 00 00 b8 02 00 00     ...o...........
    1f28:       06 00 00 00 e8 01 00 00 0a 00 00 00 da 00 00 00     ...............
    1f38:       0b 00 00 00 10 00 00 00 15 00 00 00 00 00 00 00     ...............
    1f48:       03 00 00 00 d8 1f 00 00 02 00 00 00 10 00 00 00     ...............
    1f58:       14 00 00 00 11 00 00 00 17 00 00 00 1c 04 00 00     ...............
    1f68:       11 00 00 00 dc 03 00 00 12 00 00 00 40 00 00 00     ............@..
    1f78:       13 00 00 00 08 00 00 00 1e 00 00 00 08 00 00 00     ...............
    1f88:       fb ff ff 6f 01 00 00 08 fe ff ff 6f ac 03 00 00     ...o.......o...
    1f98:       ff ff ff 6f 01 00 00 00 f0 ff ff 6f 92 03 00 00     ...o.......o...
    1fa8:       fa ff ff 6f 04 00 00 00 00 00 00 00 00 00 00 00     ...o...........
        ...
```

https://stac47.github.io/c/relocation/elf/tutorial/2018/03/01/understanding-reloca

relocation sections

- `readelf -r, --relocs`

  displays the contents of the file's relocation section,
  if it has one.

- `.rel.bss` contains all the `R_386_COPY` relocs
- `.rel.plt` contains all the `R_386_JMP_SLOT` relocs
  these modify the first half of the GOT elements
- `.rel.got` contains all the `R_386_GLOB_DATA` relocs
  these modify the second half of the GOT elements
- `.rel.data` contains all the `R_386_32` and `R_386_RELATIVE` relocs

http://netwinder.osuosl.org/users/p/patb/public_html/elf_relocs.html

relocation types in i386 (1)

- R_386_JMP_SLOT relocs (.rel.plt)
  at dynamic link time, deposit the address of "symbol"
  (a subroutine) into this dword

  ```
  00001fe4  00000107 R_386_JUMP_SLOT   00000000   doAlmostNothing
  ```

- R_386_COPY relocs (.rel.bss)
  read a string of bytes from the "symbol" address
  and deposit a copy into this location;
  the "symbol" object has an intrinsic length
  i.e. move initialized data from a library down
  into the app data space

http://netwinder.osuosl.org/users/p/patb/public_html/elf_relocs.html

relocation types in i386 (2)

- R_386_GLOB_DATA relocs (.rel.got)
  at load time, deposit the address of "symbol" into this dword;
  the "symbol" is in another module this reloc is,
  in a sense, the complement of the R_386_COPY above

  ```
  00001ff4  00000406 R_386_GLOB_DAT    00000000   __gmon_start__
  ```

- R_386_RELATIVE relocs (.rel.data)
  at dynamic link time, read the dword at this location,
  add it to the run-time start address of this module;
  deposit the result back into this dword

http://netwinder.osuosl.org/users/p/patb/public_html/elf_relocs.html

dynamic relocation section

- The dynamic relocation section describes all locations within the object that must be adjusted if the object is loaded at an address other than its linked base address.
- Only one dynamic relocation section `.rel.dyn` is used to resolve addresses in data items,

`https://www3.physnet.uni-hamburg.de/physnet/Tru64-Unix/HTML/APS31DTE/DOCU_002.HTM`

normal vs. dynamic relocation sections

- Shared executable files can contain normal relocation sections in addition to a dynamic relocation section.
- The normal relocation sections may contain resolutions for any absolute values in the main program.
- The dynamic linker does not resolve these or relocate the main program.

https://www3.physnet.uni-hamburg.de/physnet/Tru64-Unix/HTML/APS31DTE/DOCU_002.HTM

readelf -r nmain_dyn.out

```
readelf -r nmain_dyn.out

Relocation section ’.rel.dyn’ at offset 0x3dc contains 8 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00001ed0  00000008 R_386_RELATIVE
00001ed4  00000008 R_386_RELATIVE
00001ff8  00000008 R_386_RELATIVE
00002004  00000008 R_386_RELATIVE
00001fec  00000206 R_386_GLOB_DAT    00000000   _ITM_deregisterTMClone
00001ff0  00000306 R_386_GLOB_DAT    00000000   __cxa_finalize@GLIBC_2.1.3
00001ff4  00000406 R_386_GLOB_DAT    00000000   __gmon_start__
00001ffc  00000606 R_386_GLOB_DAT    00000000   _ITM_registerTMCloneTa

