

SAMBA EXPERIENCE

io_uring

Status Update within Samba

Stefan Metzmacher <metze@samba.org>

Samba Team / SerNet

2023-05-10

<https://samba.org/~metze/presentations/2023/SambaXP/>

Topics

- ▶ What is io-uring?
- ▶ io-uring for Samba
- ▶ Performance research, prototyping and ideas
- ▶ The road to upstream
- ▶ Future Improvements
- ▶ Questions? Feedback!

Last Status Updates (SDC 2020 / SDC 2021)

- ▶ I gave a similar talk at the storage developer conference 2020:
 - ▶ See <https://samba.org/~metze/presentations/2020/SDC/>
 - ▶ It explains the milestones and design up to Samba 4.13 (in detail)
- ▶ I gave a similar talk at the storage developer conference 2021:
 - ▶ See <https://samba.org/~metze/presentations/2021/SDC/>
 - ▶ It explains the milestones and updates up to Samba 4.15 (in detail)

Last Status Updates (SDC 2020 / SDC 2021)

- ▶ I gave a similar talk at the storage developer conference 2020:
 - ▶ See <https://samba.org/~metze/presentations/2020/SDC/>
 - ▶ It explains the milestones and design up to Samba 4.13 (in detail)
- ▶ I gave a similar talk at the storage developer conference 2021:
 - ▶ See <https://samba.org/~metze/presentations/2021/SDC/>
 - ▶ It explains the milestones and updates up to Samba 4.15 (in detail)

What is io-uring? (Part 1)

- ▶ Linux 5.1 introduced a new scalable AIO infrastructure
 - ▶ It's designed to avoid syscalls as much as possible
 - ▶ kernel and userspace share mmap'ed rings:
 - ▶ submission queue (SQ) ring buffer
 - ▶ completion queue (CQ) ring buffer
 - ▶ See "[Ring in a new asynchronous I/O API](#)" on LWN.NET
- ▶ This can be nicely integrated with our async tevent model
 - ▶ It may delegate work to kernel threads
 - ▶ It seems to perform better compared to our userspace threadpool
 - ▶ It can also inline non-blocking operations

What is io-uring? (Part 1)

- ▶ Linux 5.1 introduced a new scalable AIO infrastructure
 - ▶ It's designed to avoid syscalls as much as possible
 - ▶ kernel and userspace share mmap'ed rings:
 - ▶ submission queue (SQ) ring buffer
 - ▶ completion queue (CQ) ring buffer
 - ▶ See "[Ring in a new asynchronous I/O API](#)" on LWN.NET
- ▶ This can be nicely integrated with our async tevent model
 - ▶ It may delegate work to kernel threads
 - ▶ It seems to perform better compared to our userspace threadpool
 - ▶ It can also inline non-blocking operations

io-uring for Samba (Part 1)

- ▶ Between userspace and filesystem (available from 5.1):
 - ▶ IORING_OP_READV, IORING_OP_WRITEV and IORING_OP_FSYNC
 - ▶ Supports buffered and direct io
 - ▶ IORING_OP_FSETXATTR, IORING_OP_FGETXATTR (from 5.19)
 - ▶ IORING_OP_GETDENTS, under discussion, but seems to be tricky
 - ▶ IORING_OP_FADVISE (from 5.6)
- ▶ Path based syscalls with async impersonation (from 5.6)
 - ▶ IORING_OP_OPENAT2, IORING_OP_STATX
 - ▶ Using IORING_REGISTER_PERSONALITY for impersonation
 - ▶ IORING_OP_UNLINKAT, IORING_OP_RENAMEAT (from 5.10)
 - ▶ IORING_OP_MKDIRAT, IORING_OP_SYMLINKAT, IORING_OP_LINKAT (from 5.15)
 - ▶ IORING_OP_SETXATTR, IORING_OP_GETXATTR (from 5.19)

io-uring for Samba (Part 1)

