Link 8. Dynamic Linking

Young W. Lim

2019-01-01

Outline

Based on

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

Dynamic linking with shared library example

1. Compiler flags for dynamic linking
2. `addvec.c` and `multvec.c`
3. `libvec.so`
4. `main.c`
5. `p`
6. Steps of dynamic linking
7. Inputs and outputs dynamic linking steps

Compiler flags for dynamic linking

- `-fPIC` flag directs the compiler
  to generate position independent code
- `-shared` flag directs the linker
  to create a shared object file

addvec.c and multvec.c ~

```
/*::::: addvec.c ::::::::::::::::::::::::::*/
void addvec(int *x, int *y, int *z, int n)
{
  int i;

  for (i=0; i<n; i++)
    z[i] = x[i] + y[i];

}

/*::::: multvec.c ::::::::::::::::::::::::::*/
void multvec(int *x, int *y, int *z, int n)
{
  int i;

  for (i=0; i<n; i++)
    z[i] = x[i] * y[i];

}
```

libvector.so

1. gcc -c -fPIC addvec.c
   - addvec.o
2. gcc -c -fPIC multvec.c
   - multvec.o
3. gcc -shared -o libvector.so addvec.o multvec.o
   - libvector.so

main.c

```
/*::::: vector.h ::::::::::::::::::::::::::::*/
void addvec(int *x, int *y, int *z, int n);
void multvec(int *x, int *y, int *z, int n);


/*::::: main.c ::::::::::::::::::::::::::::::*/
#include <stdio.h>
#include "vector.h"

int x[2] = { 1, 2};
int y[2] = { 3, 4};
int z[2];


int main() {

  addvec(x, y, z, 2);
  printf("z= [%d %d]\n", z[0], z[1]);

}
```

p

1. gcc -c main.c
   - main.o
2. gcc -o p main.o ./libvector.so
   - p
3. LD_LIBRARY_PATH=$LD_LIBRARY_PATH:./
4. export LD_LIBRARY_PATH
   - ./p

Steps of dynamic linking

1. `main2.c`, `vector.h` ⇒ `main2.o`
   - translators (`cpp`, `ccl`, `as`)
1. `main2.o`, `libc.so`, `libvector.so` ⇒ `p2`
   - linker (`ld`)
1. `p2` ⇒ partially linked `p2` in memory
   - loader (`execve`)
1. `p2`, `libc.so`, `libvector.so` ⇒ fully linked executable in memory
   - dynamic linker (`ld-linux.so`)

Inputs and outputs dynamic linking steps

1. Linker `ld` inputs
   - relocatble object file :
     `main2.o`
   - relocation and symbol table information :
     `libc.so`, `libvector.so`
2. Loader `execve` input
   - partially linked executable object file :
     `p2`
3. Dynamic linker `ld-linux.so` inputs
   - loaded
     `p2`
   - code and data :
     `libc.so`, `libvector.so`
4. fully linke executable in memory

Shared Libraries

1. Shared libraries
2. Shared libraries - only a single copy
3. Shared libraries - shared in two ways

Shared libraries

- an object module
  - that can be loaded at run time
  - at an *arbitrary* memory address
  - linked with a *program in memory*
- also referred as shared object
  with `.so` suffix
- dynamic liking is performed by a dynamic linker
  contained in the ld-linux.so interpreter
- corresponds to DLLs (Dynamic Link Libraries) on MS Window

Shared libraries - only a single copy

- a sigle copy of .so file :
  there is exactly one .so file
  for a particular library
- in case of static libraries
  the contents are copied

Shared libraries - shared in two ways

- shared in two different ways
- the code and data in a `.so` file
  are shared by *all the executable files*
  that reference the library
- a single copy of the `.text` section
  of a shared library in memory
  can be shared by *different running processes*

Dynamic Linking

1. First link statistically and then link dynamically
2. The first partial link (static)
3. The second complete link (dynamic)
4. The second complete link (relocation)
5. execution of dynamically linked file

First link statically and then link dynamically

- basic idea :
  - link some thing *statically*
    when the executable file is *created*
  - then complete the linking process *dynamically*
    when the program is *loaded* into memory

The first partial link (static)

- none of the code or data sections are actually copied from `libvector.so` into the executable file
- the linker copies some information about
  - relocation
  - symbol table
- this will allow references to code and data in `libvector.so` to be resolved at run time

The second complete link (dynamic)

- when the loader loads and runs the executable
  it loads the *partially* linked executable
- if the executable contains `.interp` section (interpreter)
  which contains the path name of the dynamic linker
- the dynamic linker itself is a shared object `ld-linux.so`
- instead of passing control to the application
- the loader loads and runs the dynamic linker

The second complete link (relocation)

- shared libraries are loaded in the area starting at address `0x40000000`

1. relocating the text and data of `libc.so` into some memory segment

1. relocating the text and the data of `libvector.so` into another memory segment
2. relocating any references in `p2` to symbols defned by `libc.so` and `libvector.so`

execution of dynamically linked file

- finally, the dynamic linker passes control to the application
- from this point, the locations of
  the shared libraries are fixed
  and do not change during execution of the program

