**🔄 THE EVENT LOOP in XRootD**

**A high-level overview 🌟**

*[Clarifications marked with blue]* 😉

The event loop is a mechanism for getting feedback from the kernel [I assume that you are referring to a TCP kernel here] whether

- there is space in the TCP output buffer for writing

- there are data in the TCP receive buffer for reading

[we decide on the size of the buffer from the arguments given inside functions like Read(), Write(), from the XrdCl namespace, right?]

now, in the application there are two quests:

1) the queue of requests the client wants to issue to the server, whenever there is a write event we take a request from the queue and write it to the socket, note: the TCP buffer might not be big enough to accommodate the whole request, so it might be several write events will be needed to write a single request

1.1) every request is accompanied by a message handler, after the request is fully written to the socket, the message handler is being moved to the 2nd queue for the incoming responses. [so, one can imagine the response queue as being composed of a stack with pairs of Response-id/Message\_Handler-id ?]

2) whenever the event loop yields a reading event it means we can read something from the socket, so we read out a server response, again it might be one will need several read events to read a full server response. [we know when to read something from the socket using the epoll system call on that file descriptor which was opened once the corresponding request was sent to the server, right?]

2.1) once we read a full server response, we look in the queue for incoming responses for a matching message handler:

* once we have the handler, after parsing the response, we can call the user callback [the callback is more or less a function that runs whenever we get its corresponding response from the server, right?]
* mind that we are executing the callback in a thread pool [not sure what the implications of the thread pool are: a limited/fixed number of execution workers to process the callbacks and responses from the server? Is that it 😉?]
* the syscall that is used for monitoring file descriptors is epoll [epoll is “watching” the descriptor that is started once we initiate the TCP connection to the server?]
* [from my understanding, a file descriptor is started/created when creating an interface between the client-space, that is the xrootd client, and the kernel-space, that is the TCP kernel which allows a connection between client and server]

**Event-loop stages:**

The event loop generates ‘ready to read’, ‘ready to write’, ‘read timeout’ and ‘write timeout’ events. Accordingly, to those events, the client is sending requests and receiving responses, and handling timeouts. For connections using TLS, it is more complicated, but I think this is irrelevant for the paper.

Diagram

Description automatically generated