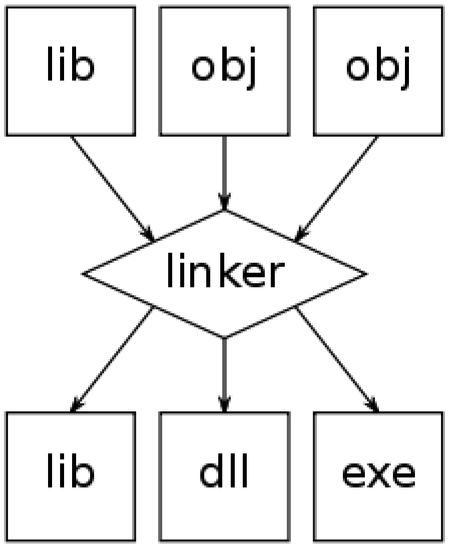
WIKIPEDIA

Linker (computing)

In computing, a **linker** or **link editor** is a computer program that takes one or more object files generated by a compiler and combines them into a single executable file, library file, or another 'object' file.

A simpler version that writes its output directly to memory is called the *loader*, though loading is typically considered a separate process.[1]



An illustration of the linking process. Object files and static libraries are assembled into a new library or executable

Contents

Overview

Computer programs typically are composed of several parts or modules; these parts/modules need not all be contained within a single object file, and in such cases refer to each other by means of symbols. Typically, an object file can contain three kinds of symbols:

- defined "external" symbols, sometimes called "public" or "entry" symbols, which allow it to be called by other modules,
- undefined "external" symbols, which reference other modules where these symbols are defined, and
- local symbols, used internally within the object file to facilitate relocation.

For most compilers, each object file is the result of compiling one input source code file. When a program comprises multiple object files, the linker combines these files into a unified executable program, resolving the symbols as it goes along.

Linkers can take objects from a collection called a *library*. Some linkers do not include the whole library in the output; they include only its symbols that are referenced from other object files or libraries. Libraries exist for diverse purposes, and one or more system libraries are usually linked in by default.

The linker also takes care of arranging the objects in a program's address space. This may involve *relocating* code that assumes a specific base address to another base. Since a compiler seldom knows where an object will reside, it often assumes a fixed base location (for example, zero). Relocating machine code may involve re-targeting of absolute jumps, loads and stores.

The executable output by the linker may need another relocation pass when it is finally loaded into memory (just before execution). This pass is usually omitted on hardware offering virtual memory: every program is put into its own address space, so there is no conflict even if all programs load at the same base address. This pass may also be omitted if the executable is a position independent executable.

On some Unix variants, such as SINTRAN III, the process performed by a linker (assembling object files into a program) was called *loading* (as in loading executable code onto a file).[2] Additionally, in some operating systems the same program handles both the jobs of linking and loading a program (dynamic linking).

Dynamic linking

Many operating system environments allow dynamic linking, that is the postponing of the resolving of some undefined symbols until a program is run. That means that the executable code still contains undefined symbols, plus a list of objects or libraries that will provide definitions for these. Loading the program will load these objects/libraries as well, and perform a final linking. Dynamic linking needs no linker.

This approach offers two advantages:

- Often-used libraries (for example the standard system libraries) need to be stored in only one location, not duplicated in every single binary.
- If a bug in a library function is corrected by replacing the library, all programs using it dynamically will benefit from the correction after restarting them. Programs that included this function by static linking would have to be re-linked first.

There are also disadvantages:

- Known on the Windows platform as "DLL Hell", an incompatible updated library will break executables that depended on the behavior of the previous version of the library if the newer version is not properly backward compatible.
- A program, together with the libraries it uses, might be certified (e.g. as to correctness, documentation requirements, or performance) as a package, but not if components can be replaced. (This also argues against automatic OS updates in critical systems; in both cases, the OS and libraries form part of a *qualified* environment.)

Static linking

Static linking is the result of the linker copying all library routines used in the program into the executable image. This may require more disk space and memory than dynamic linking, but is more portable, since it does not require the presence of the library on the system where it runs. Static linking also prevents "DLL Hell", since each program includes exactly the versions of library routines that it requires, with no conflict with other programs. In addition, a program using just a few routines from a library does not require the entire library to be installed.

Relocation

As the compiler has no information on the layout of objects in the final output, it cannot take advantage of shorter or more efficient instructions that place a requirement on the address of another object. For example, a jump instruction can reference an absolute address or an offset from the current location, and the offset could be expressed with different lengths depending on the distance to the target. By generating the most conservative instruction (usually the largest relative or absolute variant, depending on platform) and adding *relaxation hints*, it is possible to substitute shorter or more efficient instructions during the final link. This step can be performed only after all input objects have been read and assigned temporary addresses; the **linker relaxation** pass subsequently reassigns addresses, which may in turn allow more relaxations to occur. In general, the substituted sequences are shorter, which allows this process to always converge on the best solution given a fixed order of objects; if this is not the case, relaxations can conflict, and the linker needs to weigh the advantages of either option.

While instruction relaxation typically occurs at link-time, inner-module relaxation can already take place as part of the optimising process at compile-time. In some cases, relaxation can also occur at load-time as part of the relocation process or combined with dynamic dead-code elimination techniques.

Linkage editor

In IBM System/360 mainframe environments such as OS/360, including z/OS for the z/Architecture mainframes, this type of program is known as a *linkage editor*. However, a linkage *editor*, as the name implies, has the additional capability of allowing the addition, replacement, and/or deletion of individual program sections. Operating systems such as OS/360 have a different format for executable load-modules, in that they contain supplementary data about the component sections of a program, so that an individual program section can be replaced, and other parts of the program updated so that relocatable addresses and other such references can be corrected by the linkage editor, as part of the process.

One advantage of this is that it allows a program to be maintained without having to keep all of the intermediate object files, or without having to re-compile program sections that haven't changed. It also permits program updates to be distributed in the form of small files (originally card decks), containing only the object module to be replaced. In such systems, object code is in the form and format of 80-byte punched-card images, so that updates can be introduced into a system using that medium. In later releases of OS/360 and in subsequent systems, load-modules contain additional data about versions of components modules, to create a traceable record of updates.

Note: the term "linkage editor" should not be construed as implying that the program operates in a user-interactive mode (like, for example, a text editor). It is strictly intended for batch-mode execution, with the editing commands being supplied by the user on sequentially organized records, such as "unit record" media (a card deck, for example) or DASD media (a disk file, for example), but a tape is also supported, and tapes were often used during the initial installation of the OS.

See also

- Compile and go system
- DLL Hell
- Direct binding
- Dynamic binding (computing)
- Dynamic dispatch
- Dynamic library
- Dynamic linker
- Dynamic loading
- Dynamic-link library
- GNU linker
- Library (computing)
- Loader (computing)
- Name decoration
- Prelinking (prebinding)
- Relocation
- Static library
- gold (linker)

References

Citations

1. IBM Corporation (1972). *IBM OS Linkage Editor and Loader* (http://bitsavers.informatik.uni-stuttgart.de/pdf/ibm/360/os/R21.0_Mar72/GC28-6538-9_OS_Linkage_Editor_and_Loader_Release_21_Jan72.pdf) (PDF).