WIKIPEDIA

Loader (computing)

In computer systems a **loader** is the part of an operating system that is responsible for loading programs and libraries. It is one of the essential stages in the process of starting a program, as it places programs into memory and prepares them for execution. Loading a program involves reading the contents of the executable file containing the program instructions into memory, and then carrying out other required preparatory tasks to prepare the executable for running. Once loading is complete, the operating system starts the program by passing control to the loaded program code.

All operating systems that support program loading have loaders, apart from highly specialized computer systems that only have a fixed set of specialized programs. Embedded systems typically do not have loaders, and instead the code executes directly from ROM. In order to load the operating system itself, as part of booting, a specialized boot loader is used. In many operating systems the loader is permanently resident in memory, although some operating systems that support virtual memory may allow the loader to be located in a region of memory that is pageable.

In the case of operating systems that support virtual memory, the loader may not actually copy the contents of executable files into memory, but rather may simply declare to the virtual memory subsystem that there is a mapping between a region of memory allocated to contain the running program's code and the contents of the associated executable file. (See memory-mapped file.) The virtual memory subsystem is then made aware that pages with that region of memory need to be filled on demand if and when program execution actually hits those areas of unfilled memory. This may mean parts of a program's code are not actually copied into memory until they are actually used, and unused code may never be loaded into memory at all.

Contents

Responsibilities

Relocating loaders
OS/360 & Derivatives

Dynamic linkers

See also

References

Responsibilities

In Unix, the loader is the handler for the system call `execve()`.[1] The Unix loader's tasks include:

1. validation (permissions, memory requirements etc.);
2. copying the program image from the disk into main memory;
3. copying the command-line arguments on the stack;
4. initializing registers (e.g., the stack pointer);
5. jumping to the program entry point (`_start`).

In Microsoft Windows 7 and above, the loader is the `LdrInitializeThunk` function contained in ntdll.dll, that does the following:

1. initialisation of structures in the DLL itself (i.e. critical sections, module lists);
2. validation of executable to load;
3. creation of a heap (via the function `RtlCreateHeap`);
4. allocation of environment variable block and PATH block;
5. addition of executable and NTDLL to the module list (a doubly-linked list);
6. loading of KERNEL32.DLL to obtain several important functions, for instance `BaseThreadInitThunk`;
7. loading of executable's imports (i.e. dynamic-link libraries) recursively (check the imports' imports, their imports and so on);
8. in debug mode, raising of system breakpoint;
9. initialisation of DLLs;
10. garbage collection;
11. calling `NtContinue` on the context parameter given to the loader function (i.e. jumping to `RtlUserThreadStart`, that will start the executable)

Relocating loaders

Some operating systems need relocating loaders, which adjust addresses (pointers) in the executable to compensate for variations in the address at which loading starts. The operating systems that need relocating loaders are those in which a program is not always loaded into the same location in the address space and in which pointers are absolute addresses rather than offsets from the program's base address. Some well-known examples are IBM's OS/360 for their System/360 mainframes, and its descendants, including z/OS for the z/Architecture mainframes.

OS/360 & Derivatives

In OS/360 and descendant systems, the (privileged) operating system facility is called IEWFETCH,[2] and is an internal component of the OS Supervisor, whereas the (non-privileged) LOADER application can perform many of the same functions, plus those of the Linkage Editor, and is entirely external to the OS Supervisor (although it certainly uses many Supervisor services).

IEWFETCH utilizes highly specialized channel programs, and it is theoretically possible to load and to relocate an entire executable within one revolution of the DASD media (about 16.6 ms maximum, 8.3 ms average, on "legacy" 3,600 rpm drives). For load modules which exceed a track in size, it is also possible to load and to relocate the entire module without losing a revolution of the media.

IEWFETCH also incorporates facilities for so-called overlay structures, and which facilitates running potentially very large executables in a minimum memory model (as small as 44 KB on some versions of the OS, but 88 KB and 128 KB are more common).

The OS's nucleus (the always resident portion of the Supervisor) itself is formatted in a way that is compatible with a stripped-down version of IEWFETCH. Unlike normal executables, the OS's nucleus is "scatter loaded": parts of the nucleus are loaded into different portions of memory; in particular, certain system tables are required to reside below the initial 64 KB, while other tables and code may reside elsewhere.

The system's Linkage Editor application is named IEWL.[3] IEWL's main function is to associate load modules (executable programs) and object modules (the output from, say, assemblers and compilers), including "automatic calls" to libraries (high-level language "built-in functions"), into a format which may be most efficiently loaded by IEWFETCH. There are a large number of editing options, but for a conventional application only a few of these are commonly employed.

The load module format includes an initial "text record", followed immediately by the "relocation and/or control record" for that text record, followed by more instances of text record and relocation and/or control record pairs, until the end of the module.

The text records are usually very large; the relocation and/or control records are small as IEWFETCH's three relocation and/or control record buffers are fixed at 260 bytes (smaller relocation and/or control records are certainly possible, but 260 bytes is the maximum possible, and IEWL ensures that this limitation is complied with, by inserting additional relocation records, as required, before the next text record, if necessary; in this special case, the sequence of records may be: ..., text record, relocation record, ..., control record, text record, ...).

A special byte within the relocation and/or control record buffer is used as a "disabled bit spin" communication area, and is initialized to a unique value. The Read CCW for that relocation and/or control record has the Program Controlled Interrupt bit set. The processor is thereby notified when that CCW has been accessed by the channel via a special IOS exit. At this point the processor enters the "disabled bit spin" loop (sometimes called "the shortest loop in the world"). Once that byte has been changed from its initialized value, the processor instantaneously knows reading of the relocation and/or control record has been completed, the bit spin is terminated, and relocation may proceed in real-time, and synchronously with the rotation of the media (relocation occurs during the "gap" within the media between the relocation and/or control record and the next text record). If *so*, after relocation of that text record is completed, the NOP CCW will be changed to a TIC, and loading and relocating will proceed using the next buffer. If *not*, then the channel will encounter the NOP CCW, and the channel program will terminate, until it is restarted by IEWFETCH via another special IOS exit. The three buffers are in a continuous circular queue, each pointing to its next, and the last pointing to the first, and three buffers are constantly reused as loading and relocating proceeds.

IEWFETCH can, thereby, load and relocate a load module of any practical size, and in the minimum possible time.

Dynamic linkers

Dynamic linking loaders are another type of loader that load and link shared libraries (like .so files or .dll files) to already loaded running programs.

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)
- Linker (computing)
- Name decoration
- Prebinding
- Prelinking
- Relocation (computer science)
- Relocation table
- Static library
- gold (linker)
- prelink
- Bug compatibility

References

1. "exec" (http://www.opengroup.org/onlinepubs/000095399/functions/exec.html). *The Open Group Base Specifications Issue 6, IEEE Std 1003.1, 2004 Edition*. The Open Group. Retrieved 2008-06-23.