Link 5.A Relocation

Young W. Lim

2019-03-20 Wed

Outline

Based on

"Self-service Linux: Mastering the Art of Problem Determination",
Mark Wilding
"Computer Architecture: A Programmer's Perspective",
Bryant & O'Hallaron

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

TOC: Relocation

Symbol and Symbol Reference Candidates (1)

```
// main.c ---------------------

void swap();

// symbol buf
int buf[2]  = {1, 2};

int main()
{
  swap();   // symbol ref

  return 0;
}
```

- symbols :
  buf, p0, p1, swap()
- symbol references :
  rd/wr buf, p0, p1
  call swap()

```
// swap.c ---------------------

extern int buf[];

// symbols p0, p1
int *p0 = &buf[0];  // sym ref buf
int *p1;

void swap()     // symbol swap
{
  int tmp;

  p1 = &buf[1]; // sym ref p1, buf

  tmp = *p0;    // sym ref p0
  *p0 = *p1;    // sym ref p0, p1
  *p1 = tmp;    // sym ref p1

}
```

Symbol and Symbol Reference Candidates (2)

symbol candidates	symbol reference candidates
function definitions	function calls
global variable definitions	access of global variables
void swap() { . . . }	swap();
int buf[2];	p0 = buf+0;
int *p0;	p1 = buf+1;
int *p1;	tmp = *p0;
	*p0 = *p1;
	*p1 = tmp;

Types of Symbols Summary

	Global Variables	Functions
static	local linker Symbols	local linker symbols
non-static	global linker symbols	global linker symbols
external	global linker symbols	global linker symbols

Global Linker Symbols	Local Linker Symbols
non-static global variables	static global variables
non-static (global) functions	static (global) functions
external global variables	static local variables
external functions	

Relocation

- after symbol resolution
  - each *symbol reference* in the code has
    - exactly only one *symbol definition*
    - one *entry* in the *symbol table*
      which belongs to one of input object modules
  - the linker knows
    - the exact sizes of the *code* sections
    - the exact sizes of the *data* sections
- the relocation process
  - merges the input modules
  - assigns *run-time address* to each symbol and symbol reference

1. Relocating sections and symbol definitions

- the linker merges all sections of the same type
  into a new *aggregate* section of the same type
  - `.data` sections from the input modules are all merged
  - into a new `.data` section for the output executable object file
- the linker assigns *run-time memory addresses*
  - to the new *aggregated* sections (a set of sections)
  - to each *section* defined by the input modules
  - to each *symbol* defined by the input modules
- finally, every *instruction* and *global variable* has
  a unique *run-time memory addresses*

2. Relocating symbol references within sections

- the linker modifies every *symbol reference*
  in the bodies of the *code* and *data* sections
  so that they point to the correct *run-time addresses*

- relocation entries :
  the linker relies on data structures
  in the relocatable object modules
  - offset : the offset of a "symbol reference"
  - symbol : the symbol that a "symbol reference" refers to
  - type : relationcation types (ELF has 11 relocation types)

TOC: Relocation Entries

Relocation Entry (1)

- Whenever the assembler encounters a symbol reference
  to an object whose ultimate address is not known,
  the assembler generates a *relocation entry*
  (offset, symbol, type)

- Using relocation entries,
  the linker can get information about how to modify the reference
  when the linker merges the object files into an executable

- `.relo.text` : relocation entries for code
- `.relo.data` : relocation entries for initialized data

Relocation Entry (2)

Relocation Entry

- `offset` : the section offset of the reference which will be *modified*
- `symbol` : the symbol that the modified reference should point to
- `type` : tells the linker how to modify the reference

```
typedef struct {
   int offset;     // offset of the reference to relocate symbol
   int symbol:24,  // the reference should point to
       type:8;     // relocation type
} Elf32_Rel;
```

Relocation Entry Example (1)

- `offset` : `p1 = &buf[ 1]` where the symbol `p1` is referenced
- `symbol` : `int *p1` : where the symbol `p1` is defined
  `symbol value` : the address of a symbol
- `type` : `R_386_GOT32X` : how the symbol address is calculated at the symbol reference

Relocation Entry

- `offset` : the section offset of the reference which will be *modified*
- `symbol` : the symbol that the modified reference should point to
- `type` : tells the linker how to modify the reference

Relocation Entry Example (2)

- `12: R_386_GOT32X  p1`
  - `offset` : `12`
  - `symbol name` : `p1`
  - `type` : `R_386_GOT32X`

- the symbol value of `p1` is calculated by the `R_386_GOT32x` method and will be stored at this symbol reference with the offset 12

Relocation Entry

- `offset` : the section offset of the reference which will be *modified*
- `symbol` : the symbol that the modified reference should point to
- `type` : tells the linker how to modify the reference

