IEEE Standards Interpretation for IEEE Std 1003.1™-1990 IEEE Standard for Information Technology--Portable Operating System Interfaces (POSIX®)

Interpretation Request #130
Topic: 1003.1q **Relevant Sections:** 21.3.14 PASC

Extracting filters from trace logs: If the stream full policy is POSIX_TRACE_LOOP, it may happen, that the latest POSIX_TRACE_FILTER event is overwritten and that there is no POSIX_TRACE_FILTER event left in the trace stream. posix_trace_get_filter() doesn't work on trace logs (Section 21 lines 1378 ...1382). Thus, we can't reconstruct the event filters from the trace log. They are neither readable from an POSIX_TRACE_FILTER event (because there is no such event) nor from the meta data of the trace log (because there is no interface). Is this a correct interpretation of the standard?

Interpretation Response
The original question is not clear as to whether the concern is over a POSIX_TRACE_FILTER event being overwritten in a trace stream (because the policy is POSIX_TRACE_ LOOP) or being lost from a trace log (when the trace stream policy is POSIX_TRACE_ FLUSH and a log overrun occurs and events are lost from the log, one of which may be a POSIX_TRACE_FILTER event). We need some clarification on that. The former is not really an issue, because the filters associated with an active trace stream are retrievable out of band (posix_ trace_get_filter()).

Therefore lets assume the latter (note however that POSIX_TRACE_LOOP policy plays no part in this unless the application is explicitly attempting to keep the trace stream flushed to a trace log with calls to posix_trace_flush(), since POSIX_TRACE_LOOP is not required to ever flush on its own): It is a correct interpretation that you can't reconstruct the trace filters from the trace log if it contains any POSIX_TRACE_OVERFLOW events, since one of the lost events may have been a POSIX_TRACE_FILTER event. This does not represent a defect in the standard, only an implementation's failure to be able to flush a highly active (FLUSHING) trace stream to a trace log without losing events, while the

trace controller is changing the set of events filtered.

Rationale for Interpretation
Francois Riche's response follows: There are two levels of storage to memorize trace event identifiers and their arguments: - the first one is the trace stream, we can guess this one is real memory with small size and play the buffer role if it is associated with a trace log, - the second one is the trace log, we can guess this one is mass-storage, and the size is big enough considering the duration of tracing or the volume of trave events to memorize.

Therefore, the strategy for these two types of trace event storage are generally different. Maybe as you imagine, the trace stream is looping, and the associated trace log is enough big to memorize your tracing sample. We guess the rate of trace events is not too high and the trace stream can be flushed to the trace log and all trace events can be flushed in the trace log before the trace stream loops. Because the POSIX_TRACE_FILTER is a trace event, when the trace stream is terminated, you can open the trace log, read the trace events, especially the POSIX_TRACE_FILTER events and know the set of trace event types even if this one has changed during the tracing sample.

On the other hand, this is right, these trace events are lost in the trace stream, but stored in the trace log. In the case of a trace stream associated with a trace log, as you can read only the trace log, so you can retrieve the filter information. Forwarded to Interpretations Group: 27 Feb 2001 Proposed resolution: 21 March 2001 Finalized: April 5 2001