MSOpenTech's Redis on Windows

We strive to have a stable, functionally equivalent and comparably performing version of Redis on Windows. We have achieved performance nearly identical to the POSIX version running head-to-head on identical hardware across the network. Aside from feature differences that help Redis take advantage of the Windows infrastructure, our version of Redis should work in most situations with the identical setup and configuration that one would use on a POSIX operating system.

How is Redis on Windows implemented?

Redis is a C code base that compiles under Visual Studio. Most of the code compiles with only minor changes (due to syntactical differences between compilers and low-level API differences on Windows). There are a few areas where there are significant differences in how efficient Windows programs operate relative to POSIX programs. We have encapsulated most these differences in a platform specific library. The areas where there are significant differences are:

- Networking APIs
- POSIX File Descriptors
- POSIX fork()
- Logging
- Windows Services API

Networking Differences

The Windows networking stack is split between user mode code and kernel mode code. Transitions between user and kernel mode are expensive operations. The POSIX networking APIs on Windows utilize a programming model that incurs significant performance loss due to the kernel/user mode transitions. Efficient Windows networking code instead uses the IO Completion Port model to reduce the impact of this behavior. The APIs used and the programming model for IO Completion is different enough that we were forced to implement a new networking layer in Redis.

File Descriptors

In a POSIX operating system, all data sources (files, pipes, sockets, mail slots, etc.) are referenced in code with a handle called a file descriptor. These are low value integers that increment by one with each successive file descriptor creation. All POSIX APIs that work with file descriptors will function without the programmer having to know what kind of data source a file descriptor represents. On Windows, each kind of data source has a separate kind of HANDLE. APIs that work with one HANDLE type will not work with another kind of HANDLE. In order to make Redis operate with its assumptions about file descriptor values and data source agnosticism, we implemented a Redis File Descriptor API layer.

fork()

The POSIX version of Redis uses the fork() API. There is no equivalent in Windows, and it is an exceedingly difficult API to completely simulate. For most of the uses of fork() we have used Windows specific programming idioms to bypass the need to use a fork()-like API. The one case where we could not do so was with the point-in-time heap snapshot behavior that the Redis persistence model is based on. We tried several different approaches to work around the need for a fork()-like API, but always ran into significant performance penalties and stability issues.

Our current approach is to simulate the point-in-time snapshot behavior aspect of fork() without doing a complete simulation of fork(). We do this with the system-paging file that contains the Redis heap. When a fork() operation is required we do the following:

- Mark every page in the heap with the Copy on Write page protection
- Start a child process and pass it the handle to the heap mapped to the system paging file
- Signal the child to start the AOF or RDB persistence process on the memory shared via the system paging file
- Wait (asynchronously) for the child process to finish
- Map the changes in the Redis heap that occurred during the fork() operation back into the parent process heap.

The upside with this implementation is that our performance and stability is now on par with the POSIX version of Redis. The down side is that we have a runtime system paging file space requirement for Redis. A system-paging file at least of the size of physical memory is suggested.

The default configuration of Windows allows the page file to grow to 3.5 times the size of physical memory. There are scenarios where 3rd party programs also compete for system paging file space at runtime.

Logging

In addition to file-based logging, the POSIX version of Redis supports logging via the syslog facility. The equivalent in Windows is the Event Log. With the addition of the Windows Service code we have added support for logging to the Event Log. We have mapped the –syslogxxxx flags for this purpose.

Windows Service

In version 2.8.9 we added support to make Redis operate as a service. See the "Windows Service Documentation.docx" file included with the GitHub binary distribution for a description of the service commands available.

Redis on Windows Best Practices

Binary Distributions

The GitHub repository should be considered a work in progress until we release the NuGet and Chocolatey packages and tag the repository at that released version.

For instance, the Windows Service feature has taken many iterations with community input to get right. Since the initial Windows service code was checked we have added the following to the service based on community input:

- Preshutdown notification in order to clean up the memory mapped file consistently.
- Code to identify and clean up orphaned memory mapped files left behind when a machine running Redis as a service crashes or loses power.
- Self-elevation of the Redis executable so that service commands would work from a non-elevated command prompt.
- Service naming so that multiple instances of the Redis service could be installed on one machine.

- Automatically adjusting folder permissions so that when Redis is run under the NETWORK SERVICE account it could modify the files in the installation directory.
- Moved all of the pre-main() error reporting code (service and quasi-fork code) that could write errors to stdout to use the Redis logging code. This allows service initialization errors to reach the Event Log. This required intercepting all of the command line and conf file arguments before main() in order to properly initialize the logging engine. There were several fixes related to the intricacies of how to interpret the arguments passed to Redis.

Heap Sizing

Native heaps are prone to fragmentation. If we are not able to allocate more heap space due to fragmentation, Redis will flag the problem and exit. Unlike the POSIX version of Redis, our heap size is constrained by the system paging file space. It is important to consider how much data you are expecting to put into Redis, and how much fragmentation you are likely to see in the Redis heap. High levels of fragmentation of the heap may cause to run into an out of heap space. In this case, an increase of the size of the system-paging file may solve the problem.

Installation and Maintenance

Since Redis uses the system-paging file, the most stable configurations will only have Redis running on essentially a virgin operating system install.

Redis is xcopy deployable. There should be no problem upgrading versions by simply copying new binaries over old ones (assuming they are not currently in use).

Service Account

When using Redis as a Windows service, the default installation configures Redis to run under the system's NETWORK SERVICE account. There are some environments where another account must be used (perhaps a domain service account). Configuration of this account needs to be done manually at this point with the service control manager. If this is done, it is also important to give read/write/create permission to the folder that the Redis executable is in to this user identity.