Symbol value

- In relocatable files,
  - symbol value holds alignment constraints
    for a symbol whose section index is COMMON
  - symbol value holds a section offset
    for a defined symbol
- In executable and shared object files,
  - symbol value holds a virtual address

https://docs.oracle.com/cd/E19683-01/816-7529/6mdhf5r3j/index.html#chapter6-351

Symbol value in relocatable files

- symbol value holds alignment constraints
  for a symbol whose section index is COMMON
  (The symbol labels a common block
  that has not yet been allocated)

- symbol value holds a section offset for a defined symbol.
  symbol value is an offset from the beginning of the section
  that st_shndx identifies.

https://docs.oracle.com/cd/E19683-01/816-7529/6mdhf5r3j/index.html#chapter6-351

Symbol value in executable and shared object files

- symbol value holds a virtual address.
  To make these files' symbols more useful for the runtime linker,
  the section offset (file interpretation) gives way
  to a virtual address (memory interpretation)
  for which the section number is irrelevant.

https://docs.oracle.com/cd/E19683-01/816-7529/6mdhf5r3j/index.html#chapter6-351

Relocation type list

name	value	field	calculation
R_386_NONE	0	None	None
R_386_32	1	word32	S+A
R_386_PC32	2	word32	S+A-P
R_386_GOT32	3	word32	G+A
R_386_PLT32	4	word32	L+A-P
R_386_COPY	5	None	None
R_386_GLOB_DAT	6	word32	S
R_386_JMP_SLOT	7	word32	S
R_386_RELATIVE	8	word32	B+A
R_386_GOTOFF	9	word32	S+A-GOT
R_386_GOTPC	10	word32	GOT+A-P
R_386_32PLT	11	word32	L+A

Basic Relocation Types

- `type` : tells the linker how to modify the new reference
  - 11 different relocation types in ELF
  - `R_386_PC32` (PC Relative Reference Relocation)
  - `R_386_32` (Absolute Reference Relocation)

Basic Relocation Types : R_386_PC32

- relocating a reference that uses a 32-bit PC-relative address
  - a PC-relative address :
    an offset from the current run-time value of the PC
  - the program coutner (PC) :
    holds the address of the next instruction to be executed

- the *effective address* :
  - an address such as the target of the `call` instruction
  - obtained by adding 32-bit value to the current value of the PC
    encoded in the instruction

Basic Relocation Types : R_386_32

- relocating a reference that uses a 32-bit absolute address
- the *effective address* :
  the 32-bit value encoded in the instruction

TOC: Relocation Table

ELF Relocation Entry Types - Elf32_Rel, Elf32_Rela

Elf32_Rel

```
typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
} Elf32_Rel
```

Elf32_Rela

```
typedef struct {
        Elf32_Addr      r_offset;
        Elf32_Word      r_info;
        Elf32_Sword     r_addend;
} Elf32_Rela
```

http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf

ELF Relocation Entry Members - r_offset

- the location at which to apply the relocation action
- symbol reference location
- for a relocatable file,
  - the offset value is the byte offset
    from the beginning of the section
    to the storage unit affected by the relocation
- for an executable or a shared object file,
  - the offset value is the virtual address of
    the storage unit affected by the relocation

http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf

ELF Relocation Entry Members - r_info

1. the symbol table index
   with respect to which the relocation must be made
   - application result of ELF32_R_SYM to r_info member
2. the relocation type to be applied
   - application result of ELF32_R_TYPE to r_info member

ELF32_R_TYPE, ELF32_R_SYM

```
#define ELF32_R_SYM(i)     ((i)>>8)
#define ELF32_R_TYPE(i)    ((unsigned char) (i))
#define ELF32_R_INFO(s,t)  (((s)<<8) + (unsigned char)(t))
```

http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf

ELF Relocation Entry Members - r_addend

- specifies a constant addend used to compute
  - the value to be stored into the relocation field
  - the symbol value to the symbol reference location
- only Elf32_Rela entries contain an explicit addend
- Elf32_Rel entries store an implicit addend
  in the location to be modified

http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf

ELF Relocation Section

- starts with a table of relocation entries
  which can be located using the relevant section header
- the section header
  - when `sh_type` is either `SHT_REL` or `SHT_RELA`
  - `sh_link` : the section header index of
    the associated symbol table
  - `sh_info` : the section header index of
    the section to which the relocation applies
- a relocation section references *two other sections*
  - a symbol table section
  - a section to modify a symbol reference

`http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf`

ELF Sections

- an object file's section header table lets one locate all the file's sections
- an array of `Elf32_Shdr` structures
- a section header table index is a subscript into this array
- ELF header members related to the section header table
  - `e_shoff` byte offset from the beginning of the file to the section header table
  - `e_shnum` : the number of entries the section header table contains
  - `e_shentsize` : the size in bytes of each entry

`http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf`

ELF Section Conditions (1)

- Every section in an object file has exactly
  one section header describing it
- Section headers may exist
  which do not have a section
- Each section occupies one contiguous (possibly empty)
  sequence of bytes within a file

`http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf`

ELF Section Conditions (2)

