IoRing vs. io_uring: a comparison of Windows and Linux implementations

windows-internals.com/ioring-vs-io_uring-a-comparison-of-windows-and-linux-implementations

By Yarden Shafir

A few months ago I wrote this post about the introduction of I/O Rings in Windows. After publishing it a few people asked for a comparison of the Windows I/O Ring and the Linux `io_uring`, so I decided to do just that. The short answer – the Windows implementation is almost identical to the Linux one, especially when using the wrapper function provided by helper libraries. The long answer is what I'll be covering in the rest of this post.
The information about the `io_uring` implementation was gathered mostly from here – a paper documenting the internal implementation and usage of `io_uring` on Linux and explaining some of the reasons for its existence and the way it was built.
As I said, the basic implementation of both mechanisms is very similar – both are built around a submission queue and a completion queue that have shared views in both user and kernel address spaces. The application writes the requested operation data into the submission queue and submits it to the kernel, which processes the requested number of entries and writes the results into the completion queue. In both cases there is a maximum number of allowed entries per ring and the completion queue can have up to However, there are some differences in the internal structures as well as the way the application is expected to interact with the I/O ring.

Initialization and Memory Mapping

One such difference is the initialization stage and mapping of the queues into user space: on Windows the kernel fully initializes the new ring, including the creation of both queues and creating a shared view in the application's user-mode address space, using an `MDL`.
However, in the Linux `io_uring` implementation, the system creates the requested ring and the queues but does not map them into user space. The application is expected to call `mmap(2)` using the appropriate file descriptors to map both queues into its address space, as well as the `SQE` array, which is separate from the main queue.
This is another difference worth noticing – on Linux the completion ring (or queue) directly contains the array of `CQE`s, but the submission ring does not. Instead, the `sqes` field in the submission ring is a pointer to another memory region containing the array of `SQE`s, that has to be mapped separately. To index this array, the `sqring` has an additional array field which contains the index into the `SQE`s array. Not being a Linux expert, I won't try to explain the reasoning behind this design and will simply quote the reasoning given in the paper mentioned above:

> *This might initially seem odd and confusing, but there's some reasoning behind it. Some applications may embed request units inside internal data structures, and this allows them the flexibility to do so while retaining the ability to submit multiple sqes in one operation. That in turns allows for easier conversion of said applications to the io_uring interface.*

On Windows there are only two important regions since the `SQE`s are part of the submission ring. In fact both rings are allocated by the system in the same memory region so there is only one shared view between the user and kernel space, containing two separate rings.
One more difference exists when creating a new I/O ring: on Linux the number of entries in a submission ring can be between `1` and `0x1000` (`4096`) while on Windows it can be between `1` and `0x10000`, but at least `8` entries will always be allocated. In both cases the completion queue will have twice the number of entries as the submission queue. There is one small difference regarding the exact number of entries requested for the ring: For technical reasons the number of entries in both rings has to be a power of two. On Windows, the system takes the requested ring size and aligns it to the nearest power of two to receive the actual size that will be used to allocate the ring memory. On Linux the system does not do that, and the application is expected to request a size that is a power of two.

Versioning

Windows puts far more focus on compatibility than Linux does, putting a lot of effort into making sure that when a new feature ships, applications using it will be able to work properly across different Windows builds even as the feature changes. For that reason, Windows implements versioning for its structures and features and Linux does not. Windows also implements I/O rings in phases, marked by those versions, where the first versions only implemented read operations, the next version will implement write and flush operations, and so on. When creating an I/O ring the caller needs to pass in a version to indicate which version of I/O rings it wants to use.
On Linux, however, the feature was implemented fully from the beginning and does not require versioning. Also, Linux doesn't put as much focus on compatibility and users of `io_uring` are expected to use and support the latest features.

Waiting for Operation Completion

On both Windows and Linux the caller can choose to not wait on the completion of events in the I/O ring and simply get notified when all operations are complete, making this feature fully asynchronous. In both systems the caller can also choose to wait on all events in a fully synchronous way, specifying a timeout in case processing the events takes too long. Everything in between is the area where the systems differ.
On Linux, a caller can request a wait on the completion of a specific number of operations in the ring, a capability Windows doesn't allow. This capability allows applications to start processing the results after a certain amount of operations were completed, instead of

waiting for all of them. In newer builds Windows did add a similar yet slightly more limited option – registering a notification event that will be set when the first entry in the ring gets completed to signal to the waiting application that it's safe to start processing the results now.

Helper Libraries

In both systems it is possible for an application to manage its rings itself through system calls. This is an option that's accepted on Linux and highly discouraged on Windows, where the NT API is undocumented and officially should not be used by non-Microsoft code. However, in both systems most applications have no need to manage the rings themselves and a lot of a generic ring management code can be abstracted and managed by a separate component. This is done through helper libraries – `KernelBase.dll` on Windows and `liburing` on Linux.
Both libraries export generic functionality like creating, initializing and deleting an I/O ring, creating submission queue entries, submitting a ring and getting a result from the completion queue.
Both libraries use very similar functions and data structures, making the task of porting code from one platform to the other much easier.

Conclusion

The implementation of I/O rings on Windows is so similar to the Linux `io_uring` that it looks like some headers were almost copied from the `io_uring` implementation. There are some differences between the two features, mostly due to philosophical differences between the two systems and the role and responsibilities they give the user. The Linux `io_uring` was added a couple of years ago, making it a more mature feature than the new Windows implementation, though still a relatively young one and not without issues. It will be interesting to see where these two features will go in the future and what parity will exist in them in a few years.