

Re: Alternate file types for RUN ?

Source: <http://unix.derkeiler.com/Newsgroups/comp.os.vms/2006-10/msg02145.html>

- *From:* briggs@xxxxxxxxxxxxxxxxxxxx
 - *Date:* 26 Oct 2006 07:53:55 -0500
-

In article <454048DC.E6B87C82@xxxxxxxxxxxx>, JF Mezei <jfmezei.spamnot@xxxxxxxxxxxx> writes:

Dave Froble wrote:

One of the ideas of transfer vectors in a RTL is to allow replacement of modules in the RTL without re-linking. Have done this many times.

Correct. But the references to the RTL routine symbols/entry points are resolved at link time and are fixed.

The references to the RTL routine symbols/entry points are only partially resolved at link time. The linker can only take the references as far as (roughly speaking) a shareable image name and an offset. Since the location where the shareable image will be mapped is not fixed, the references cannot always be resolved all the way down to either an absolute or a relative virtual address.

At \$ RUN time, the image activator can go back and process the fix-ups, that were prepared by the linker, converting all the inter-image references to actual absolute or relative virtual addresses. It does not consult the target RTL's symbol table to do so.

At \$ LIB\$FIND_IMAGE_SYMBOL time there are no unresolved fixups to process. The programmer explicitly uses the value returned by LIB\$FIND_IMAGE_SYMBOL. LIB\$FIND_IMAGE_SYMBOL needs to consult the target RTL's symbol table to determine this value.

The idea behind the transfer vector is that you can have a structure at the top of the shareable image which is fixed for the lifetime of the OS where symbols are defined. When you call LIB\$EAT_CHOCOLATE, your main image has been told by the linker to branch to a fixed location within that shareable image. At that location (which happens to be within the transfer vector), you find another branch instruction which redirects you to some "random" location in the shareable image where the actual code for LIB\$EAT_CHOCOLATE resides.

Re: Alternate file types for RUN ?

However, when you link your application, the entry point of LIB\$EAT_CHOCOLATE is fully resolved at LINK time.

As above, no, it isn't fully resolved at LINK time. It is only resolved down to an image name and an offset.

The idea behind the transfer vector is to allow the entry point offsets within a shareable image section to be fixed without having to lock down the layout of the bulk of the image section. (Yes, this matches what you just said).

If you are using LIB\$IMAGE_FIND_SYMBOL on the same shareable image, the LINKER resolves nothing (because LIB\$EAT_CHOCOLATE is merely a string that is passed as an argument to LIB\$FIND_IMAGE_SYMBOL, so the linker doesn't need to resolve this). It is all done at run time.

Yes.

And, as you point out below, this means that references that are created by the linker and fixed up by the image activator are somewhat more static than references that are produced by LIB\$FIND_IMAGE_SYMBOL.

Linker references use the run time address range where the relevant image section is mapped and the link time offset within that section where the relevant symbol pointed.

LIB\$FIND_IMAGE_SYMBOL references use the run time address range where the relevant image section is mapped and the run time offset within that section where the relevant symbol points.

So while this works with shareable images that have transfer vectors, you don't actually need transfer vectors when using LIB\$FIND_IMAGE_SYMBOL because if you update the shareable image, the next invocation of the main image will resolve to whatever new address LIB\$EAT_CHOCOLATE happens to be at in the new version.