NAME

ld.so, ld-linux.so* - dynamic linker/loader

SYNOPSIS

The dynamic linker can be run either indirectly by running some dynamically linked program or library (in which case no command-line options to the dynamic linker can be passed and, in the ELF case, the dynamic linker which is stored in the **.interp** section of the program is executed) or directly by running:

*/lib/ld-linux.so.** [OPTIONS] [PROGRAM [ARGUMENTS]]

DESCRIPTION

The programs **ld.so** and **ld-linux.so*** find and load the shared libraries needed by a program, prepare the program to run, and then run it.

Linux binaries require dynamic linking (linking at run time) unless the **-static** option was given to ld(1) during compilation.

The program **ld.so** handles a.out binaries, a format used long ago; **ld-linux.so*** handles ELF (*/lib/ld-linux.so.1* for libc5, */lib/ld-linux.so.2* for glibc2), which everybody has been using for years now. Otherwise, both have the same behavior, and use the same support files and programs ldd(1), ldconfig(8), and */etc/ld.so.conf*.

When resolving library dependencies, the dynamic linker first inspects each dependency string to see if it contains a slash (this can occur if a library pathname containing slashes was specified at link time). If a slash is found, then the dependency string is interpreted as a (relative or absolute) pathname, and the library is loaded using that pathname.

If a library dependency does not contain a slash, then it is searched for in the following order:

- (ELF only) Using the directories specified in the DT_RPATH dynamic section attribute of the binary if present and DT_RUNPATH attribute does not exist. Use of DT_RPATH is deprecated.
- Using the environment variable **LD_LIBRARY_PATH**. Except if the executable is a set-user-ID/set-group-ID binary, in which case it is ignored.
- (ELF only) Using the directories specified in the DT_RUNPATH dynamic section attribute of the binary if present.
- From the cache file */etc/ld.so.cache*, which contains a compiled list of candidate libraries previously found in the augmented library path. If, however, the binary was linked with the **-z nodeflib** linker option, libraries in the default library paths are skipped. Libraries installed in hardware capability directories (see below) are preferred to other libraries.
- In the default path */lib*, and then */usr/lib*. If the binary was linked with the **-z nodeflib** linker option, this step is skipped.

Rpath token expansion

ld.so understands certain strings in an rpath specification (DT_RPATH or DT_RUNPATH); those strings are substituted as follows

ORIGIN (or equivalently {*ORIGIN*})

This expands to the directory containing the application executable. Thus, an application located in *somedir/app* could be compiled with

gcc -Wl,-rpath,$ORIGIN/../lib

so that it finds an associated shared library in *somedir/lib* no matter where *somedir* is located in the directory hierarchy. This facilitates the creation of turn-key applications that do not need to be installed into special directories, but can instead be unpacked into any directory and still find their own shared libraries.

LIB (or equivalently {*LIB*})
This expands to *lib* or *lib64* depending on the architecture (e.g., on x86-64, it expands to *lib64* and on x86-32, it expands to *lib*).

PLATFORM (or equivalently {*PLATFORM*})
This expands to a string corresponding to the processor type of the host system (e.g., x86_64). On some architectures, the Linux kernel doesn't provide a platform string to the dynamic linker. The value of this string is taken from the **AT_PLATFORM** value in the auxiliary vector (see getauxval(3)).

OPTIONS

--list List all dependencies and how they are resolved.

--verify
Verify that program is dynamically linked and this dynamic linker can handle it.

--library-path PATH
Use PATH instead of **LD_LIBRARY_PATH** environment variable setting (see below).

--inhibit-rpath LIST
Ignore RPATH and RUNPATH information in object names in LIST. This option is ignored if **ld.so** is set-user-ID or set-group-ID.

--audit LIST
Use objects named in LIST as auditors.

HARDWARE CAPABILITIES

Some libraries are compiled using hardware-specific instructions which do not exist on every CPU. Such libraries should be installed in directories whose names define the required hardware capabilities, such as */usr/lib/sse2/*. The dynamic linker checks these directories against the hardware of the machine and selects the most suitable version of a given library. Hardware capability directories can be cascaded to combine CPU features. The list of supported hardware capability names depends on the CPU. The following names are currently recognized:

Alpha ev4, ev5, ev56, ev6, ev67

MIPS loongson2e, loongson2f, octeon, octeon2

PowerPC
4xxmac, altivec, arch_2_05, arch_2_06, booke, cellbe, dfp, efpdouble, efpsingle, fpu, ic_snoop, mmu, notb, pa6t, power4, power5, power5+, power6x, ppc32, ppc601, ppc64, smt, spe, ucache, vsx

