ELF1 7C Design Cycles - ELF Study 1999

Young W. Lim

2020-03-17 Tue

Outline

1. Based on
2. Relocs Design
   - TOC
   - 1. Object files for shared libraries (PIC .o files)
   - 2. Object files for executables (non-PIC .o files)
   - 3. Shared library files
   - 4. Executable files
   - 5. Summary
3. Copy Relocs
   - TOC
   - .bss section
   - Copy relocs
   - Referencing external data using the GOT
   - Referencing external data by copying
4. Relative Reloc
   - TOC
   - Relative reloc `R_386_RELATIVE`

Based on

"Study of ELF loading and relocs", 1999
http://netwinder.osuosl.org/users/p/patb/public_html/elf_relocs.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`

TOC: Relocs Design

1. Object files for shared libraries (PIC `.o` files)
2. Object files for executables (non-PIC `.o` files)
3. Shared library files
4. Executable files
5. Summary

Relocs in Design cycles

1. .o files for executbles
   R_386_PC32, R_386_32
2. .o files for shared libraries

	local symbols	global symbols
code	R_386_GOTOFF	R_386_GOT32, R_386_PLT32
data	R_386_32	R_386_32
	by the section number	by the symbol name

3. executables
   R_386_COPY, R_386_JMP_SLOT
4. shared libraries
   R_386_RELATIVE, R_386_GLOB_DAT, R_386_JMP_SLOT

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

TOC : Object files for shared libraries (PIC .o files)

- Relocs in a PIC .o file
- Relocs in a PIC .o : local symbols
- Relocs in a PIC .o : global symbols
- Relocs in a PIC .o : a global symbol reference in the code
- Relocs in a PIC .o : a local symbol reference in the code
- Relocs in a PIC .o : a global symbol reference in the data
- Relocs in a PIC .o : a local symbol reference in the data

Relocs in a PIC .o file

- for a position independent code (PIC)
  - must use GOT / PLT
  - must distinguish
    - local and global objects
    - data and function objects (.data and .text)
- the relocs in the code and .rodata sections must use GOT based relocs, because
  - they are read-only and
  - cannot be modified at run time

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

Relocs in a PIC .o : local symbols

- static variables are allocated in .data or .bss

1. a local symbol reference, in the code section
   - R_386_GOTOFF : offset relative to &GOT[0]
   - actually, offset relative to .data
     (GOT is at the beginning of .data)
2. a local symbol reference, in the data section
   - R_386_32 : section-offset address
   - reference the symbol by the section number
     (.data, .bss)

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

Relocs in a PIC .o : global symbols

1. a global symbol reference, in the code section
   - R_386_GOT32 : offset to a entry in the GOT[k]

1. a global symbol reference, in the data section
   - R_386_32 : absolute address
   - reference the symbol by the symbol name

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

Relocs in a PIC .o : a global symbol reference in the *code*

- R_386_GOT32 (G+A)
  - create an entry in the GOT
  - the run-time system will *fill* the GOT entry with the symbol address
  - store the distance from GOT[0] to the related GOT entry
- R_386_PLT32 (L+A-P)
  - PC-relative calls to a PLT entry for a external function

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

Relocs in a PIC .o : a local symbol reference in the *code*

- R_386_GOTOFF (S+A-GOT)
  - relative distance from the GOT to the local symbol
  - can exist in the code (read-only) section,
    because it will be fully resolved at link time
    (the symbol address is known as the offset to GOT)
  - actually, this offset is relative to the .data section

- R_386_PC32 (S+A-P)
  - PC-relative calls to a local function

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

Relocs in a PIC .o : a global symbol reference in the *data*

- R_386_32 (S+A)
  - a reloc that references the symbol by name
  - absolute reference to the symbol
  - example :

```
R_ARM_32 Lextern ...... by the symbol name
R_ARM_32 .text ........ by the section number
```

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

Relocs in a PIC .o : a local symbol reference in the *data*

- R_386_32 (S+A)
  - when it can be *fixed* in memory with respect to a section, the object file is allowed to drop the symbol name replace it with a section plus offset expression
  - access by the section number not by the symbol name
- R_386_32 for a local symbol will be transformed into R_386_RELATIVE