- ▶ Between userspace and filesystem (available from 5.1):
 - ▶ IORING_OP_READV, IORING_OP_WRITEV and IORING_OP_FSYNC
 - ▶ Supports buffered and direct io
 - ▶ IORING_OP_FSETXATTR, IORING_OP_FGETXATTR (from 5.19)
 - ▶ IORING_OP_GETDENTS, under discussion, but seems to be tricky
 - ▶ IORING_OP_FADVISE (from 5.6)
- ▶ Path based syscalls with async impersonation (from 5.6)
 - ▶ IORING_OP_OPENAT2, IORING_OP_STATX
 - ▶ Using IORING_REGISTER_PERSONALITY for impersonation
 - ▶ IORING_OP_UNLINKAT, IORING_OP_RENAMEAT (from 5.10)
 - ▶ IORING_OP_MKDIRAT, IORING_OP_SYMLINKAT, IORING_OP_LINKAT (from 5.15)
 - ▶ IORING_OP_SETXATTR, IORING_OP_GETXATTR (from 5.19)

io-uring for Samba (Part 2)

- ▶ Between userspace and socket (and also filesystem) (from 5.8)
 - ▶ IORING_OP_SENDMSG, IORING_OP_RECVMSG
 - ▶ Improved MSG_WAITALL support (5.12, backported to 5.11, 5.10)
 - ▶ Maybe using IOSQE_ASYNC in order to avoid inline memcpy
 - ▶ IORING_OP_SPLICE, IORING_OP_TEE
 - ▶ IORING_OP_SENDMSG_ZC, zero copy with an extra completion (from 6.1)
 - ▶ IORING_OP_GET_BUF, under discussion to replace IORING_OP_SPLICE

vfs_io_uring in Samba 4.12 (2020)

- ▶ With Samba 4.12 we added "io_uring" vfs module
 - ▶ For now it only implements SMB_VFS_PREAD,PWRITE,FSYNC_SEND/RECV
 - ▶ It has less overhead than our pthreadpool default implementations
 - ▶ I was able to speed up a smbclient 'get largefile /dev/null'
 - ▶ Using against smbd on loopback
 - ▶ The speed changes from 2.2GBytes/s to 2.7GBytes/s
- ▶ The improvement only happens by avoiding context switches
 - ▶ But the data copying still happens:
 - ▶ From/to a userspace buffer to/from the filesystem/page cache
 - ▶ The data path between userspace and socket is completely unchanged
 - ▶ For both cases the cpu is mostly busy with memcpy

vfs_io_uring in Samba 4.12 (2020)

- ▶ With Samba 4.12 we added "io_uring" vfs module
 - ▶ For now it only implements SMB_VFS_PREAD,PWRITE,FSYNC_SEND/RECV
 - ▶ It has less overhead than our pthreadpool default implementations
 - ▶ I was able to speed up a smbclient 'get largefile /dev/null'
 - ▶ Using against smbd on loopback
 - ▶ The speed changes from 2.2GBytes/s to 2.7GBytes/s
- ▶ The improvement only happens by avoiding context switches
 - ▶ But the data copying still happens:
 - ▶ From/to a userspace buffer to/from the filesystem/page cache
 - ▶ The data path between userspace and socket is completely unchanged
 - ▶ For both cases the cpu is mostly busy with memcpy

Performance research (SMB2 Read)

- ▶ In October 2020 I was able to do some performance research
 - ▶ With 100Gbit/s interfaces and two NUMA nodes per server.
- ▶ At that time I focussed on the SMB2 Read performance only
 - ▶ We had limited time on the given hardware
 - ▶ We mainly tested with fio.exe on a Windows client
 - ▶ Linux kernel 5.8.12 on the server
- ▶ More verbose details can be found here:
 - ▶ <https://lists.samba.org/archive/samba-technical/2020-October/135856.html>

Performance research (SMB2 Read)

- ▶ In October 2020 I was able to do some performance research
 - ▶ With 100Gbit/s interfaces and two NUMA nodes per server.
- ▶ At that time I focussed on the SMB2 Read performance only
 - ▶ We had limited time on the given hardware
 - ▶ We mainly tested with fio.exe on a Windows client
 - ▶ Linux kernel 5.8.12 on the server
- ▶ More verbose details can be found here:
 - ▶ <https://lists.samba.org/archive/samba-technical/2020-October/135856.html>

Performance research (SMB2 Read)