SPARC
flush, muldiv, stbar, swap, ultra3, v9, v9v, v9v2

s390 dfp, eimm, esan3, etf3enh, g5, highgprs, hpage, ldisp, msa, stfle, z900, z990, z9-109, z10, zarch

x86 (32-bit only)
acpi, apic, clflush, cmov, cx8, dts, fxsr, ht, i386, i486, i586, i686, mca, mmx, mtrr, pat, pbe, pge, pn, pse36, sep, ss, sse, sse2, tm

ENVIRONMENT

Among the more important environment variables are the following:

LD_ASSUME_KERNEL
(glibc since 2.2.3) Each shared library can inform the dynamic linker of the minimum kernel ABI version that it requires. (This requirement is encoded in an ELF note section that is viewable via *readelf -n* as a section labeled **NT_GNU_ABI_TAG**.) At run time, the dynamic linker determines the ABI version of the running kernel and will reject loading shared libraries that specify minimum ABI versions that exceed that ABI version.

LD_ASSUME_KERNEL can be used to cause the dynamic linker to assume that it is

running on a system with a different kernel ABI version. For example, the following command line causes the dynamic linker to assume it is running on Linux 2.2.5 when loading the shared libraries required by *myprog*:

$ **LD_ASSUME_KERNEL=2.2.5 ./myprog**

On systems that provide multiple versions of a shared library (in different directories in the search path) that have different minimum kernel ABI version requirements, **LD_ASSUME_KERNEL** can be used to select the version of the library that is used (dependent on the directory search order). Historically, the most common use of the **LD_ASSUME_KERNEL** feature was to manually select the older LinuxThreads POSIX threads implementation on systems that provided both LinuxThreads and NPTL (which latter was typically the default on such systems); see pthreads(7).

LD_BIND_NOT
(glibc since 2.2) Don't update the Global Offset Table (GOT) and Procedure Linkage Table (PLT) when resolving a symbol.

LD_BIND_NOW
(libc5; glibc since 2.1.1) If set to a nonempty string, causes the dynamic linker to resolve all symbols at program startup instead of deferring function call resolution to the point when they are first referenced. This is useful when using a debugger.

LD_LIBRARY_PATH
A colon-separated list of directories in which to search for ELF libraries at execution-time. Similar to the**P ATH** environment variable. Ignored in set-user-ID and set-group-ID programs.

LD_PRELOAD
A list of additional, user-specified, ELF shared libraries to be loaded before all others. The items of the list can be separated by spaces or colons. This can be used to selectively override functions in other shared libraries. The libraries are searched for using the rules given under DESCRIPTION. For set-user-ID/set-group-ID ELF binaries, preload pathnames containing slashes are ignored, and libraries in the standard search directories are loaded only if the set-user-ID permission bit is enabled on the library file.

LD_TRACE_LOADED_OBJECTS
(ELF only) If set to a nonempty string, causes the program to list its dynamic library dependencies, as if run by ldd(1), instead of running normally.

Then there are lots of more or less obscure variables, many obsolete or only for internal use.

LD_AOUT_LIBRARY_PATH
(libc5) Version of **LD_LIBRARY_PATH** for a.out binaries only. Old versions of ld-linux.so.1 also supported **LD_ELF_LIBRARY_PATH**.

LD_AOUT_PRELOAD
(libc5) Version of **LD_PRELOAD** for a.out binaries only. Old versions of ld-linux.so.1 also supported **LD_ELF_PRELOAD**.

LD_AUDIT
(glibc since 2.4) A colon-separated list of user-specified, ELF shared objects to be loaded before all others in a separate linker namespace (i.e., one that does not intrude upon the normal symbol bindings that would occur in the process). These libraries can be used to audit the operation of the dynamic linker. **LD_AUDIT** is ignored for set-user-ID/set-group-ID binaries.

The dynamic linker will notify the audit libraries at so-called auditing checkpoints—for example, loading a new library, resolving a symbol, or calling a symbol from another shared object—by calling an appropriate function within the audit library. For details, see rtld-audit(7). The auditing interface is largely compatible with that provided on

Solaris, as described in its *Linker and Libraries Guide*, in the chapter *Runtime Linker Auditing Interface*.

LD_BIND_NOT
(glibc since 2.1.95) Do not update the GOT (global offset table) and PLT (procedure linkage table) after resolving a symbol.