Relocation section ’.rel.plt’ at offset 0x41c contains 2 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00001fe4  00000107 R_386_JUMP_SLOT   00000000   doAlmostNothing
00001fe8  00000507 R_386_JUMP_SLOT   00000000   __libc_start_main@GLIBC_2.0
```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou

readelf -SW nmain_dyn.out (1)

```
young@USys1:~$ readelf -SW nmain_dyn.out
There are 29 section headers, starting at offset 0x17a8:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        00000154 000154 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            00000168 000168 000020 00   A  0   0  4
  [ 3] .note.gnu.build-id NOTE            00000188 000188 000024 00   A  0   0  4
  [ 4] .gnu.hash         GNU_HASH        000001ac 0001ac 00003c 04   A  5   0  4
  [ 5] .dynsym           DYNSYM          000001e8 0001e8 0000d0 10   A  6   1  4
  [ 6] .dynstr           STRTAB          000002b8 0002b8 0000da 00   A  0   0  1
  [ 7] .gnu.version      VERSYM          00000392 000392 00001a 02   A  5   0  2
  [ 8] .gnu.version_r    VERNEED         000003ac 0003ac 000030 00   A  6   1  4
  [ 9] .rel.dyn          REL             000003dc 0003dc 000040 08   A  5   0  4
  [10] .rel.plt          REL             0000041c 00041c 000010 08  AI  5  22  4
  [11] .init             PROGBITS        0000042c 00042c 000023 00  AX  0   0  4
  [12] .plt              PROGBITS        00000450 000450 000030 04  AX  0   0 16
  [13] .plt.got          PROGBITS        00000480 000480 000010 08  AX  0   0  8
  [14] .text             PROGBITS        00000490 000490 0001d2 00  AX  0   0 16
  [15] .fini             PROGBITS        00000664 000664 000014 00  AX  0   0  4
  [16] .rodata           PROGBITS        00000678 000678 000008 00   A  0   0  4
```

readelf -SW nmain_dyn.out (2)

```
  [17] .eh_frame_hdr     PROGBITS        00000680 000680 00003c 00   A  0   0  4
  [18] .eh_frame         PROGBITS        000006bc 0006bc 0000fc 00   A  0   0  4
  [19] .init_array       INIT_ARRAY      00001ed0 000ed0 000004 04  WA  0   0  4
  [20] .fini_array       FINI_ARRAY      00001ed4 000ed4 000004 04  WA  0   0  4
  [21] .dynamic          DYNAMIC         00001ed8 000ed8 000100 08  WA  6   0  4
  [22] .got              PROGBITS        00001fd8 000fd8 000028 04  WA  0   0  4
  [23] .data             PROGBITS        00002000 001000 000008 00  WA  0   0  4
  [24] .bss              NOBITS          00002008 001008 000004 00  WA  0   0  1
  [25] .comment          PROGBITS        00000000 001008 00002a 01  MS  0   0  1
  [26] .symtab           SYMTAB          00000000 001034 000430 10     27  43  4
  [27] .strtab           STRTAB          00000000 001464 000248 00      0   0  1
  [28] .shstrtab         STRTAB          00000000 0016ac 0000fc 00      0   0  1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  p (processor specific)
```

dynamic section

- `readelf -d, --dynamic`

  displays the contents of the file's dynamic section, if it has one.

- contains information that the dynamic linker uses to bind procedure addresses
  - the location of symbol table

    ```
    0x00000006 (SYMTAB) 0x1e8
    ```

  - the location of relocation information

    ```
    0x00000011 (REL) 0x3dc
    ```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou

readelf -d nmain_dyn.out (1)

