BENCHMARKING LIBEVENT AGAINST LIBEV

Top

2011-01-11, Version 6

This document briefly describes the results of running the libevent benchmark program against both libevent and libev.

Libevent Overview

Libevent (first released in 2000-11-14) is a high-performance event loop that supports a simple API, two event types (which is can either I/O+timeout or signal+timeout) and a number of "backends" (select, poll, epoll, kqueue and /dev/poll at the time of this writing).

Algorithmically, it uses red-black trees to organise timers and doubly linked lists for other types. It can have at most one read and one write watcher per file descriptor or signal. It also offers a simple DNS resolver and server and HTTP server and client code, as well as a socket buffering abstraction.

Libev Overview

Libev (first released in 2007-11-12) is also a high-performance event loop, supporting eight event types (I/O, real time timers, wall clock timers, signals, child status changes, idle, check and prepare handlers).

It uses a priority queue to manage timers and uses arrays as fundamental data structure. It has no artificial limitations on the number of watchers waiting for the same event. It offers an emulation layer for libevent and optionally the same DNS, HTTP and buffer management (by reusing the corresponding libevent code through its emulation layer).

Benchmark Setup

The benchmark is very simple: first a number of socket pairs is created, then event watchers for those pairs are installed and then a (smaller) number of "active clients" send and receive data on a subset of those sockets.

The benchmark program used was bench.c, taken from the libevent distribution, modified to collect total time per test iteration, optionally enable timeouts on the event watchers as well as optionally using the native libev API and to output the times differently.

For libevent, version 1.4.3 was used, while for libev, version 3.31 was used. Both libraries and the test program were compiled by gcc version 4.2.3 with the optimisation options `-O3 -fno-guess-branch-probability -g` and run on a Core2 Quad at 3.6GHz with Debian GNU/Linux (Linux version 2.6.24-1). Both libraries were configured to use the epoll interface (the most performant interface available in either library on the test machine).

The same benchmark program was used to both run the libevent vs. libevent emulation benchmarks (the same code paths/source lines were executed in this case) and run the the native libev API benchmark (with different codepaths, but functionally equivalent).

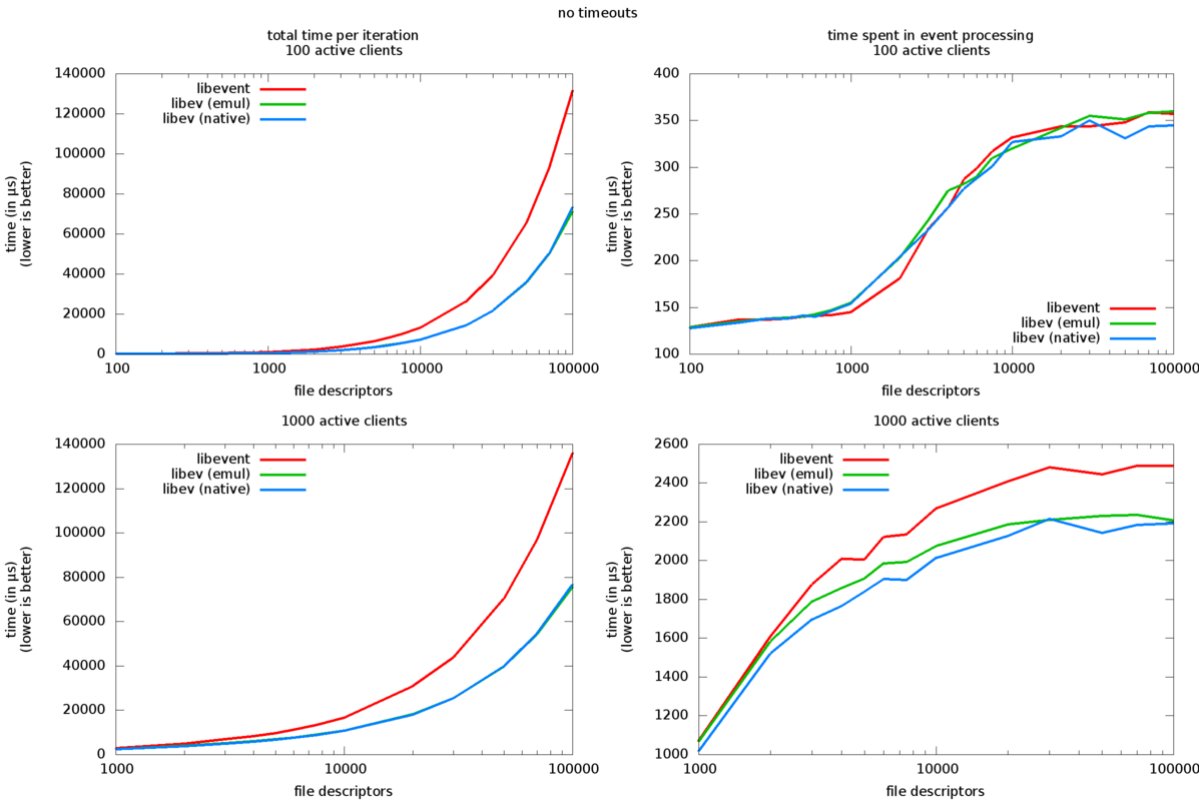
The difference between the libevent and libev+libevent emulation versions is strictly limited to the use of different header files (event.h from libevent, or the event.h emulation from libev).

