Give io_uring a Chance!

Kyoung (lseo@uwaterloo.ca)

2024-07-23 Tue

Outline

Asynchronous IO in Linux

`io_uring` concepts

Examples

Conclusion

Readiness vs Completion

epoll

- Since Linux 2.5
- Is the socket ready for IO?

AIO

- subsumed by . . .

`io_uring`

- Since Linux 5.1
- Was the kernel able to complete the IO on the socket?

Basic Concepts

Submission Queue, Completion Queue

Ring Provided Buffers

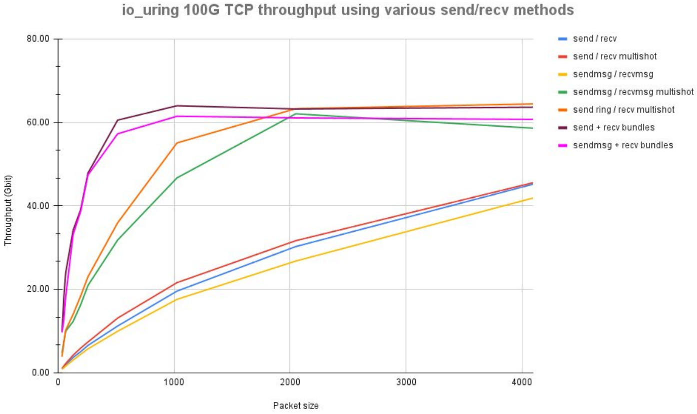
Multishot

Bundle

Sources of Potential Performance Gains

- “Batching”
  →
  less poll rearming?
  latency throughput tradeoff?
- Less system calls
- Less memcpy

proxy server

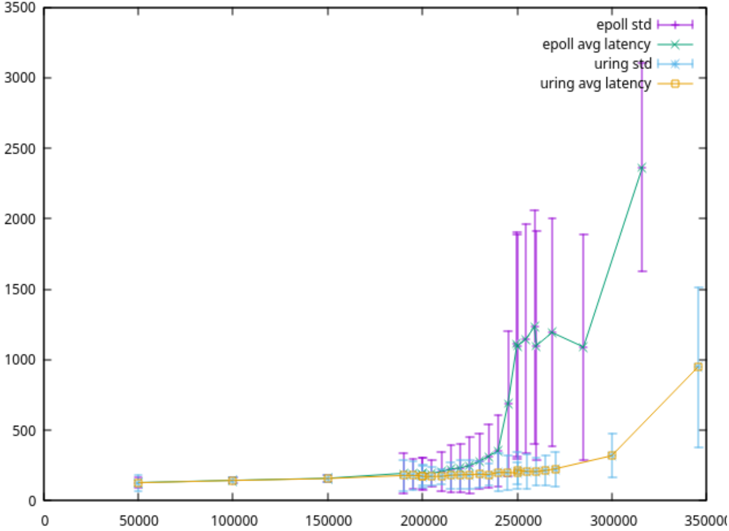


- multishot + provided buffers + bundles
- drastically reduced memory copies leading to high throughput

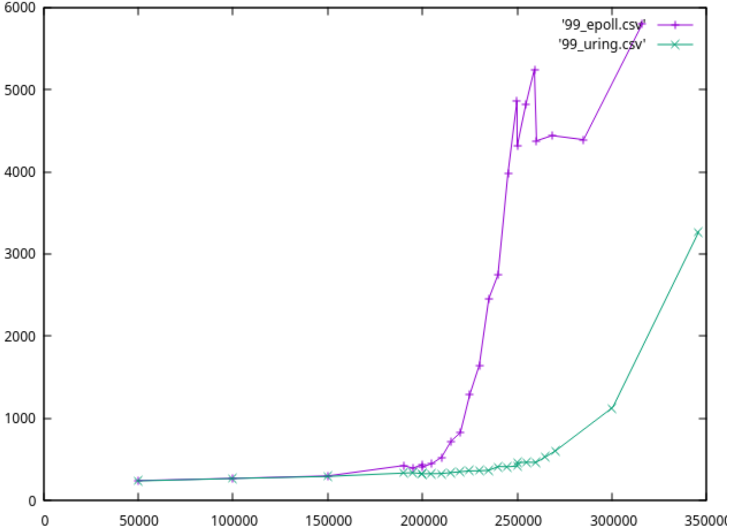
memcached

What would happen if we replace the epoll based event loop with `io_uring` ?

memcached


3500
3000
2500
2000
1500
1000
500
0
0
50000
100000
150000
200000
250000
300000
35000
epoll std
epoll avg latency
uring std
uring avg latency

memcached


6000
5000
4000
3000
2000
1000
0
0
50000
100000
150000
200000
250000
300000
35000(
'99_epoll.csv'
'99_uring.csv'

memcached

- 15% increase in throughput
- 50% reduction in 99 percentile tail latency

  . . . we are still coming up with explanations :D

Conclusion

Reasons to use `io_uring`

- Continuously being improved by very talented engineers at Meta
- Potential massive performance gain under certain scenarios

Challenges

- Lack of examples that demonstrate idioms
- Can be challenging to (re)design application logic and state machines

References

- https://github.com/axboe/liburing/blob/master/examples/proxy.c
- https://github.com/axboe/liburing/wiki/io_uring-and-networking-in-2023
- https://github.com/axboe/liburing/wiki/What’s-new-with-io_uring-in-6.10
- https://git.uwaterloo.ca/lseo/memcached-io_uring