🔄 THE EVENT LOOP in XRootD

A high-level overview 🌟

The in-draft text:

A user might want to retrieve some data using the XRootD client from a file that is located on a server. Interaction between the XRootD client and that particular server is done through a TCP implementation. The mechanism that allows the client to receive feedback from the TCP kernel is called event-loop. The feedback consists of communicating whether there is available space in the TCP-output buffer for writing data (i.e. requests which will be sent to the server) or if there is some data in the TCP-receive buffer for reading responses from the server. In this event-based workflow, there is a queue of requests that the client is issuing to the server, and with each write-event, a request is removed from the queue and it is being written on the socket. It is worth mentioning that the TCP buffers (for both sending and receiving data) might not have enough size to allow requests/responses to be written/read in a single event, meaning that it can take several write/read events to process an entire request to the server or a response from the server. Furthermore, each request has a corresponding message handler, so that after a request is written to the socket (in order to be sent to the server), the accompanying message handler is moved into a queue for incoming responses. During a read-event yielded by the event-loop, the client is informed that it can read from the socket, that is a server response. Once a response arrived from the server, its corresponding message handler (located inside the incoming queue) is also taken out from the queue, and finally, after the response is parsed, the callback function is being called.

[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. If so, there is a socket on the client and also a socket on the server, right?] whether

- there is space in the TCP output buffer for writing

- there are data in the TCP receive buffer for reading

[size of buffer is established on the client-server handshake, right? And moreover, they can differ in size?]

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, and with each successful request sending to the server, the Message\_Handler moves into the response stack?]

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 – as the XRootD client – 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 [Is epoll “watching” the file descriptor that is started once we initiate the TCP connection to the server? Or with each request that we create to the server, a new FD starts. In the second case, that would imply a “Stack” of descriptors which the event-loop]
* [from my understanding, a file descriptor is started/created when creating an interface between the client-space 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