```
young@USys1:~$ readelf -d nmain_dyn.out

Dynamic section at offset 0xed8 contains 28 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [./libnothing.so]
 0x00000001 (NEEDED)                     Shared library: [libc.so.6]
 0x0000000c (INIT)                       0x42c
 0x0000000d (FINI)                       0x664
 0x00000019 (INIT_ARRAY)                 0x1ed0
 0x0000001b (INIT_ARRAYSZ)               4 (bytes)
 0x0000001a (FINI_ARRAY)                 0x1ed4
 0x0000001c (FINI_ARRAYSZ)               4 (bytes)
 0x6ffffef5 (GNU_HASH)                   0x1ac
 0x00000005 (STRTAB)                     0x2b8
 0x00000006 (SYMTAB)                     0x1e8
 0x0000000a (STRSZ)                      218 (bytes)
 0x0000000b (SYMENT)                     16 (bytes)
 0x00000015 (DEBUG)                      0x0
 0x00000003 (PLTGOT)                     0x1fd8
```

readelf -d nmain_dyn.out (2)

```
0x00000002 (PLTRELSZ)                      16 (bytes)
0x00000014 (PLTREL)                        REL
0x00000017 (JMPREL)                        0x41c
0x00000011 (REL)                           0x3dc
0x00000012 (RELSZ)                         64 (bytes)
0x00000013 (RELENT)                        8 (bytes)
0x0000001e (FLAGS)                         BIND_NOW
0x6ffffffb (FLAGS_1)                       Flags: NOW PIE
0x6ffffffe (VERNEED)                       0x3ac
0x6fffffff (VERNEEDNUM)                    1
0x6ffffff0 (VERSYM)                        0x392
0x6ffffffa (RELCOUNT)                      4
0x00000000 (NULL)                          0x0
```

finding pid

-
  ```
  gcc -g options
  gdb nmain_dyn.out
  ```

-
  ```
  (gdb) info program
        Using the running image of child process 4528.
  Program stopped at 0xf7fcc48f.
  It stopped at breakpoint 1.
  Type "info stack" or "info registers" for more information.
  ```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou

memory map on 32-bit Mint (1)

- [main]...
  ```
  00400000-00401000 r-xp 00000000 08:01 919273     /home/young/nmain_dyn/nmain_d
  00401000-00402000 r--p 00000000 08:01 919273     /home/young/nmain_dyn/nmain_d
  00402000-00403000 rw-p 00001000 08:01 919273     /home/young/nmain_dyn/nmain_d
  ```
- [shared library]...
  ```
  b7fce000-b7fcf000 r-xp 00000000 08:01 919472     /home/young/nmain_dyn/libnoth
  b7fcf000-b7fd0000 r--p 00000000 08:01 919472     /home/young/nmain_dyn/libnoth
  b7fd0000-b7fd1000 rw-p 00001000 08:01 919472     /home/young/nmain_dyn/libnoth
  ```
- [dynamic linker]...
  ```
  b7fd8000-b7ffe000 r-xp 00000000 08:01 526155     /lib/i386-linux-gnu/ld-2.27.s
  b7ffe000-b7fff000 r--p 00025000 08:01 526155     /lib/i386-linux-gnu/ld-2.27.s
  b7fff000-b8000000 rw-p 00026000 08:01 526155     /lib/i386-linux-gnu/ld-2.27.s
  ```
- [stack]..
  ```
  bffdf000-c0000000 rw-p 00000000 00:00 0          [stack]
  ```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou

memory map on 32-bit Mint (2)

- (gdb) shell cat /proc/29145/maps