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

TOC: 1. Object files for executables (non-PIC .o files)

- Relocs in .o files for executables

Relocs in .o files for executables

- relative reference to external symbols (R_386_PC32)
  - from here to a symbol
  - used for branches
- absolute reference to external symbols (R_386_32)

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

TOC: Shared library files

- Relocs in a `.so` : relocs for local symbols
- Relocs in a `.so` : transformed reloc for local symbols
- Relocs in a `.so` : PIC referencing of a local symbol in the data
- Relocs in a `.so` : PIC referencing of a global symbol
- Relocs in a `.so` : PIC referencing of a function symbol

Relocs in a .so - relocs for local symbols

- relocs in .o files for shared libraries
  - R_386_GOTOFF
    relocs for referencing a local symbol in the code
  - R_386_32
    relocs for referencing a local symbol in the data

```
http://netwinder.osuosl.org/users/p/patb/public_html/elf_relocs.html
Linkers and Loaders, J. R. Levine
```

Relocs in a `.so` - transformed reloc for local symbols

- these relocs for local symbols have offset to a given section
  - `R_386_GOTOFF` has an offset to the `.data` (`&GOT[0]`)
    - will fully resolved at the link time
  - `R_386_32` has an offset to a section (`.data`, `.bss`, `.text`)
    - will be transformed to `R_386_RELATIVE`
- `R_386_RELATIVE`
  - module-relative address in a library will be added with the module-load address, at run time

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

Relocs in a .so - *PIC* referencing of a local symbol

- local symbol reference in PIC shared libraries

- R_386_RELATIVE reloc has a module-relative address of the symbol at run time, add the module-load address to it
- used to mark data addresses in a PIC shared library that need to be relocated at load time
- the run-time loader, part of the dynamic linker, uses to perform load-time relocation

http://netwinder.osuosl.org/users/p/patb/public_html/elf_relocs.html
Linkers and Loaders, J. R. Levine

Relocs in a .so - *PIC* referencing of a global symbol

- the global data symbol reference within PIC shared libaries
- `R_386_GLOB_DAT` reloc at a GOT entry
  - fill the GOT entry with the address of a global data at the load time
- the `R_386_GOT32` reloc at the reference of the data symbol
  - the offset field in this reloc → the GOT entry
  - in order to fetch the address of the global data symbol at the run time

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

Relocs in a .so - *PIC* referencing of a function symbol

- function function symbol reference within PIC shared libraries
- R_386_JMP_SLOT reloc at a PLT entry
  - the PLT entry → the jump target → the GOT entry →
  - fill the GOT entry with the address of the function symbol
  - the resolver fills, after lazy binding
- the R_386_PLT32 reloc at the reference of the function symbol
  - the offset field in this reloc → the PLT entry →
    the jump target → the GOT entry →
    the filled function address at the dynamic link time
  - in order to fetch the address of the function symbol
    at the run time

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

Relocs in a PIC shared library - summary

R_386_JMP_SLOT	S	• *PIC* reference to a function symbol • offset : a PLT entry location • fill the GOT entry with a function symbol address
R_386_GLOB_DAT	S	• *PIC* reference to a global symbol • offset to a GOT entry • fill the GOT entry with a global symbol address
R_386_RELATIVE	B+A	• *PIC* reference to a local symbol • offset to a section • add the load address to the relative address

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

TOC: Executable files

- Separate GOTs and PLTs
- Relocs in an exe : non-PIC referencing of a global symbol
- Relocs in an exe : non-PIC referencing of a function symbol
- Relocs in an exe : PIC referencing of a global symbol
- Relocs in an exe : PIC referencing of a function symbol

Separate GOTs and PLTs

- The GOT converts
  position-independent address calculations
  to absolute locations.
- The PLT converts
  position-independent function calls
  to absolute locations.
- an executable file have its own GOT and PLT
- a shared object file have its own GOT and PLT
- they do not share a GOT nor a PLT

https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-74186.html

Relocs in an exe - *non-PIC* referencing of a global symbol

- non-PIC executable file's access of global symbols in PIC shared libraries

- use `R_386_COPY` instead of `R_386_GLOB_DAT`

- `R_386_COPY` allocates and copies *initialized* global symbols into the application .bss space.