Each run of the benchmark program consists of two iterations outputting the total time per iteration as well as the time used in handling the requests only. The program was run for various total numbers of file descriptors. Each run is composed of six individual runs, and the result used is the minimum time from these runs, for the second iteration of the test.

The test program was run on its own cpu, with realtime priority, to achieve stable timings.

First Benchmark: no timeouts, 100 and 1000 active clients

Without further ado, here are the results (click for larger version):



The left two graphs show the overall time spent for setting up watchers, preparing the sockets and polling for events, while the right two graphs only include the actual poll processing. The top row represents 100 active clients (clients doing I/O), the bottom row uses 1000. All graphs have a logarithmic fd-axis to sensibly display the large range of file descriptor numbers of 100 to 100000 (in reality, its actually socket pairs, thus there are actually twice the number of file descriptors in the process).

Discussion

The total time per iteration increases much faster for libevent than for libev, taking almost twice as much time than libev regardless of the number of clients. Both exhibit similar growth characteristics, though.

The polling time is also very similar, with libevent being consistently being slower in the 1000-fd case, and virtually identical timings in the 100-fd case. The absolute difference, however, is small (less than 5%).

The native API timings are consistently better than the emulation API, but the absolute difference again is small.

Interpretation

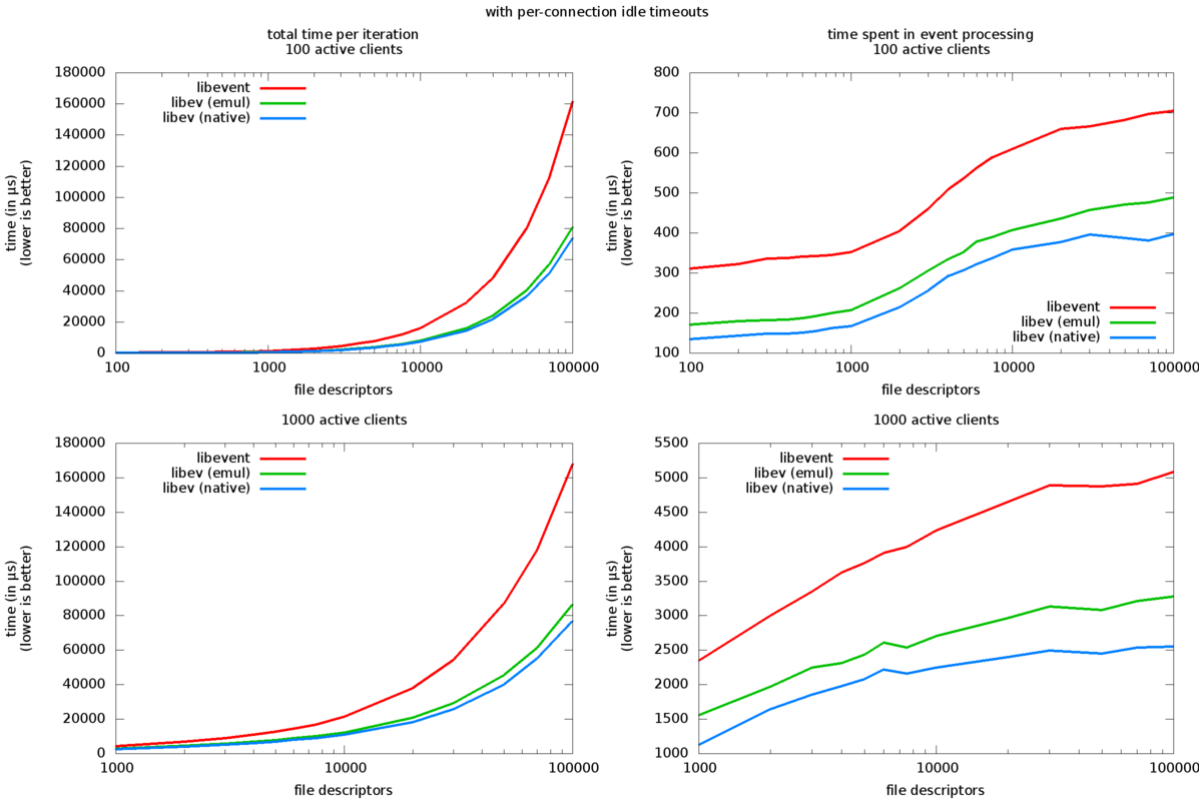
The cost for setting up or changing event watchers is clearly much higher for libevent than for libev, and API differences cannot account for this (the difference between native API and emulation API in libev is very small). This is important in practise, as the libevent API has no good interface to change event watchers or timeouts on the fly.

