Anatomy of the C Library#

After the kernel/process manager the C library is the heart and soul of Neutrino applications.

Where is it?#

The C library (henceforce known as libc) lives with all of the common libraries in /lib/c part of the tree.

What is in it?#

The browsing the content of libc is like going to a great buffet. It has content ranging the from the client stubs for the Neutrino kernel calls to the intricate server bindings for resource managers to the mundane code for min() (yes, there really is a function). Something for everyone!

Major sections of libc include:

- POSIX/C99 functionality (ie stdio, string, ...)
- Resource manager framework (ie thread pools, dispatch framework, ...)
- Special server functionality (ie system logging, pci, ...)
- OS compatability functions (QNX4, BSD, Solaris, Linux, ...)
- Application runtime loader, startup and profiling
- Kernel call stubs

How is it organized?#

The top level of the libc contains directories that allow some "intuitive" navigation and classification.

1, 1[a-j], ansi, xopen, stdio, string, time
: These directories have content that match the named specifications or parts of the specifications (POSIX provisional specs are what the 1* directories represent in case you were curious). This is handy if you know your way around the POSIX spec, but more or less a crapshoot if you don't.

iofunc, resmgr, dispatch
: These are the core of the resource manager framework. Responsible for the management, decoding and dispatch of standard messages as well as the default POSIX filesystem behaviours.

kercalls, kercover
: Kernel calls and cover functions for kernel calls. The kernel calls are generated using a script during the build of the C library.

misc, unix, watcom
: Compatability and historical support for various odds and ends.

qnx, services, support
: General QNX'ism and support for QNX services live here. The heart of the distributed namespace resolution, central to every name based operation - *open(), can be found in these directories.

ldd, prof, startup
: These directories contain the runtime loader, the gear for the dl* functions, the profiling and startup stubs.

The other few token directories not listed contain the header files (public ones in "public" and libc private ones in "inc") as well as the main build directory for the library "lib".

What is interesting about this is the build process for libc. In order to maintain a relatively "svelt" figure for the general C library, we have specifically relegated certain functionalities to the static version of the library.

High runner functionality is maintained in the shared version of the library while discreet functions often found only in one or two applications (like login()) are shifted off to the static library. You can see the full configuration if you take a peek at the contents of the lib/c/lib/common.mk file.

What can I do with it?#

Without a doubt, trolling the source is cool and exciting, but you want more!

Probably the most refreshing thing that you can now do is to take advantage of the source being available to help debug those issues where you really wanted to get into libc to find out what was going on.

If this is the case, then the easiest thing to is to bring the libc source into your application!

1. Checkout the libc source
2. Locate the functions you are interested in
3. Symlink (assuming you aren't running on a dumb OS) those files to your project
4. Add the lib/c/inc header file to your include paths (if required)
5. Rebuild your program with debugging enabled as usual
6. Go forth and walk through the libc code!

Fast and Easy!

Of course if you really want to go all the way, then you should really build a custom libc shared object with full debug information (so.g variants in the tree) and join the ranks of true developers who will attach a debugger to just about anything and get into its guts.

Hey ... who knows, maybe doing this will mean you stop passing NULL into printf() and calling up to us and tell us that we are breaking your application 'cause the stack trace tells you so!