- then the executable and all the shared libraries point to this single copy

- executables need to be able to refer to global data (such as `errno`) as if there is only one copy.

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

Relocs in an exe - *non-PIC* referencing of a function symbol

- non-PIC executable file's access of
  function symbols in PIC shared libraries

- the same as the PIC referencing function symbols

- `R_386_JMP_SLOT` reloc has an offset member
  of a PLT entry location
  the corresponding entry will be filled
  with the address of a library function
  at the dynamic link time

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

Relocs in an exe - *PIC* referencing of a global symbol

- the global data symbol reference in a PIC executable
- R_386_GLOB_DAT reloc at a GOT entry
  - fill the GOT entry with the address of a global data at the load time
- the R_386_GOT32 reloc at the reference of the data symbol
  - the offset field in this reloc → the GOT entry
  - in order to fetch the address of the referenced global symbol at the run time

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

Relocs in an exe - *PIC* referencing of a function symbol

- the global function symbol reference in a PIC executable
- `R_386_JMP_SLOT` reloc at a PLT entry
  - the PLT entry → the jump target → the GOT entry →
  - fill the GOT entry with the address of the function symbol
  - the resolver fills, after lazy binding
- the `R_386_PLT32` reloc at the reference of the function symbol
  - the offset field in this reloc → the PLT entry →
    the jump target → the GOT entry →
    the filled function address at the dynamic link time
  - in order to fetch the address of the function symbol
    at the run time

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

Relocs in a non-PIC exe - sumamary

`R_386_COPY`	None	• *non-PIC* reference to a global symbol • offset : a location in a WR segment • copy the library symbol data into an app's data space
`R_386_JMP_SLOT`	S	• *PIC* reference to a global symbol • offset : a PLT entry location of a *PIC* shared library • fill the location with a function symbol address

- `R_386_GLOB_DAT` : not used in a non-PIC executable file
- these days, PIE (Position Independent Executables), by default
  - no difference in shared library relocs and executable relocs

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

Relocs in an exe - read-only code section

- all the relocs from the .o file have been
  - either resolved or
  - changed into one of three relocs
    1. R_386_COPY (non-PIC reference) → copy into .bss
    2. R_386_GLOB_DAT (PIC reference) → fill the GOT entry in .data
    3. R_386_JMP_SLOT (PIC reference) → fill the GOT entry in .data
- Notice that all of these relocs must modifiy only the data section of the executable
- the code section is read-only

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

TOC: Summary

- Summary - PIC relocs in design cycles
- PIC reloc offsets in an object `.o` file
- PIC reloc offsets in an shared library `.so` file

Summary - PIC relocs in design cycles

	reference in .o	reference in .so
a global symbol	R_386_GOT32	R_386_GLOB_DAT
a local symbol (code)	R_386_GOTOFF	R_386_RELATIVE
a local symbol (data)	R_386_PC32	R_386_RELATIVE
a function symbol	R_386_PLT32	R_386_JMP_SLOT

https://docs.oracle.com/cd/E19683-01/817-3677/chapter6-26/index.html

PIC reloc offsets in an object .o file

R_386_GLOB_DAT $G + A$	• pointing to the GOT entry • distance from GOT[0] to the GOT entry • offset from the start of the GOT to the GOT slot
R_386_GOTOFF $S + A - GOT$	• pointing to the GOT • distance from GOT[0] to the given symbol • offset from the start of the GOT to the symbol
R_386_PC32 $S + A - P$	• pointing to a section (.bss, .data, .text) • distance from a section to the given symbol • offset from the start of a section to the symbol
R_386_PLT32 $L + A - P$	• pointing the PLT entry • distance from the symbol reference to the PLT entry • the address of the PLT entry

https://docs.oracle.com/cd/E19683-01/817-3677/chapter6-26/index.html

PIC reloc offsets in a shared library .so file

R_386_GLOB_DAT S	• pointing to the GOT entry • distance from GOT[0] to the GOT entry • offset from the start of the GOT to the GOT slot
R_386_RELATIVE $B + A$	• pointing to a section • distance from a section to the given symbol • offset from the start of a section to the symbol
R_386_JMP_SLOT S	• pointing the PLT entry • distance from the symbol reference to the PLT entry • the address of the PLT entry

https://docs.oracle.com/cd/E19683-01/817-3677/chapter6-26/index.html

TOC: Copy Relocs

- `.bss` section
- Copy reloc
- Referencing external data using the GOT
- Referencing external data by copying