Compiler options and paths for dynamic linking

1. `-fPIC` flag for dynamic linking
2. `-fPIC` vs `-fPIC` flagsO
3. `-shared` flag for dynamic linking
4. `-shared` flag
5. locating shared libraries

-fPIC flag for dynamic linking

- Generate position-independent code (PIC) suitable for use in a shared library, if supported for the target machine.
- PIC code accesses all constant addresses through a global offset table (GOT).
- The dynamic loader resolves the GOT entries when the program starts

-fpic vs -fPIC flags

- the dynamic loader is not part of GCC;
  it is part of the operating system.
- If the GOT size for the linked executable exceeds
  a machine-specific *maximum size*, -fpic does not work;
  in that case, recompile with -fPIC instead.

-shared flag for dynamic linking

- -shared
  - Create a shared library.
  - This is currently only supported on ELF, XCOFF and SunOS platforms.
- -soname=name
  - When creating an ELF shared object, set the internal DT_SONAME field to the specified name.
  - When an executable is linked with a shared object which has a DT_SONAME field, then when the executable is run the dynamic linker will attempt to load the shared object specified by the DT_SONAME field rather than the using the file name given to the linker.

-static flag

- -static
  - Do not link against shared libraries.
  - You may use this option multiple times on the command line:
  - it affects library searching for -l options which follow it.
  - This option also implies --unresolved-symbols=report-all.
  - This option can be used with -shared.
    - Doing so means that a shared library is being created
      but that all of the library's external references
      must be resolved by pulling in entries from static libraries.

locating shared libraries (1)

1. Any directories specified by `-rpath-link` options.
   - only effective at link time
1. Any directories specified by `rpath` options.
   - used at runtime
   - supported by native linkers
   - supported by cross linkers
     that are configured with `--with-systroot`
1. On an ELF system, for native linkers
   if the -rpath and -rpath-link options were not used
   search the contents of the environment variable
   `LD_RUN_PATH`

locating shared libraries (2)

1. On SunOS, if the `-rpath` option was not used,
   search any directories specified using `-L` options.

1. For a native linker,
   search the contents of the environment variable
   `LD_LIBRARY_PATH`

1. For a native ELF linker,
   the directories in `DT_RUNPATH` or `DT_RPATH`
   of a shared library are searched for
   shared libraries needed by it.
   The `DT_RPATH` entries are ignored
   if `DT_RUNPATH` entries exist.

locating shared libraries (3)

1. The default directories,
   normally `/lib` and `/usr/lib`.

1. For a native linker on an ELF system,
   if the file `/etc/ld.so.conf` exists,
   the list of directories found in that file.

Some links

https://stackoverflow.com/questions/25084855/
how-does-gcc-shared-option-affect-the-output
https://unix.stackexchange.com/questions/475/
how-do-so-shared-object-numbers-work
https://stackoverflow.com/questions/12237282/
whats-the-difference-between-so-la-and-a-library-files

Loading and linking shared libraries from Applications

1. Dynamic Linker Interface
2. `dl_open`
3. `dlsym`
4. `dlclose`
5. `dlerror`
6. an example for application's dynamic linking
7. compiler options

Dynamic Linker vs. Applications

- the dynamic linker loads and links shared libraries
  when application is loaded, just before it executes
- an applications can also request the dynamic linker
  to load and link arbitrary shared libraries
  while the application is running
  without having to link in the applications
  against those libraries at compile time

Dynamic Linker Interface

```
#include <dlfcn.h>

void *dlopen(const char *filename, int flag);
      returns ptr to handle if OK, NULL on error

void *dlsym(void *handle, char *symbol);
      returns ptr to symbol if OK, NULL on error

int dlclose (void *handle);
      returns zero if OK, -1 on error

const char *delerror(void);
      returns error message if previous call
      to dlopen, dlsym, dlclose failed
      NULL if previous call is OK
```

dlopen (1)

```
void *dlopen(const char *filename, int flag)
```

- loads and links the shared library `filename`
- the external symbols in `filename` are resolved
  using libraries previously opened with the `RTLD_GLOBAL` flag
- if the current was compiled with `rdynamic` flag,
  then its global symbols are also available for symbol resolution

dlopen (2)

```
void *dlopen(const char *filename, int flag);
```

- the flag argument must include
  - RTLD_NOW
    tells the linker to resolve references immediately
  - RTLD_LAZY
    tells the linker to defer symbol resolution
    until the code from the library is executed
  - RTLD_GLOBAL flag can be or'ed

dlsym

void *dlsym(void *handle, char *symbol);

- inputs
  - a handle to a previously opened shared library
  - a symbol name
- returns the address of the symbol if it exists
  or NULL otherwise

dlclose

int dlclose (void *handle);

- unloads the shared library
  if no other shared libraries are still using it

dlerror

const char *delerror(void);

- returns a string describing the most recent error
  that occurred as a result of calling
  `dlopen`, `dlsym`, `dlclose`
  or `NULL` if no error occurred

an example for an application's dynamic linking (1)