- Sections in a file may not overlap
  No byte in a file resides in more than one section
- An object file may have inactive space
  The various headers and the sections might not cover every byte in an object file
  The contents of the inactive data are unspecified

`http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf`

ELF Section Header Structure

ELF Section Header Structure : Elf32_Shdr

```
typedef struct {
        Elf32_Word    sh_name;
        Elf32_Word    sh_type;
        Elf32_Word    sh_flags;
        Elf32_Addr    sh_addr;
        Elf32_Off     sh_offset;
        Elf32_Word    sh_size;
        Elf32_Word    sh_link;
        Elf32_Word    sh_addralign;
        Elf32_Word    sh_entsize;
} Elf32_Shdr;
```

http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf

ELF Relocation Entries for relocatable object files

relocatable files

- `r_offset` holds a section offset
- the relocation section itself describes
  how to modify another section in the file
  - relocation entries in a relocation table
- the relocation offsets designate a storage unit
  within the second section (symbol reference)

`http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf`

ELF Relocation Entries for executable and shared object files

executable and shared object files

- `r_offset` holds a virtual address
- to make these files' relocation entries more useful for dynmic linker
- the section offset (file interpretation) gives way to a virtul address (memory interpretation)

`http://www.cs.cmu.edu/afs/cs/academic/class/15213-s01/s00/doc/elf.pdf`

Columns of readelf -r (1)

readelf -r /bin/ls | head -n 20

```
Relocation section '.rela.dyn' at offset 0x15b8 contains 7 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000619ff0  003e00000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0

Relocation section '.rela.plt' at offset 0x1660 contains 105 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
00000061a018  000100000007 R_X86_64_JUMP_SLO 0000000000000000 __ctype_toupper_loc +
```

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

Columns of `readelf -r` (2)

- `readelf -r /bin/ls | head -n 20`

Offset	000000619ff0
Info	003e00000006
Type	R_X86_64_GLOB_DAT
Sym. Value	0000000000000000
Sym. Name + Addend	__gmon_start__ + 0

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

Relocation Table

Offset	where the symbol value should go
Info	- the type (the exact calculation depends on the arch) - the symbol index in the symtab
Type	relocation type of the symbol according to the ABI
Sym value	the addend to be added to the symbol resolution
Sym name Addend	a pretty printing of the symbol name + addend.

```
Relocation section '.rela.dyn' at offset 0x15b8 contains 7 entries:
  Offset          Info           Type           Sym. Value    Sym. Name + Addend
000000619ff0  003e00000006 R_X86_64_GLOB_DAT 0000000000000000 __gmon_start__ + 0
```

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

Relocation Table Example

readelf -r swap.o

```
Relocation section '.rel.text' at offset 0x478 contains 8 entries:
 Offset     Info    Type            Sym.Value  Sym. Name
00000007  00001402 R_386_PC32        00000000   __x86.get_pc_thunk.ax
0000000c  0000150a R_386_GOTPC       00000000   _GLOBAL_OFFSET_TABLE_
00000012  0000122b R_386_GOT32X      00000004   p1
00000018  0000112b R_386_GOT32X      00000000   buf
00000023  00001009 R_386_GOTOFF      00000000   p0
0000002e  0000122b R_386_GOT32X      00000004   p1
00000036  00001009 R_386_GOTOFF      00000000   p0
00000040  0000122b R_386_GOT32X      00000004   p1
```

Relocation Example other Reference

- https://wiki.osdev.org/ELF_Tutorial

TOC: Relocation Algorithm

Pseudocode

```
foreach section s {
  foreach relocation entry r {
    refp = s + r.offset; // p to references to be relocated

    // relocate a PC-relative references
    if (r.type == R_386_PC32) {
      refaddr = ADDR(s) + r.offset;
      *refp = (unsigned) (ADDR(r.symbol) + *refp - refaddr;
    }

    // relocate an absolute reference
    if (r.type == R_386_3) {
      *refp = (unsigned) (ADDR(r.symbol) + *refp);
    }
  }
}
```

section s and symbol reference r

- `s` : one of sections
- `r` : an entry of the relocation table in a section
  - a symbol reference whose ultimate address is unknown,
  - an entry for each of such symbol references

- each section `s`
  can be considered as an *array* of bytes
- each relocation entry `r`
  is a *struct* of type `Elf32_Rel` (offset, symbol, type)

relocation entry : r.offset, r.type, r.symbol

- for each reference `r`
  - `r.offset` : the section offset of the reference to be modified
  - `r.symbol` : the symbol to which the reference should point
  - `r.type` : how to modify for a new reference

- the `buf` symbol in `swap.c`
  - `buf` is defined in `main.c`
  - 4 instances in `swap.c`
  - each of these instances is a reference to the symbol `buf`
  - each of these references has its own offset within the .text section

run-time address ADDR(s) and ADDR(r.symbol)