TOC: .bss section

- All uninitialized objects
- No static local constants
- Summary

All uninitialized objects

- statically-allocated objects without an explicit initializer
  - initialized to zero (for arithmetic types)
  - initialized to a null pointer (for pointer types)
- the .bss section typically includes *all* uninitialized objects
  - uninitialized global symbols
    uninitialized variables and constants
    declared at file scope (i.e., outside any function)
  - uninitialized local symbols
    uninitialized static local variables
    local variables declared with the static keyword

`https://en.wikipedia.org/wiki/.bss`

No static local constants

- static local constants must be initialized with values at declaration, however, as they do not have a separate declaration, and thus are typically not in the `.bss` section, though they may be implicitly or explicitly initialized to zero
- An implementation may also assign to the `.bss` section statically-allocated variables and constants initialized with values consisting solely of zero-valued bits

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

Summary

	global symbols	local symbols
uninitialized	global variables	static global variables
	global constants	static local variables
		static global constants (X)
		static local constants (X)

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

TOC: Copy relocs

- Non-PIC dynamic executable
- Non-PIC dynamic executable's referencing of external data
- Copy relocs
- `R_386_COPY`
- `R_386_COPY` vs. `R_386_GLOB_DATA`
- `R_386_COPY` copies shared library data
- `R_386_GLOB_DATA` references the copied data

Non-PIC dynamic executable

- dynamic executables are generally not created from position-independent code
    - non-PIC executable + PIC shared libraries
    - the non-PIC executable does not have its own GOT / PLT

`819-0690.pdf linker and libraries guide, Oracle`

Non-PIC dynamic executable's referencing of external data

- when a non-PIC executable references external data in PIC shared libraries
- any references to external data (global symbols) can only be achieved at runtime
    - at link time, the exact address is not known
    - the code that references needs to be *modified* at runtime
    - but a read-only text segment cannot be *modified*

`819-0690.pdf linker and libraries guide, Oracle`

Copy relocs

- the copy relocation technique can solve this reference.
  - the run time linker to copy the data from the shared object to the allocated space within the dynamic executable.
  - the executable and the shared libraries refer the copied data instead of the original data in the shared library

`819-0690.pdf linker and libraries guide, Oracle`

R_386_COPY

- created by the link-editor for dynamic executables
  to preserve a read-only text segment.
  - the relocation offset member refers
    to a location in a writable segment.
  - the symbol table index specifies a symbol that should exist
    both in the current object file and in a shared object.
  - during execution, the runtime linker copies
    data associated with the shared object's symbol
    to the location specified by the offset

`819-0690.pdf linker and libraries guide, Oracle`

R_386_COPY vs. R_386_GLOB_DATA

- R_386_COPY copy to the applications data space
  - non-PIC access of external global variables
  - when an non-PIC executable accesses
    a global symbol in a shared object
- R_386_GLOB_DATA indirect reference through GOT
  - PIC access of external global variables
  - when a shared object accesses
    a global symbol in other module
- these are complements of each other

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

R_386_COPY copies shared library data

- Suppose a global data object is defined in a dynamic library
  - the library will have the binary version of the global data object in its data space.
  - when the application is built, the linker puts a R_386_COPY reloc there (in the app) to copy the data down to the application's .bss space.

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

R_386_GLOB_DATA references the copied data

- In turn, the library never references the original global object;
- it references the copied data that is in the application data space, through a corresponding R_386_GLOB_DATA.
- After loading and copying, the original data (from the library) is never used; only the copy (in the app's data space).

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

TOC: Referencing external data using the GOT

- PIC referencing of external data
- Referencing absolute addresses using the GOT
- Link editor vs. runtime linker
- Runtime linker sets absolute addresses
- Multiple GOT's for an absolute address

PIC referencing of external data

- Shared objects are usually built with PIC
- References to external data items from PIC employs indirect addressing through the GOT
- These tables are updated at runtime with the real address of the data items.
- These updated tables enable access to the data without the code itself being modified

`819-0690.pdf linker and libraries guide, Oracle`

Referencing absolute addresses using the GOT

- PIC has no absolute virtual addresses, in general
- the absolute adderess can be stored in the GOT
- A program references its GOT entry and
  extracts absolute values.
  without compromising the position independence and
  shareability of a program's text.

