NAME

rrdthreads − Provisions for linking the RRD library to use in multi−threaded programs

SYNOPSIS

Using librrd in multi-threaded programs requires some extra precautions, as the RRD library in its original form was not thread-safe at all. This document describes requirements and pitfalls on the way to use the multi-threaded version of librrd in your own programs. It also gives hints for future RRD development to keep the library thread-safe.

Currently only some RRD operations are implemented in a thread-safe way. They all end in the usual "`_r`" suffix.

DESCRIPTION

In order to use librrd in multi-threaded programs you must:

- Link with *librrd_th* instead of *librrd* (use `-lrrd_th` when linking)
- Use the "`_r`" functions instead of the normal API-functions
- Do not use any at-style time specifications. Parsing of such time specifications is terribly non-thread-safe.
- Never use non *`_r` functions unless it is explicitly documented that the function is tread-safe.
- Every thread SHOULD call `rrd_get_context()` before its first call to any `librrd_th` function in order to set up thread specific data. This is not strictly required, but it is the only way to test if memory allocation can be done by this function. Otherwise the program may die with a SIGSEGV in a low-memory situation.
- Always call `rrd_error_clear()` before any call to the library. Otherwise the call might fail due to some earlier error.

NOTES FOR RRD CONTRIBUTORS

Some precautions must be followed when developing RRD from now on:

- Only use thread-safe functions in library code. Many often used libc functions aren't thread-safe. Take care in the following situations or when using the following library functions:
  - Direct calls to `strerror()` must be avoided: use `rrd_strerror()` instead, it provides a per-thread error message.
  - The `getpw*`, `getgr*`, `gethost*` function families (and some more `get*` functions) are not thread-safe: use the *`_r` variants
  - Time functions: `asctime`, `ctime`, `gmtime`, `localtime`: use *`_r` variants
  - `strtok`: use `strtok_r`
  - `tmpnam`: use `tmpnam_r`
  - Many others (lookup documentation)
- A header file named *rrd_is_thread_safe.h* is provided that works with the GNU C−preprocessor to ''poison'' some of the most common non-thread-safe functions using the `#pragma GCC poison` directive. Just include this header in source files you want to keep thread-safe.
- Do not introduce global variables!

  If you really, really have to use a global variable you may add a new field to the `rrd_context` structure and modify *rrd_error.c*, *rrd_thread_safe.c* and *rrd_non_thread_safe.c*
- Do not use `getopt` or `getopt_long` in *`_r` (neither directly nor indirectly).

  `getopt` uses global variables and behaves badly in a multi-threaded application when called concurrently. Instead provide a *_r function taking all options as function parameters. You may provide argc and **argv arguments for variable length argument lists. See `rrd_update_r` as an example.

- Do not use the `rrd_parsetime` function!

  It uses lots of global variables. You may use it in functions not designed to be thread-safe, like in functions wrapping the `_r` version of some operation (e.g., `rrd_create`, but not in `rrd_create_r`)

CURRENTLY IMPLEMENTED THREAD SAFE FUNCTIONS

Currently there exist thread-safe variants of `rrd_update`, `rrd_create`, `rrd_dump`, `rrd_info`, `rrd_last`, and `rrd_fetch`.

AUTHOR

Peter Stamfest <peter@stamfest.at>