At higher numbers of file descriptors, libev is consistently faster than libevent.

Also, the libevent API emulation itself only results in a small overhead in this case.

Second Benchmark: idle timeouts, 100 and 1000 active clients

Again, the results first (click for larger version):



The graph layout is identical to the first benchmark. The difference is that this time, there is a random timeout attached to each socket. This was done to mirror real-world conditions where network servers usually need to maintain idle timeouts per connection. Those idle timeouts to be reset on activity. This is implemented by setting a random 10 to 11 second timeout during set-up, and deleting/re-adding the event watcher each time a client receives data.

Discussion

The graphs have both changed dramatically. The total time per iteration has increased dramatically for libevent, but has increased only slightly for the libev curves. The difference between native and emulated API has become more apparent.

The event processing graphs look very different now, with libev being consistently faster (with a factor of two to almost three) over the whole range of file descriptor numbers. The growth behaviour exhibited is roughly similar, but much lower for libev than for libevent.

As for libev alone, the native API is consistently faster than the emulated API, and the difference is noticable compared to the first benchmark, but still relatively small when compared to the difference between libevent and libev, regarding poll times. The overall time has been almost halved, however.

Interpretation

Both libev and libevent use a binary heap for timer management (earlier versions of libevent used a red-black tree), which explains the similar growth characteristics. Apparently, libev makes better use of the binary heap than libevent, even with identical API calls (note that the upcoming 3.33 release of libev, not used in this benchmark, uses a cache-aligned 4-heap and benchmarks consistently faster than 3.31).

Another reason for the higher performance, especially in the set-up phase, might be that libevent calls the epoll_ctl syscall on each change (twice per fd for del/add), while libev only sends changes to the kernel before the next poll (EPOLL_MOD).