LD_DEBUG
(glibc since 2.1) Output verbose debugging information about the dynamic linker. If set to **all** prints all debugging information it has, if set to **help** prints a help message about which categories can be specified in this environment variable. Since glibc 2.3.4, **LD_DEBUG** is ignored for set-user-ID/set-group-ID binaries.

LD_DEBUG_OUTPUT
(glibc since 2.1) File in which **LD_DEBUG** output should be written. The default is standard error. **LD_DEBUG_OUTPUT** is ignored for set-user-ID/set-group-ID binaries.

LD_DYNAMIC_WEAK
(glibc since 2.1.91) Allow weak symbols to be overridden (reverting to old glibc behavior). For security reasons, since glibc 2.3.4, **LD_DYNAMIC_WEAK** is ignored for set-user-ID/set-group-ID binaries.

LD_HWCAP_MASK
(glibc since 2.1) Mask for hardware capabilities.

LD_KEEPDIR
(a.out only)(libc5) Don't ignore the directory in the names of a.out libraries to be loaded. Use of this option is strongly discouraged.

LD_NOWARN
(a.out only)(libc5) Suppress warnings about a.out libraries with incompatible minor version numbers.

LD_ORIGIN_PATH
(glibc since 2.1) Path where the binary is found (for non-set-user-ID programs). For security reasons, since glibc 2.4, **LD_ORIGIN_PATH** is ignored for set-user-ID/set-group-ID binaries.

LD_POINTER_GUARD
(glibc since 2.4) Set to 0 to disable pointer guarding. Any other value enables pointer guarding, which is also the default. Pointer guarding is a security mechanism whereby some pointers to code stored in writable program memory (return addresses saved by setjmp(3) or function pointers used by various glibc internals) are mangled semi-randomly to make it more difficult for an attacker to hijack the pointers for use in the event of a buffer overrun or stack-smashing attack.

LD_PROFILE
(glibc since 2.1) The name of a (single) shared object to be profiled, specified either as a pathname or a soname. Profiling output is appended to the file whose name is: *LD_PROFILE_OUTPUT*/*LD_PROFILE*.profile.

LD_PROFILE_OUTPUT
(glibc since 2.1) Directory where **LD_PROFILE** output should be written. If this variable is not defined, or is defined as an empty string, then the default is */var/tmp*. **LD_PROFILE_OUTPUT** is ignored for set-user-ID and set-group-ID programs, which always use */var/profile*.

LD_SHOW_AUXV
(glibc since 2.1) Show auxiliary array passed up from the kernel. For security reasons, since glibc 2.3.5, **LD_SHOW_AUXV** is ignored for set-user-ID/set-group-ID binaries.

LD_USE_LOAD_BIAS
By default (i.e., if this variable is not defined) executables and prelinked shared objects will honor base addresses of their dependent libraries and (nonprelinked) position-independent executables (PIEs) and other shared objects will not honor them. If **LD_USE_LOAD_BIAS** is defined wit the value, both executables and PIEs will honor the base addresses. If **LD_USE_LOAD_BIAS** is defined with the value 0, neither executables nor PIEs will honor the base addresses. This variable is ignored by set-user-ID and set-group-ID programs.

LD_VERBOSE
(glibc since 2.1) If set to a nonempty string, output symbol versioning information about the program if the **LD_TRACE_LOADED_OBJECTS** environment variable has been set.

LD_WARN
(ELF only)(glibc since 2.1.3) If set to a nonempty string, warn about unresolved symbols.

LDD_ARGV0
(libc5) *argv*[0] to be used by ldd(1) when none is present.

FILES

/lib/ld.so
a.out dynamic linker/loader

/lib/ld-linux.so.{*1*,*2*}
ELF dynamic linker/loader

/etc/ld.so.cache
File containing a compiled list of directories in which to search for libraries and an ordered list of candidate libraries.

/etc/ld.so.preload
File containing a whitespace-separated list of ELF shared libraries to be loaded before the program.

lib*.so*
shared libraries

NOTES

The **ld.so** functionality is available for executables compiled using libc version 4.4.3 or greater. ELF functionality is available since Linux 1.1.52 and libc5.

SEE ALSO

ld(1), ldd(1), pldd(1), sprof(1), dlopen(3), getauxval(3), rtld-audit(7), ldconfig(8), sln(8)

COLOPHON

This page is part of release 3.74 of the Linux *man-pages* project. A description of the project, information about reporting bugs, and the latest version of this page, can be found at http://www.kernel.org/doc/man-pages/.