- each symbol reference (r) refers to its corresponding symbol (r.symbol)
- assume that when the relocation algorithm runs,
  the linker has already chosen
  - run-time addresses for each section (`ADDR(s)`)
  - run-time addresses for each symbol (`ADDR(r.symbol)`)

1. section address : `ADDR(s)`
2. symbol address : `ADDR(r.symbol)`

section address and symbol address

section address

ADDR(s) : : v	.text
r.offset+ ADDR(s)	reference r

symbol address

	.data
ADDR(sym)	symbol sym

- `r` $\rightarrow$ `sym` (`r` *points to* `sym`)
- `ADDR(r.symbol)` == `ADDR(sym)`

reference address (`refp` and `refaddr`)

- a *symbol reference* points to its associated *symbol definition* location
- a particular symbol reference location before relocation
  can be represented as a pointer variable (`refp`)
  - the value of a pointer variable (`refp`) is
    the address of a particular symbol reference (*reference address*)
  - the data at the reference location is (`*refp`)
- that particular symbol reference location after relocation
  - is `refaddr` is the run-time address of `refp`
  - uses the run-time address of ADDR(s)

reference data (*refp)

- the data at a particular symbol reference location
  can be represented as a pointer variable (*refp)

- ~*refp* before relocation
  - the initial *refp can be
    - a PC conpensation value
    - array index offset value
- ~*refp* after relocation
  - the final *rep is
    the run-time address
    of the pointed symbol (r.symbol) location

refp and refaddr

- a symbol reference address shows
  where is the particular reference to a symbol made

1. `refp = s + r.offset`
2. `refaddr = ADDR(s) + r.offset`

- `refp` : the reference address before relocation
  - pointer (address) to a particular symbol reference to be relocated
  - the address of a particular symbol reference before relocation
- `refaddr` : the reference address after relocation
  - the run-time address of a particular symbol reference

relocation of a pc-relative reference

- initial `*refp` : initial reference data
  - PC compensation value : e.g. (-4)
- final `*refp` : final reference data (the corresponding symbol address)
  - `ADDR(r.symbol) - refaddr` :
    the relative symbol address w.r.t. the reference address
  - `(unsigned) (ADDR(r.symbol) + *refp - refaddr)`
    the relative symbol address with the PC value compensated

relocation of an absolute reference

- initial `*refp` : initial reference data
  - array index offset value : e.g. buf[1]
- final `*refp` : final reference data (the corresponding symbol address)
  - `(unsigned) (ADDR(r.symbol) + *refp)`
    the absolute symbol address with an array index offset

TOC: Reloation Examples

Example codes

```
// main.c ---------------------

void swap();

int buf[2]  = {1, 2};

int main()
{
  swap();

  return 0;
}
```

```
// swap.c ---------------------

extern int buf[];

int *p0 = &buf[0];
int *p1;

void swap()
{
  int tmp;

  p1 = &buf[1];

  tmp = *p0;
  *p0 = *p1;
  *p1 = tmp;

}
```

Compile commands

- default compiling

```
gcc -m32 -Wall -c swap.c
gcc -m32 -Wall -c smain.c
gcc -m32 -o smain.out smain.o swap.o
```

- do not use the system startup files or libraries

```
gcc -m32 -Wall -g -O0 -c swap.c
gcc -m32 -Wall -g -O0 -c smain.c
gcc -m32 -nostdlib -o smain.out smain.o swap.o
```

Analyzing commands

- disassembly

```
objdump -d smain.o
objdump -d swap.o
objdump -d smain.out
```

- symbol table section

```
objdump -t smain.o
objdump -t swap.o
objdump -s smain.out
```

objdump -d main.o (disassemble)

- call 12 : offset=12, symbol=swap, type=R_386_PC32

```
main.o:     formato del fichero elf32-i386


Desensamblado de la sección .text.startup:

00000000 <main>:
   0:   8d 4c 24 04             lea    0x4(%esp),%ecx
   4:   83 e4 f0                and    $0xfffffff0,%esp
   7:   ff 71 fc                pushl  -0x4(%ecx)
   a:   55                      push   %ebp
   b:   89 e5                   mov    %esp,%ebp
   d:   51                      push   %ecx
   e:   83 ec 04                sub    $0x4,%esp
  11:...e8.fc.ff.ff.ff.........call...12.<main+0x12>.....
  16:   83 c4 04                add    $0x4,%esp
  19:   31 c0                   xor    %eax,%eax
  1b:   59                      pop    %ecx
  1c:   5d                      pop    %ebp
  1d:   8d 61 fc                lea    -0x4(%ecx),%esp
  20:   c3                      ret
```

objdump -d swap.o (disassemble)

