RTLD_LAZY

RTLD_LAZY is a flag that you can pass to *dlopen()* when you load a shared object.

Even though the word "lazy" in the name suggests that it's about lazy binding as described above in "Lazy binding," it has different semantics. It makes (semantically) no difference whether a *program* is lazy- or now- bound, but for objects that you load with *dlopen()*, RTLD_LAZY means "there may be symbols that can't be resolved; don't try to resolve them until they're used." This flag currently applies only to function symbols, not data symbols.

What does it practically mean? To explain that, consider a system that comprises an executable X, and shared objects P (primary) and S (secondary). X uses *dlopen()* to load P, and P loads S. Let's assume that P has a reference to *some_function()*, and S has the definition of *some_function()*.

If X opens P without RTLD_LAZY binding, the symbol *some_function()* doesn't get resolved — not at the load time, nor later by opening S. However, if P is loaded with `RTLD_LAZY | RTLD_WORLD`, the runtime linker doesn't try to resolve the symbol *some_function()*, and there's an opportunity for us to call `dlopen("S", RTLD_GLOBAL)` before calling *some_function()*. This way, the *some_function()* reference in P will be satisfied by the definition of *some_function()* in S.

There are several programming models made possible by RTLD_LAZY:

> X uses *dlopen()* to load P and calls a function in P; P determines its own requirements and loads the object with the appropriate implementation. For that, P needs to be opened with RTLD_LAZY. For example, the X server opens a video driver (P), and the video driver opens its own dependencies.
>
> X uses *dlopen()* to load P, and then determines the implementation that P needs to use (e.g. P is a user interface, and S is the "skin" implementation).

Parent topic: Optimizing the runtime linker

Related reference
dlopen()
gdb
ld
pdebug
QCC, qcc