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

int x[2] = {1,2};
int y[2] = {3,4};
int z[2];

int main() {
  void *handle;
  void (*addvec) (int*, int*, int*, int);
  char *error;

  handle = dlopen("./libvector.so", RTLD_LAZY);
  if (!handle) {
    fprintf(stderr, "%s\n", dlerror());
    exit(1);
  }
```

an example for an application's dynamic linking (2)

```
  addvec = dlsym(handle, "addvec");
  if ((error = dlerror()) != NULL) {
    fprintf(stderr, "%s\n", error);
    exit(1);
  }

  addvec(x, y, z, 2);
  printf("z = [%d %d]\n", z[0], z[1]);

  if (dlclose(handle) < 0) {
    fprintf("stderr, "%s\n", dlerror());
    exit(1);
  }

  return 0;
}
```

summary

- declaration

```
void *handle;
void (*addvec) (int*, int*, int*, int);
char *error;
```

- loading a shared library

```
handle = dlopen("./libvector.so", RTLD_LAZY) ;
```

- locating address of a fuction

```
addvec = dlsym(handle, "addvec") ;
```

- unloading the shared library

```
dlclose(handle) ;
```

compiler options

- #include <dlfcn.h>
- -ldl

- gcc -o p3 dlex.c -ldl

n_main.c

- ```
  // nothing.h ------------------------------------
  static void doNothingStatic();
  void doNothing();
  void doAlmostNothing();
  ```
- ```
  // n_main.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.h ------------------------------------
```
static void doNothingStatic();
void doNothing();
void doAlmostNothing();
```
- // nothing.c ------------------------------------
```
#include "nothing.h"

static void doNothingStatic() {
}

void doNothing() {
}

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

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

commands for an executable and a shared library

```
$ 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
```

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 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 namin_dyn.out      ..... address = 154
  $hexdump -C namin_dyn.out      ..... path = /lib/ld-linux.so.2
  ```

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

sections vs. segmens

- 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-findi`

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-findi

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-findi

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-findi

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

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

PIC (Position Independent Code)

- GOT (Global Offset Table)
- PLT (Procedure Linkage Table)

- ```
  $ readelf --segments nmain_dyn.out
  $ gcc -Wall -g -O0 -fPIC -c nothing.c -o nothing_pic.o
  $ gcc -shared -o libnothing.so nothing_pic.o
  $ objdump -d -s nmain.out
  $ objdump -d -s 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
  ```

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`)
- 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`

.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           R E  00 .dynsym .dynstr .rel.dyn .rel.plt .plt .plt.got .text
LOAD           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

.got, .got.plt, .plt, .plt.got

- text segment
  - Read only
  - .plt, .plt.got (got in plt region)
- data segement
  - Read Write
  - .got, .got.plt (plt in got region)
- 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

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$_d$yn.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`
- `objdump -d -s nmain.out`

```
508:  e8 30 00 00 00          call   53d <doAlmostNothing>
```

- `objdump -d -s nmain_dyn.out`

```
5e8:  e8 73 fe ff ff          call   460 <doAlmostNothing@plt>
```

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

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

.plt section disasembly

```
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

doAlmostNothing (2)

```
0000053d <doAlmostNothing>:
 53d:   55                      push   %ebp
 53e:   89 e5                   mov    %esp,%ebp
 540:   53                      push   %ebx
 541:   83 ec 04                sub    $0x4,%esp
 544:   e8 a7 fe ff ff          call   3f0 <__x86.get_pc_thunk.bx>
 549:   81 c3 93 1a 00 00       add    $0x1a93,%ebx
 54f:   e8 c9 ff ff ff          call   51d <doNothingStatic>
 554:   e8 d4 ff ff ff          call   52d <doNothing>
 559:   90                      nop
 55a:   83 c4 04                add    $0x4,%esp
 55d:   5b                      pop    %ebx
 55e:   5d                      pop    %ebp
 55f:   c3                      ret
```

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

readelf -r nmain$_d$yn.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

.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-ar

.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-ar

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)
```

as suffix : GOT

- attaching this suffix to a symbol in an instruction causes the symbol to be entered into the got (global offset table).
- The value is a 32-bit index for that symbol into 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)

- this suffix is used for function symbols.
- It causes a plt (procedure linkage table),
  an array of code stubs, to be created at the time
  the shared object is created or linked against,
- each entry of plt (a code stub) is associated with
  a got (global offset table) entry
  - the value of such a got entry
    is a pc-relative offset
    to the corresponding stub code in the plt

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

as suffix : PLT (2)

- this arrangement causes the run-time symbol resolver
  to 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
- the value is relative to the beginning of the got
  not a pc-relative offset
- the relocation is 'R_CRIS_32_PLT_GOTREL'.

- `move.dfnname:PLTG,$r3`

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

as suffix : GOTPLT

- a mix between the effect of the GOT and the PLT suffix;
- Similar to PLT
- the value of the symbol is a 32-bit index into the got
- 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`