- mov 0x0, %eax : offset=0x1, symbol=p0
- mov 0x4, %ecx : offset=0x7, symbol=buf
- movl $0x4, 0x0 : offset=0xd, symbol=p1 ; offset=0x11, symbol=buf
- mov %edx, 0x4 : offset= 0x1b, symbol=buf

```
swap.o:     formato del fichero elf32-i386


Desensamblado de la sección .text:

00000000 <swap>:
   0:...a1.00.00.00.00.........mov....0x0,%eax........
   5:...8b.0d.04.00.00.00.......mov....0x4,%ecx........
   b:...c7.05.00.00.00.00.04....movl...$0x4,0x0........
  12:   00 00 00
  15:   8b 10                   mov    (%eax),%edx
  17:   89 08                   mov    %ecx,(%eax)
  19:...89.15.04.00.00.00.......mov....%edx,0x4........
  1f:   c3                      ret
```

readelf -x .data swap.o (.data section)

```
Hex dump of section '.data':
 NOTE: This section has relocations against it, but these have NOT been applied to
  0x00000000 00000000                                    ....
```

arrays and pointers in swap

.....	p1
.....	p0

.....	buf[1]
.....	buf[0]

0x804a028	p1 = 0x0804a01c
0x804a024	
0x804a020	p0 = 0x0804a018

0x804a01c	buf[1] = 2
0x804a018	buf[0] = 1

- Relocatable Object
  - symbol table
- Executable Object
  - run-time address

refp and *refp in relocatable main

- refp = 0x12, *refp = 0xfffffffc -4 PC compensation

```
00000000 <main>:
   0:   8d 4c 24 04             lea    0x4(%esp),%ecx
   4:   83 e4 f0                and    $0xfffffff0,%esp
   7:   ff 71 fc                pushl  -0x4(%ecx)
   a:   55                      push   %ebp
   b:   89 e5                   mov    %esp,%ebp
   d:   51                      push   %ecx
   e:   83 ec 04                sub    $0x4,%esp
  11:   e8 fc ff ff ff          call   12 <main+0x12>
  16:   83 c4 04                add    $0x4,%esp
  19:   31 c0                   xor    %eax,%eax
  1b:   59                      pop    %ecx
  1c:   5d                      pop    %ebp
  1d:   8d 61 fc                lea    -0x4(%ecx),%esp
  20:   c3                      ret
```

refp and *refp in relocatable swap

- refp= 0x01, *refp= 0x00000000
- refp= 0x07, *refp= 0x00000004 buf[1] array index offset
- refp= 0x0d, *refp= 0x00000000
- refp= 0x11, *refp= 0x00000004 buf[1] array index offset
- refp= 0x1b, *refp= 0x00000004 buf[1] array index offset

```
00000000 <swap>:
   0:   a1 00 00 00 00          mov    0x0,%eax     ;; %eax=p0
   5:   8b 0d 04 00 00 00       mov    0x4,%ecx     ;; %ecx=buf[ 1 ]
   b:   c7 05 00 00 00 00 04    movl   $0x4,0x0     ;; p1 = buf + 1
  12:   00 00 00
  15:   8b 10                   mov    (%eax),%edx  ;; tmp=*p0
  17:   89 08                   mov    %ecx,(%eax)  ;; *p0=buf[1]
  19:   89 15 04 00 00 00       mov    %edx,0x4     ;; buf[1]=tmp
  1f:   c3                      ret
```

relocation result : main

- refp = 0x12, *refp = 0xfffffffc -> 0x08048400

```
00000000 <main>:
   0: lea    0x4(%esp),%ecx
   4: and    $0xfffffff0,%esp
   7: pushl  -0x4(%ecx)
   a: push   %ebp
   b: mov    %esp,%ebp
   d: push   %ecx
   e: sub    $0x4,%esp
  11: call   12 <main+0x12>
  16: add    $0x4,%esp
  19: xor    %eax,%eax
  1b: pop    %ecx
  1c: pop    %ebp
  1d: lea    -0x4(%ecx),%esp
  20: ret
```

```
080482e0 <main>:
 80482e0: lea    0x4(%esp),%ecx
 80482e4: and    $0xfffffff0,%esp
 80482e7: pushl  -0x4(%ecx)
 80482ea: push   %ebp
 80482eb: mov    %esp,%ebp
 80482ed: push   %ecx
 80482ee: sub    $0x4,%esp
 80482f1: call   8048400 <swap>
 80482f6: add    $0x4,%esp
 80482f9: xor    %eax,%eax
 80482fb: pop    %ecx
 80482fc: pop    %ebp
 80482fd: lea    -0x4(%ecx),%esp
 8048300: ret
```

relocation result : swap

- refp= 0x01, *refp= 0x00000000 -> 0x0804a020
- refp= 0x07, *refp= 0x00000004 -> 0x0804a01c
- refp= 0x0d, *refp= 0x00000000 -> 0x0804a028
- refp= 0x11, *refp= 0x00000004 -> 0x0804a01c
- refp= 0x1b, *refp= 0x00000004 -> 0x0804a01c

```
00000000 <swap>:
   0: mov    0x0,%eax
   5: mov    0x4,%ecx
   b: movl   $0x4,0x0
  12:
  15: mov    (%eax),%edx
  17: mov    %ecx,(%eax)
  19: mov    %edx,0x4
  1f: ret
```

```
08048400 <swap>:
 8048400: mov    0x804a020,%eax
 8048405: mov    0x804a01c,%ecx
 804840b: movl   $0x804a01c,0x804a028
 8048412:
 8048415: mov    (%eax),%edx
 8048417: mov    %ecx,(%eax)
 8048419: mov    %edx,0x804a01c
 804841f: ret
```