`819-0690.pdf linker and libraries guide, Oracle`

Link editor vs. runtime linker

- if a program requires the absolute address of a symbol, that symbol will have a GOT entry.
- R_386_GLOB_DAT referes to the GOT entry
- the link-editor does not know the absolute addresses
- the runtime linker knows all the addresses

`819-0690.pdf linker and libraries guide, Oracle`

Runtime linker sets absolute addresses

- initially, the GOT holds relocation entry information
- after memory segments for a loadable object file is created the runtime linker processes the relocation entries.
- the runtime linker
    - determines the associated symbol values,
    - calculates their absolute addresses, and
    - sets the appropriate GOT entries to the proper values.

`819-0690.pdf linker and libraries guide, Oracle`

Multiple GOT's for an absolute address

- because the executable file and shared objects
  have a separate GOT,
  a symbol's address can appear in several GOTs.
- The runtime linker processes
  all the GOT relocations
  before giving control to any code

`819-0690.pdf linker and libraries guide, Oracle`

TOC: Referencing external data by copying

- non-PIC referencing of external data
- Assumption for the copy relocaton
- Processing of the copy relocation
- Using the copied data only

non-PIC referencing of external data

- dynamic executables, however, are generally not PIC
- Any references to external data they make can seemingly only be achieved at runtime by modifying the code that makes the reference.
- Modifying a read-only text segment is not allowed
- The copy relocation technique can solve this reference.

`819-0690.pdf linker and libraries guide, Oracle`

Assumption for the copy relocation

- Suppose
  - the link-editor creates a dynamic executable
  - a reference to a data item which is located in one of shared objects
- the link-editor generates a special copy relocation record
- the runtime linker processes this copy relocaiton

`819-0690.pdf linker and libraries guide, Oracle`

Processing of the copy relocation

- Copy relocation
  - allocates space in the dynamic executable's .bss
    with the same size data item in the shared object.
  - assigns the same symbolic name to this space
    as defined in the shared object.
  - instructs the runtime linker
    to copy the data from the shared object
    to the allocated space within the dynamic executable

`819-0690.pdf linker and libraries guide, Oracle`

Using the copied data only

- Because the symbol is global,
  any shared objects can reference this copied symbol
- the dynamic executable owns the copied data item.
- any other objects within the process
  that make reference to this item are bound to this copy
- the original data from which the copy is
  made effectively becomes unused

`819-0690.pdf linker and libraries guide, Oracle`

TOC: Relative Reloc

- Relative reloc `R_386_RELATIVE`
- Load time relocation
- Base address

TOC: Relative reloc R_386_RELATIVE

- Local symbol PIC relocs
- Resolving R_386_GOTOFF and R_386_32
- Zero symol table index
- Offset address
- Base address

Local symbol PIC relocs

a .text section reference of a local symbol defined in .bss or .data	a .data section reference of a local symbol defined in .bss or .data
R_386_GOTOFF	R_386_32
offset relative to &GOT[0] (.data)	offset relative to a section (.data, .bss)
fully resolved at link time no reloc is needed	transformed to R_386_RELATIVE

Resolving R_386_GOTOFF and R_386_32

- an ELF executable consists of a group of code segments followed by a group of data segments
- GOT is located at the beginning of data segments
- &GOT[0] is obtained by GLOBAL_OFFSET_TABLE
  - R_386_GOTOFF is fully resolved at the link time
- .bss does not have corresponding address symbol
  - R_386_32 is converted into R_386_RELATIVE

Zero symbol table index

- created by the link-editor for dynamic objects
- relocation entries for R_386_RELATIVE must specify a value of zero for the symbol table index

`819-0690.pdf linker and libraries guide, Oracle`

Offset address

- Offset Address : the relocation offset member gives the location within a shared object that contains a value representing a relative address