```
00400000-00401000 r-xp 00000000 08:01 919273     /home/young/nmain_dyn/nmain_d
00401000-00402000 r--p 00000000 08:01 919273     /home/young/nmain_dyn/nmain_d
00402000-00403000 rw-p 00001000 08:01 919273     /home/young/nmain_dyn/nmain_d
b7dd8000-b7fad000 r-xp 00000000 08:01 526183     /lib/i386-linux-gnu/libc-2.27
b7fad000-b7fae000 ---p 001d5000 08:01 526183     /lib/i386-linux-gnu/libc-2.27
b7fae000-b7fb0000 r--p 001d5000 08:01 526183     /lib/i386-linux-gnu/libc-2.27
b7fb0000-b7fb1000 rw-p 001d7000 08:01 526183     /lib/i386-linux-gnu/libc-2.27
b7fb1000-b7fb4000 rw-p 00000000 00:00 0
b7fce000-b7fcf000 r-xp 00000000 08:01 919472     /home/young/nmain_dyn/libnoth
b7fcf000-b7fd0000 r--p 00000000 08:01 919472     /home/young/nmain_dyn/libnoth
b7fd0000-b7fd1000 rw-p 00001000 08:01 919472     /home/young/nmain_dyn/libnoth
b7fd1000-b7fd3000 rw-p 00000000 00:00 0
b7fd3000-b7fd6000 r--p 00000000 00:00 0          [vvar]
b7fd6000-b7fd8000 r-xp 00000000 00:00 0          [vdso]
b7fd8000-b7ffe000 r-xp 00000000 08:01 526155     /lib/i386-linux-gnu/ld-2.27.s
b7ffe000-b7fff000 r--p 00025000 08:01 526155     /lib/i386-linux-gnu/ld-2.27.s
b7fff000-b8000000 rw-p 00026000 08:01 526155     /lib/i386-linux-gnu/ld-2.27.s
bffdf000-c0000000 rw-p 00000000 00:00 0          [stack]
```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou

memory map on 64-bit Mint (1)

-
  ```
  [main].........................................................
  56555000-56556000 r-xp 00000000 08:01 142361   /home/young/nmain_dyn.out
  56556000-56557000 r--p 00000000 08:01 142361   /home/young/nmain_dyn.out
  56557000-56558000 rw-p 00001000 08:01 142361   /home/young/nmain_dyn.out
  ```

-
  ```
  [shared library]...............................................
  f7fcc000-f7fcd000 r-xp 00000000 08:01 142429   /home/young/libnothing.so
  f7fcd000-f7fce000 r--p 00000000 08:01 142429   /home/young/libnothing.so
  f7fce000-f7fcf000 rw-p 00001000 08:01 142429   /home/young/libnothing.so
  ```

-
  ```
  [dynamic linker]........................................
  f7fd6000-f7ffc000 r-xp 00000000 08:01 3280770  /lib32/ld-2.27.so
  f7ffc000-f7ffd000 r--p 00025000 08:01 3280770  /lib32/ld-2.27.so
  f7ffd000-f7ffe000 rw-p 00026000 08:01 3280770  /lib32/ld-2.27.so
  ```

-
  ```
  fffdd000-ffffe000 rw-p 00000000 00:00 0        [stack]
  ```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou

memory map on 64-bit Mint (2)

-
  ```
  (gdb) shell cat /proc/4528/maps
  56555000-56556000 r-xp 00000000 08:01 142361   /home/young/nmain_dyn.out
  56556000-56557000 r--p 00000000 08:01 142361   /home/young/nmain_dyn.out
  56557000-56558000 rw-p 00001000 08:01 142361   /home/young/nmain_dyn.out
  f7dd7000-f7fa9000 r-xp 00000000 08:01 3280774  /lib32/libc-2.27.so
  f7fa9000-f7faa000 ---p 001d2000 08:01 3280774  /lib32/libc-2.27.so
  f7faa000-f7fac000 r--p 001d2000 08:01 3280774  /lib32/libc-2.27.so
  f7fac000-f7fad000 rw-p 001d4000 08:01 3280774  /lib32/libc-2.27.so
  f7fad000-f7fb0000 rw-p 00000000 00:00 0
  f7fcc000-f7fcd000 r-xp 00000000 08:01 142429   /home/young/libnothing.so
  f7fcd000-f7fce000 r--p 00000000 08:01 142429   /home/young/libnothing.so
  f7fce000-f7fcf000 rw-p 00001000 08:01 142429   /home/young/libnothing.so
  f7fcf000-f7fd1000 rw-p 00000000 00:00 0
  f7fd1000-f7fd4000 r--p 00000000 00:00 0        [vvar]
  f7fd4000-f7fd6000 r-xp 00000000 00:00 0        [vdso]
  f7fd6000-f7ffc000 r-xp 00000000 08:01 3280770  /lib32/ld-2.27.so
  f7ffc000-f7ffd000 r--p 00025000 08:01 3280770  /lib32/ld-2.27.so
  f7ffd000-f7ffe000 rw-p 00026000 08:01 3280770  /lib32/ld-2.27.so
  fffdd000-ffffe000 rw-p 00000000 00:00 0        [stack]
  (gdb)
  ```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou

plt (1)