Call Instruction

- `call` Label : direct procedure call
- `call` *Operand : indirect procedure call

Imm	M[Imm]	Absolute
(Ea)	M[R[Ea]]	Indirect
Imm (Eb)	M[Imm + R[Eb]]	Base + displace
(Eb, Ei)	M[R[Eb]+R[Ei]]	Indexed
Imm (Eb, Ei)	M[Imm + R[Eb] + R[Ei]]	Indexed
(, Ei, s)	M[R[Ei]*s]	Scaled Indexed
Imm (, Ei, s)	M[Imm + R[Ei]*s]	Scaled Indexed
Eb, Ei, s)	M[R[Eb] + R[Ei]*s]	Scaled Indexed
m (Eb, Ei, s)	M[Imm + R[Eb] + R[Ei]*s]	Scaled Indexed

Relocation Examples : main.o relocation entry

1. objdump -t main.o (symbol table)
2. bojdump -r main.o (relocation entry)

objdump -t main.o (symbol table)

- main, swap, buf

```
main.o:     formato del fichero elf32-i386

SYMBOL TABLE:
00000000 l    df *ABS*           00000000 main.i
00000000 l    d  .text           00000000 .text
00000000 l    d  .data           00000000 .data
00000000 l    d  .bss            00000000 .bss
00000000 l    d  .text.unlikely  00000000 .text.unlikely
00000000 l    d  .text.startup   00000000 .text.startup
00000000 l    d  .note.GNU-stack 00000000 .note.GNU-stack
00000000 l    d  .eh_frame       00000000 .eh_frame
00000000 l    d  .comment        00000000 .comment
00000000 g     F .text.startup   00000021 main
00000000.........*UND*...........00000000 swap.............
00000000 g     O .data           00000008 buf
```

objdump -r main.o (relocation entry)

- swap, R_386_PC32, 00000012

```
main.o:     formato del fichero elf32-i386

RELOCATION RECORDS FOR [.text.startup]:
OFFSET   TYPE              VALUE
00000012.R_386_PC32........swap.....................


RELOCATION RECORDS FOR [.eh_frame]:
OFFSET   TYPE              VALUE
00000020 R_386_PC32        .text.startup
```

Relocation Examples : swap.o relocation entry

1. objdump -t swap.o (symbol table)
2. objdump -r swap.o (relcation entry)

objdump -t swap.o (symbol table)

- swap, p0, buf, p1

```
SYMBOL TABLE:
00000000 l    df *ABS*           00000000 swap.i
00000000 l    d  .text           00000000 .text
00000000 l    d  .data           00000000 .data
00000000 l    d  .bss            00000000 .bss
00000000 l    d  .text.unlikely  00000000 .text.unlikely
00000000 l    d  .note.GNU-stack 00000000 .note.GNU-stack
00000000 l    d  .eh_frame       00000000 .eh_frame
00000000 l    d  .comment        00000000 .comment
00000000.g.....F..text...........00000020.swap..............
00000000.g.....O..data...........00000004.p0................
00000000.........*UND*...........00000000.buf...............
00000004.......O.*COM*...........00000004.p1................
```

objdump -r swap.o (relocation entry)

- p0, p1, buf(3 references in .text and 1 in .data)

```
swap.o:     formato del fichero elf32-i386

RELOCATION RECORDS FOR [.text]:
OFFSET   TYPE              VALUE
00000001.R_386_32..........p0..............
00000007.R_386_32..........buf.............
0000000d.R_386_32..........p1..............
00000011.R_386_32..........buf.............
0000001b.R_386_32..........buf.............


RELOCATION RECORDS FOR [.data]:
OFFSET   TYPE              VALUE
00000000.R_386_32..........buf.............


RELOCATION RECORDS FOR [.eh_frame]:
OFFSET   TYPE              VALUE
00000020 R_386_PC32        .text
```

Relocation Examples : R_386_PC32 Reference Relocation

1. Relocating PC-Relative References

Relocating PC-Relative References (1)

- main in the .text of main.o calls swap in swap.o
- 11: e8 fc ff ff ff call 12 <main+0x12>

```
11:   e8 fc ff ff ff          call   12 <main+0x12>

11    e8 call
12    fc       <--- PC-4  : location of the reference to "swap" symbol
13    ff
14    ff
15    ff
16             <--- PC

      0xfffffffc = -4 (little endian)
      PC -4 = 16-4 = 12
```

Relocating PC-Relative References (2)