`819-0690.pdf linker and libraries guide, Oracle`

Base address

- Base Address : the runtime linker computes
  the corresponding virtual address of the referenced symbol
  by adding the virtual address (*base*)
  at which the shared object is loaded
  to the relative address (*offset*)

`819-0690.pdf linker and libraries guide, Oracle`

TOC: Symbol table index

- Symbol table
- Symbol table index
- Symbol table - special values
- Symbol table - ABS
- Symbol table - COMMON
- Symbol table - UNDEF

Symbol table

- An object file's symbol table holds information needed to locate and relocate a program's symbolic definitions and references.
- A ymbol table index is a subscript into this array.
- Index 0 both
  designates the first entry in the table and
  serves as the undefined symbol index.

https://refspecs.linuxbase.org/elf/gabi4+/ch4.symtab.html

Symbol table index

- If a symbol's value refers to
  a specific location within a section,
  the symbols's section index member, st_shndx,
  holds an index into the section header table.

- Every symbol table entry is defined
  in relation to some section.
  This member holds the relevant
  section header table index.

https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-79797.html#chapter6-t

Symbol Index - special values

- Some special section index values give other semantics.
    - SHN_ABS
    - SHN_COMMON
    - SHN_UNDEF

`https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-79797.html#chapter6-t`

Symbol Index - ABS

- SHN_ABS
  - This symbol has an absolute value that does not change because of relocation.

https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-79797.html#chapter6-tl

Symbol Index - COMMON

- `SHN_COMMON`
  - This symbol labels a common block that has not yet been allocated.
  - The symbol's value gives alignment constraints
  - The link-editor allocates the storage for the symbol at an address that is a multiple of `st_value`.
  - The symbol's size tells how many bytes are required.

`https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-79797.html#chapter6-t`

Symbol Index - UNDEF

- SHN_UNDEF
  - This section table index indicates that the symbol is undefined.
  - When the link-editor combines this object file with another object that defines the indicated symbol, this file's references to the symbol is bound to the definition.

https://docs.oracle.com/cd/E23824_01/html/819-0690/chapter6-79797.html#chapter6-tl

TOC: Load time relocation

- Shared library's text is PIC
- Shared library's data is non-PIC
- Load-time relocation
- GOT pointers to data

Shared library's text is PIC

- the text in shared libraries is always PIC
  there are no relocation entries for the code,
- if a shared library is built with non-PIC code
  then there will be relocation entries for the text as well,
  although it is useless because nonsharable text

J. R. Levine, Linkers and Loaders

Shared library's data is non-PIC

- data can be non-PIC,
  so there is a relocation entry
  for every pointer in the data segment
  - global symbols : `R_386_GLOB_DAT` at the GOT in the data
  - local symbols : `R_386_RELATIVE` in the data and the code

J. R. Levine, Linkers and Loaders

Load-time relocation

- ELF shared libraries contain R_386_RELATIVE reloc entries that the run-time loader uses to do load-time relocation
- at load time,
  - the code segment of a PIC file need not be *relocated*
  - the data segment does need to be *relocated*

J. R. Levine, Linkers and Loaders

GOT pointers to data

- in large libraries, the GOT can be very large,
  it can take a long time to resolve all the entries
  - problem in dynamic linking
- handling `R_386_RELATIVE` items or
  the equivalent to relocate GOT pointers to data
  in the same executable is fairly fast,
- but the problem is that many GOT pointers to data
  in other executables would require
  a symbol table lookup to resolve

`J. R. Levine, Linkers and Loaders`

TOC: Base address

- Base address
- Computing base addresses

Base address

- B in B+A
- the base address at which a shared library object has been loaded into memory during execution
- Generally, a shared library object file is built with a 0 base virtual address but the actual execution address will be different.

https://stackoverflow.com/questions/28805940/how-can-i-get-a-value-of-elf-file

Computing base address

- to compute the base address,
  one determines the memory address associated with
  the lowest p_vaddr value for a PT_LOAD segment
- one then obtains the base address
  by truncating the memory address
  to the nearest multiple of the maximum page size
- Depending on the kind of file being loaded into memory,
  the memory address might or might not
  match the p_vaddr values.

https://stackoverflow.com/questions/28805940/how-can-i-get-a-value-of-elf-file