```
(gdb) break main
Breakpoint 1 at 0x5e6: file nmain.c, line 6.
(gdb) run
Starting program: /home/young/nmain_dyn/nmain_dyn.out

Breakpoint 1, main () at nmain.c:6
6           doAlmostNothing();
```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou

plt (2)

```
(gdb) disas
Dump of assembler code for function main:
   0x004005cd <+0>:     lea    0x4(%esp),%ecx
   0x004005d1 <+4>:     and    $0xfffffff0,%esp
   0x004005d4 <+7>:     pushl  -0x4(%ecx)
   0x004005d7 <+10>:    push   %ebp
   0x004005d8 <+11>:    mov    %esp,%ebp
   0x004005da <+13>:    push   %ebx
   0x004005db <+14>:    push   %ecx
   0x004005dc <+15>:    call   0x4005f9 <__x86.get_pc_thunk.ax>
   0x004005e1 <+20>:    add    $0x19f7,%eax
=> 0x004005e6 <+25>:    mov    %eax,%ebx
   0x004005e8 <+27>:    call   0x400460 <doAlmostNothing@plt>
   0x004005ed <+32>:    mov    $0x0,%eax
   0x004005f2 <+37>:    pop    %ecx
   0x004005f3 <+38>:    pop    %ebx
   0x004005f4 <+39>:    pop    %ebp
   0x004005f5 <+40>:    lea    -0x4(%ecx),%esp
   0x004005f8 <+43>:    ret
End of assembler dump.
```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou

plt (3)

```
(gdb) x /16xw 0x400460
0x400460 <doAlmostNothing@plt>:   0x000ca3ff  0x00680000  0xe9000000  0xffffffe0
0x400470 <__libc_start_main@plt>: 0x0010a3ff  0x08680000  0xe9000000  0xffffffd0
0x400480 <__cxa_finalize@plt>:    0x0018a3ff  0x90660000  0x001ca3ff  0x90660000
0x400490 <_start>:                0x895eed31  0xf0e483e1  0xe8525450  0x00000022
```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou

plt (4)

```
(gdb) x /16xw 0x400460
0x400460 <doAlmostNothing@plt>:   0x000ca3ff  0x00680000  0xe9000000  0xffffffe0
0x400470 <__libc_start_main@plt>: 0x0010a3ff  0x08680000  0xe9000000  0xffffffd0
0x400480 <__cxa_finalize@plt>:    0x0018a3ff  0x90660000  0x001ca3ff  0x90660000
0x400490 <_start>:                0x895eed31  0xf0e483e1  0xe8525450  0x00000022
```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou

got (1)

```
(gdb) si
0x004005e8      6           doAlmostNothing();
(gdb) list
1       #include "nothing.h"
2
3
4       int main(void)
5       {
6           doAlmostNothing();
7           return 0;
8       }
(gdb) si
0x00400460 in doAlmostNothing@plt ()
(gdb) si
doAlmostNothing () at nothing.c:10
10      {
(gdb) print /x $ebx
$1 = 0x401fd8
(gdb) x /16xw 0x401fd8
0x401fd8:       0x00001ed8      0x00000000      0x00000000      0xb7fce49d
0x401fe8:       0xb7deed90      0x00000000      0xb7e066b0      0x00000000
0x401ff8:       0x004005cd      0x00000000      0x00000000      0x00402004
0x402008 <completed.7281>:      0x00000000      0x00000000      0x00000000      0x0
```

got (2)

```
(gdb) x 0xb7fce49d
0xb7fce49d <doAlmostNothing>:   0x53e58955
(gdb) list
5       void doNothing()
6       {
7       }
8
9       void doAlmostNothing()
10      {
11          doNothingStatic();
12          doNothing();
13      }
```

https://stackoverflow.com/questions/19593883/understanding-the-relocation-table-ou