- `main` in the `.text` of `main.o` calls `swap` in `swap.o`
- `11: e8 fc ff ff ff  call  12 <main+0x12>`

- `offset=12`
  the section offset (12) of the reference which will be modified
- `symbol=swap`
  the modified reference at 12 should point to the symbol (`swap`)
- `type=R_386_PC32` :
  tells the linker how to modify the new reference

```
RELOCATION RECORDS FOR [.text.startup]:
OFFSET   TYPE              VALUE
00000012 R_386_PC32        swap
```

Relocating PC-Relative References (3)

- r.offset = 12
- r.symbol = swap
- r.type = R_386_PC32

Assume

```
ADDR(s)         = ADDR(.text) = 0x80482e0
ADDR(r.symbol) = ADDR(swap)  = 0x8048400
```

```
the run-time address of the reference

refaddr = ADDR(s) + r.offset
        = 0x80482e0 + 0x12
        = 0x80482f2
```

Relocating PC-Relative References (4)

- ADDR(s) = ADDR(.text) = 0x80482e0
- ADDR(r.symbol) = ADDR(swap) = 0x8048400

```
updates the reference value : (-4) -->

*refp = (unsigned) (ADDR(r.symbol) + *refp - refaddr)
      = (unsigned) (ADDR(r.symbol) - refaddr + *refp)
      = (unsigned) (0x8048400 - 0x80482f2 +(-4))
      = (unsigned) (0x10a)

                 11 12 13 14 15
                 :  :  :  :  :
       11:       e8 fc ff ff ff          call   12 <main+0x12>
       ||           || || || ||                 ||
       ||           || || || ||                 ||
       VV           VV VV VV VV                 VV
  80482f1:       e8 0a 01 00 00          call   8048400 <swap>
                 :  :  :  :  :
                 f1 f2 f3 f4 f5

                    04 + 4 = 8
```

Relocation Examples : R_386_32 Reference Relocation

1. Relocating Absolute References

Relocating Absolute References (1)

- main in the .text of main.o calls swap in swap.o
- 11: e8 fc ff ff ff call 12 <main+0x12>

```
11:   e8 fc ff ff ff          call   12 <main+0x12>                          11:   e8 fc f

11    e8 call                                                                11    e8 call
12    fc       <--- PC-4  : location of the reference to "swap" symbol       12    fc
13    ff                                                                     13    ff
14    ff                                                                     14    ff
15    ff                                                                     15    ff
16             <--- PC                                                      16

      0xfffffffc = -4 (little endian)                                              0xffffff
      PC -4 = 16-4 = 12                                                            PC -4 =
```

TOC: Results of Relocation

1. objdump -d p (disassemble) - main
2. objdump -d p (disassemble) - swap
3. objdump -d p (disassemble) - section summary
4. objdump -d p (symbol table)

objdump -d p (disassemble) - main

```
080482e0 <main>:
 80482e0:       8d 4c 24 04             lea    0x4(%esp),%ecx
 80482e4:       83 e4 f0                and    $0xfffffff0,%esp
 80482e7:       ff 71 fc                pushl  -0x4(%ecx)
 80482ea:       55                      push   %ebp
 80482eb:       89 e5                   mov    %esp,%ebp
 80482ed:       51                      push   %ecx
 80482ee:       83 ec 04                sub    $0x4,%esp
 80482f1:       e8 0a 01 00 00          call   8048400 <swap>
 80482f6:       83 c4 04                add    $0x4,%esp
 80482f9:       31 c0                   xor    %eax,%eax
 80482fb:       59                      pop    %ecx
 80482fc:       5d                      pop    %ebp
 80482fd:       8d 61 fc                lea    -0x4(%ecx),%esp
 8048300:       c3                      ret
```

objdump -d p (disassemble) - swap

```
08048400 <swap>:
 8048400:       a1 20 a0 04 08          mov    0x804a020,%eax
 8048405:       8b 0d 1c a0 04 08       mov    0x804a01c,%ecx
 804840b:       c7 05 28 a0 04 08 1c    movl   $0x804a01c,0x804a028
 8048412:       a0 04 08
 8048415:       8b 10                   mov    (%eax),%edx
 8048417:       89 08                   mov    %ecx,(%eax)
 8048419:       89 15 1c a0 04 08       mov    %edx,0x804a01c
 804841f:       c3                      ret
```

objdump -d p (disassemble) - section summary

```
./p:     formato del fichero elf32-i386
Desensamblado de la sección .init: ..........................
0804828c <_init>:
Desensamblado de la sección .plt: ...........................
080482b0 <__libc_start_main@plt-0x10>:
080482c0 <__libc_start_main@plt>:
Desensamblado de la sección .plt.got: .......................
080482d0 <.plt.got>:
Desensamblado de la sección .text: ..........................
080482e0 <main>:
08048301 <_start>:
08048330 <__x86.get_pc_thunk.bx>:
08048340 <deregister_tm_clones>:
08048370 <register_tm_clones>:
080483b0 <__do_global_dtors_aux>:
080483d0 <frame_dummy>:
08048400 <swap>:
08048420 <__libc_csu_init>:
08048480 <__libc_csu_fini>:
Desensamblado de la sección .fini: ..........................
08048484 <_fini>:
```

