WIKIPEDIA

Relocation (computing)

Relocation is the process of assigning load addresses for position-dependent code and data of a program and adjusting the code and data to reflect the assigned addresses.[1][2] Prior to the advent of multiprocess systems, and still in many embedded systems the addresses for objects were absolute starting at a known location, often zero. Since multiprocessing systems dynamically link and switch between programs it became necessary to be able to relocate objects using position-independent code. A linker usually performs relocation in conjunction with **symbol resolution**, the process of searching files and libraries to replace symbolic references or names of libraries with actual usable addresses in memory before running a program.

Relocation is typically done by the linker at link time, but it can also be done at load time by a relocating loader, or at run time by the running program itself. Some architectures avoid relocation entirely by deferring address assignment to run time; this is known as zero address arithmetic.

Contents

Segmentation

Object files are segmented into various memory segment types. Example segments include code segment(.text), initialized data segment(.data), uninitialized data segment(.bss), or others.

Relocation table

The relocation table is a list of pointers created by the translator (a compiler or assembler) and stored in the object or

executable file. Each entry in the table, or "fixup", is a pointer to an absolute address in the object code that must be changed when the loader relocates the program so that it will refer to the correct location. Fixups are designed to support relocation of the program as a complete unit. In some cases, each fixup in the table is itself relative to a base address of zero, so the fixups themselves must be changed as the loader moves through the table.[3]

In some architectures a fixup that crosses certain boundaries (such as a segment boundary) or that is not aligned on a word boundary is illegal and flagged as an error by the linker.[4]

DOS and 16-bit Windows

Far pointers (32-bit pointers with segment:offset, used to address 20-bit 640 KB memory space available to DOS programs), which point to code or data within a DOS executable (EXE), do not have absolute segments, because the actual address of code/data depends on where the program is loaded in memory and this is not known until the program is loaded.

Instead, segments are relative values in the DOS EXE file. These segments need to be corrected, when the executable has been loaded into memory. The EXE loader uses a relocation table to find the segments which need to be adjusted.

32-bit Windows

With 32-bit Windows operating systems it is not mandatory to provide relocation tables for EXE files, since they are the first image loaded into the virtual address space and thus will be loaded at their preferred base address.

For both DLLs and for EXEs which opt into address space layout randomization (ASLR) - an exploit mitigation technique introduced with Windows Vista, relocation tables once again become mandatory because of the possibility that the binary may be dynamically moved before being executed, even though they are still the first thing loaded in the virtual address space.

64-bit Windows

When running native 64-bit binaries on Windows Vista and above, ASLR is mandatory, and thus relocation sections cannot be omitted by the compiler.

Unix-like systems

The Executable and Linkable Format (ELF) executable format and shared library format used by most Unix-like systems allows several types of relocation to be defined.[5]

Relocation procedure

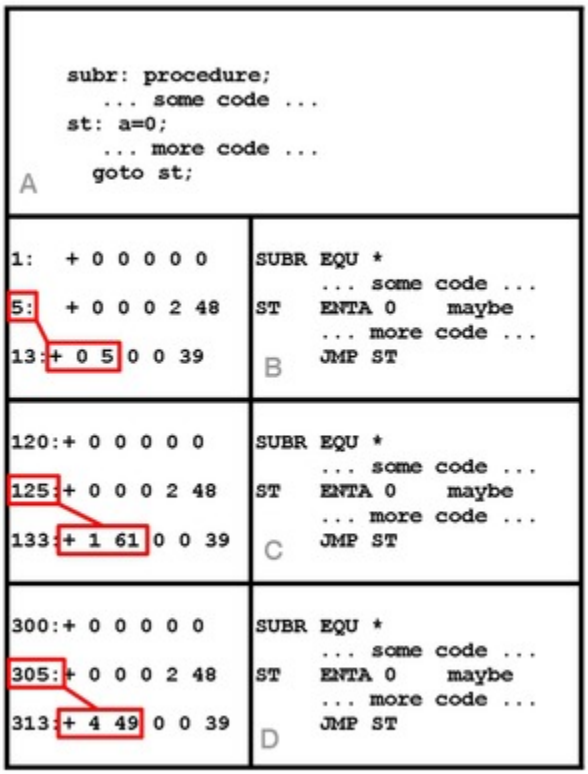
The linker reads segment information and relocation tables in the object files and performs relocation by:

- merging all segments of common type into a single segment of that type

- assigning unique run time addresses to each section and each symbol, giving all code (functions) and data (global variables) unique run time addresses
- referring to the **relocation table** to modify symbols so that they point to the correct run time addresses.

Example

The following example uses Donald Knuth's MIX architecture and MIXAL assembly language. The principles are the same for any architecture, though the details will change.



- (A) Program *SUBR* is compiled to produce object file (B), shown as both machine code and assembler. The compiler may start the compiled code at an arbitrary location, often location 1 as shown. Location 13 contains the machine code for the jump instruction to statement *ST* in location 5.
- (C) If *SUBR* is later linked with other code it may be stored at a location other than 1. In this example the linker places it at location 120. The address in the jump instruction, which is now at location 133, must be **relocated** to point to the new location of the code for statement *ST*, now 125. [1 61 shown in the instruction is the MIX machine code representation of 125].
- (D) When the program is loaded into memory to run it may be loaded at some location other than the one assigned by the linker. This example shows *SUBR* now at location 300. The address in the jump instruction, now at 313, needs to be relocated again so that it points to the updated location of *ST*, 305. [4 49 is the MIX machine representation of 305].

See also

- Linker (computing)
- Library (computing)
- Object file
- Prebinding
- Static library
- Self-relocation
- Position-independent code (PIC)

- Rebasing
- Garbage collection

References

1. "Types of Object Code". *iRMX 86 Application Loader Reference Manual* (ftp://bitsavers.informatik.uni-stuttgart.de/pdf/intel/iRMX/iRMX_86_Rev_6_Mar_1984/146196_Burst/iRMX_86_Application_Loader_Reference_Manual.pdf) (PDF). Intel. pp. 1–2, 1–3. Retrieved 2017-08-21. "[…] *Absolute code*, and an absolute object module, is code that has been processed by LOC86 to run only at a specific location in memory. The Loader loads an absolute object module only into the specific location the module must occupy. *Position-independent code* (commonly referred to as PIC) differs from absolute code in that PIC can be loaded into any memory location. The advantage of PIC over absolute code is that PIC does not require you to reserve a specific block of memory. When the Loader loads PIC, it obtains iRMX 86 memory segments from the pool of the calling task's job and loads the PIC into the segments. A restriction concerning PIC is that, as in the PL/M-86 COMPACT model of segmentation […], it can have only one code segment and one data segment, rather than letting the base addresses of these segments, and therefore the segments themselves, vary dynamically. This means that PIC programs are necessarily less than 64K bytes in length. PIC code can be produced by means of the BIND control of LINK86. *Load-time locatable code* (commonly referred to as LTL code) is the third form of object code. LTL code is similar to PIC in that LTL code can be loaded anywhere in memory. However, when loading LTL code, the Loader changes the base portion of pointers so that the pointers are independent of the initial contents of the registers in the microprocessor. Because of this fixup (adjustment of base addresses), LTL code can be used by tasks having more than one code segment or more than one data segment. This means that LTL programs may be more than 64K bytes in length. FORTRAN 86 and Pascal 86 automatically produce LTL code, even for short programs. LTL code can be produced by means of the BIND control of LINK86. […]"
2. Levine, John R. (October 1999). "Chapter 1: Linking and Loading". *Linkers and Loaders*. Morgan Kaufmann. p. 5. ISBN 1-55860-496-0.
3. Levine, John R. (October 1999). "Chapter 3: Object Files". *Linkers and Loaders*. Morgan Kaufmann. ISBN 1-55860-496-0.
4. "Borland article #15961: Coping with 'Fixup Overflow' messages" (https://web.archive.org/web/20070324041806/http://vmlinux.org/~jakov/community.borland.com/15961.html). Archived from the original (http://vmlinux.org/~jakov/community.borland.com/15961.html) on 2007-03-24. Retrieved 2007-01-15.
5. "Executable and Linkable Format (ELF)" (http://www.skyfree.org/linux/references/ELF_Format.pdf) (PDF). *skyfree.org*. Retrieved 2018-10-01.