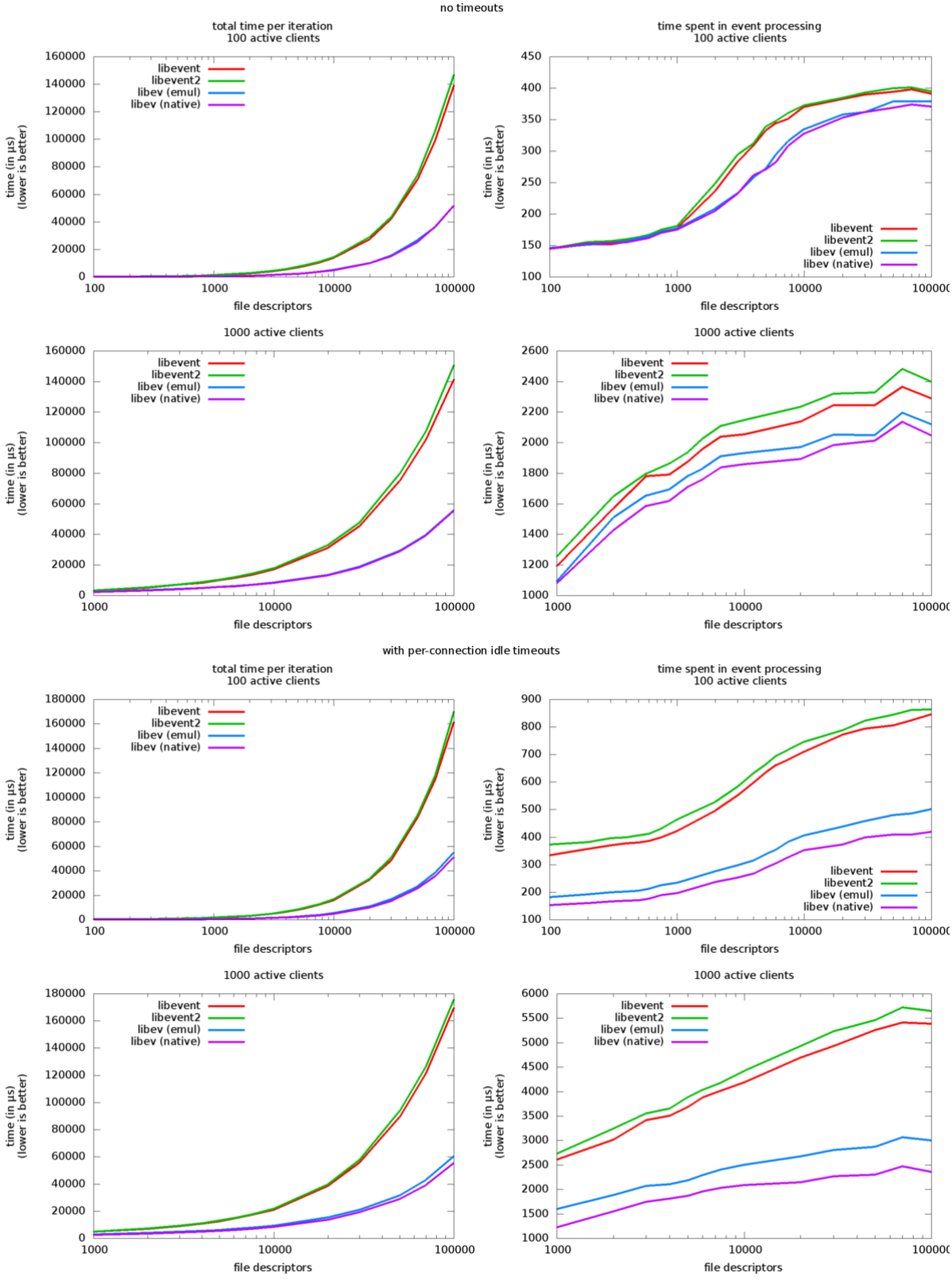
The native API is noticably faster in this case (almost twice as fast overall). The most likely reason is again timer management, as libevent uses two O(log n) operations, while libev needs a single and simpler O(log n) operation.

Summary

The benchmark clearly shows that libev has much lower costs and is consequently faster than libevent. API design issues also play a role in the results as the native API can do much better than the emulated API when timers are being used. Even though this puts libev at a disadvantage (the emulation layer has to emulate some aspects of libevent that its native API does not. It also has to map each libevent watcher to three of its own watchers, and has to run thunking code to map from those three watchers to the libevent user code, due to the different structure of their callbacks). It is still much faster than libevent even when using the libevent emulation API.

Appendix: Benchmark graphs for libev 4.03, libevent 1.4.13, libevent 2.0.10

Here are the benchmark graphs, redone with more current versions. Nothing drastic has changed, libevent2 seems to be a tiny bit slower (probably due to the extra thread locking), libev a tiny bit faster.



Author/Contact

Marc Alexander Lehmann <libev@schmorp.de>