objdump -t p (symbol table) (1)

```
SYMBOL TABLE:
08048154 l    d  .interp        00000000   .interp
08048168 l    d  .note.ABI-tag  00000000   .note.ABI-tag
08048188 l    d  .note.gnu.build-id     00000000 .note.gnu.build-id
080481ac l    d  .gnu.hash      00000000   .gnu.hash
080481cc l    d  .dynsym        00000000   .dynsym
0804820c l    d  .dynstr        00000000   .dynstr
08048252 l    d  .gnu.version   00000000   .gnu.version
0804825c l    d  .gnu.version_r 00000000   .gnu.version_r
0804827c l    d  .rel.dyn       00000000   .rel.dyn
08048284 l    d  .rel.plt       00000000   .rel.plt
0804828c l    d  .init          00000000   .init
080482b0 l    d  .plt           00000000   .plt
080482d0 l    d  .plt.got       00000000   .plt.got
080482e0 l    d  .text          00000000   .text........................
08048484 l    d  .fini          00000000   .fini
08048498 l    d  .rodata        00000000   .rodata......................
080484a0 l    d  .eh_frame_hdr  00000000   .eh_frame_hdr
080484d4 l    d  .eh_frame      00000000   .eh_frame
08049f08 l    d  .init_array    00000000   .init_array
08049f0c l    d  .fini_array    00000000   .fini_array
08049f10 l    d  .jcr           00000000   .jcr
```

objdump -t p (symbol table) (2)

```
08049f14 l    d  .dynamic       00000000  .dynamic
08049ffc l    d  .got           00000000  .got
0804a000 l    d  .got.plt       00000000  .got.plt
0804a010 l    d  .data          00000000  .data.........................
0804a024 l    d  .bss           00000000  .bss..........................
00000000 l    d  .comment       00000000  .comment......................
00000000 l    df *ABS*          00000000  crtstuff.c
08049f10 l     O .jcr           00000000  __JCR_LIST__
08048340 l     F .text          00000000  deregister_tm_clones
08048370 l     F .text          00000000  register_tm_clones
080483b0 l     F .text          00000000  __do_global_dtors_aux
0804a024 l     O .bss           00000001  completed.7200
08049f0c l     O .fini_array    00000000  __do_global_dtors_aux_fini_array_entry
080483d0 l     F .text          00000000  frame_dummy
08049f08 l     O .init_array    00000000  __frame_dummy_init_array_entry
00000000 l    df *ABS*          00000000  main.i
00000000 l    df *ABS*          00000000  swap.i
00000000 l    df *ABS*          00000000  crtstuff.c
080485b0 l     O .eh_frame      00000000  __FRAME_END__
08049f10 l     O .jcr           00000000  __JCR_END__
00000000 l    df *ABS*          00000000
```

objdump -t p (symbol table) (3)

```
08049f0c l       .init_array    00000000  __init_array_end
08049f14 l     O .dynamic       00000000  _DYNAMIC
08049f08 l       .init_array    00000000  __init_array_start
080484a0 l       .eh_frame_hdr  00000000  __GNU_EH_FRAME_HDR
0804a000 l     O .got.plt       00000000  _GLOBAL_OFFSET_TABLE_
08048480 g     F .text          00000002  __libc_csu_fini
00000000  w      *UND*          00000000  _ITM_deregisterTMCloneTable
08048330 g     F .text          00000004  .hidden __x86.get_pc_thunk.bx
0804a010  w      .data          00000000  data_start
0804a024 g       .data          00000000  _edata
0804a020 g     O .data          00000004  p0..........................
08048484 g     F .fini          00000000  _fini
0804a010 g       .data          00000000  __data_start
00000000  w      *UND*          00000000  __gmon_start__
0804a014 g     O .data          00000000  .hidden __dso_handle
0804849c g     O .rodata        00000004  _IO_stdin_used
00000000       F *UND*          00000000  __libc_start_main@@GLIBC_2.0
08048420 g     F .text          0000005d  __libc_csu_init
0804a02c g       .bss           00000000  _end
08048301 g     F .text          00000000  _start
08048498 g     O .rodata        00000004  _fp_hw
```

objdump -t p (symbol table) (4)

```
0804a018 g     O .data          00000008  buf.........................
0804a024 g       .bss           00000000  __bss_start
080482e0 g     F .text          00000021  main........................
0804a028 g     O .bss           00000004  p1..........................
00000000  w      *UND*          00000000  _Jv_RegisterClasses
0804a024 g     O .data          00000000  .hidden __TMC_END__
00000000  w      *UND*          00000000  _ITM_registerTMCloneTable
08048400 g     F .text          00000020  swap........................
0804828c g     F .init          00000000  _init
```