Further reading

- Johnson, Glenn (1975-12-21) [1975-11-13], *11/34 Memory Management Basic Logic test* (https://archive.org/stream/bitsavers_decpdp11xxAINDEC11DFKTAAD1134MEMORYMANAGEMENTBASIC_2477046/MAINDEC-11-DFKTA-A-D_1134_MEMORY_MANAGEMENT_BASIC_LOGIC_Dec75), Digital

Equipment Corporation (DEC), MAINDEC-11-DFKTA-A-D, retrieved 2017-08-19

- Kildall, Gary Arlen (February 1978). "A simple technique for static relocation of absolute machine code" (https://groups.google.com/d/msg/comp.os.cpm/TLHgIi16yTo/gupNB1ai8UQJ). *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia (DDJ)*. People's Computer Company. **3** (2): 10-13 (66-69). ISBN 0-8104-5490-4. #22. Archived (https://archive.today/20170909091943/https://groups.google.com/forum/%23!msg/comp.os.cpm/TLHgIi16yTo/gupNB1ai8UQJ#!topic/comp.os.cpm/TLHgIi16yTo) from the original on 2017-09-09. Retrieved 2017-08-19. [1] (https://archive.org/stream/dr_dobbs_journal_vol_03/dr_dobbs_journal_vol_03_djvu.txt) [2] (https://web.archive.org/web/20170819141800/http://www.retrotechnology.com/dri/d_dri_refs.html) [3] (https://web.archive.org/web/20170819173516/http://archive.computerhistory.org/resources/access/text/2016/12/102762506-05-01-acc.pdf) (This "resize" method, named page boundary relocation, could be applied to a CP/M-80 disk image using MOVCPM in order to maximize the TPA for programs to run. It was also utilized by the CP/M debugger Dynamic Debugging Tool (DDT). The same approach was independently developed by Bruce Van Natta of IMS Associates to produce relocatable PL/M code. Another variant of this method was later utilized by HMA self-relocating TSRs like KEYB, SHARE, and NLSFUNC under DR DOS 6.0 and higher. A much more sophisticated method based on a somewhat similar approach was independently conceived and implemented by Matthias R. Paul and Axel C. Frinke for their dynamic dead-code elimination to dynamically minimize the runtime footprint of resident drivers and TSRs (like FreeKEYB).)
- Roth, Richard L. (February 1978) [1977]. "Relocation Is Not Just Moving Programs" (https://archive.org/stream/dr_dobbs_journal_vol_03/dr_dobbs_journal_vol_03_djvu.txt). *Dr. Dobb's Journal of Computer Calisthenics & Orthodontia (DDJ)*. Ridgefield, CA, USA: People's Computer Company. **3** (2): 14-20 (70-76). ISBN 0-8104-5490-4. #22. Archived (https://web.archive.org/web/20190420010941/https://archive.org/stream/dr_dobbs_journal_vol_03/dr_dobbs_journal_vol_03_djvu.txt) from the original on 2019-04-19. Retrieved 2019-04-19.
- *The Microsoft OBJ File Format* (https://www.fileformat.info/format/ms-obj/corion.htm). Microsoft, Product Support Services. Application Note SS0288. Archived (https://archive.today/20170909092856/http://www.fileformat.info/format/ms-obj/corion.htm) from the original on 2017-09-09. Retrieved 2017-08-21.
- Tanenbaum, Andrew Stuart; Bos, Herbert (2015). *Modern Operating Systems* (4 ed.). Pearson Education Inc. ISBN 978-013359162-0.