- ▶ In October 2020 I was able to do some performance research
 - ▶ With 100Gbit/s interfaces and two NUMA nodes per server.
- ▶ At that time I focussed on the SMB2 Read performance only
 - ▶ We had limited time on the given hardware
 - ▶ We mainly tested with fio.exe on a Windows client
 - ▶ Linux kernel 5.8.12 on the server
- ▶ More verbose details can be found here:
 - ▶ <https://lists.samba.org/archive/samba-technical/2020-October/135856.html>

Performance with MultiChannel, sendmsg()

4 connections, ~3.8 GBytes/s, bound by >500% cpu in total, sendmsg() takes up to 0.5 msec

```
mp ~ 01:43:18 up 2 days, 44 min, 7 users, load average: 5.42, 9.22, 1.52
Threads: 823 total, 33 running, 798 sleeping, 0 stopped, 0 zombie
%cpu(s): 0.0 us, 6.3 sy, 0.0 ni, 93.4 id, 0.0 wa, 0.1 hi, 0.2 si, 0.0 st
i/o Mem : 191624.1 total, 182280.4 free, 2617.5 used, 4726.1 buff/cache
i/o Swp: 1824.8 total, 1824.8 free, 0.0 used, 185648.1 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME	COMMAND
307372	root	20	0	2426196	62800	19104	R	96.0	0.0	0:52.24	nsmbd
307406	root	20	0	2426196	63400	19104	R	14.3	0.0	0:06.90	nsmbd
307412	root	20	0	2426196	65256	19104	R	14.0	0.0	0:06.92	nsmbd
307405	root	20	0	2426196	63164	19104	R	13.6	0.0	0:06.84	nsmbd
307416	root	20	0	2426196	64464	19104	R	13.5	0.0	0:06.79	nsmbd
307414	root	20	0	2426196	65520	19104	R	13.5	0.0	0:06.88	nsmbd
307422	root	20	0	2426196	64952	19104	R	13.5	0.0	0:06.78	nsmbd
307432	root	20	0	2426196	71582	19104	R	13.6	0.0	0:06.66	nsmbd
307408	root	20	0	2426196	63936	19104	R	13.3	0.0	0:06.58	nsmbd
307411	root	20	0	2426196	64992	19104	R	13.3	0.0	0:06.77	nsmbd
307413	root	20	0	2426196	65256	19104	R	13.3	0.0	0:06.68	nsmbd
307415	root	20	0	2426196	65520	19104	R	13.3	0.0	0:06.63	nsmbd
307418	root	20	0	2426196	66040	19104	R	13.3	0.0	0:06.69	nsmbd
307419	root	20	0	2426196	67184	19104	R	13.3	0.0	0:06.84	nsmbd
307428	root	20	0	2426196	67632	19104	R	13.3	0.0	0:06.78	nsmbd
307421	root	20	0	2426196	68160	19104	R	13.3	0.0	0:06.71	nsmbd
307423	root	20	0	2426196	69400	19104	R	13.3	0.0	0:06.64	nsmbd
307425	root	20	0	2426196	69400	19104	R	13.3	0.0	0:06.58	nsmbd
307428	root	20	0	2426196	70800	19104	R	13.3	0.0	0:06.58	nsmbd
307438	root	20	0	2426196	70800	19104	R	13.3	0.0	0:06.84	nsmbd
307433	root	20	0	2426196	72304	19104	R	13.3	0.0	0:06.61	nsmbd
307426	root	20	0	2426196	70800	19104	R	13.0	0.0	0:06.62	nsmbd
307429	root	20	0	2426196	70800	19104	R	13.0	0.0	0:06.67	nsmbd
307434	root	20	0	2426196	72304	19104	R	13.0	0.0	0:06.70	nsmbd
307425	root	20	0	2426196	72640	19104	R	13.0	0.0	0:06.71	nsmbd
307402	root	20	0	2426196	83672	19104	R	12.8	0.0	0:06.58	nsmbd
307416	root	20	0	2426196	64040	19104	R	12.8	0.0	0:06.68	nsmbd
307417	root	20	0	2426196	64332	19104	R	12.8	0.0	0:06.53	nsmbd
307427	root	20	0	2426196	70800	19104	R	12.8	0.0	0:06.87	nsmbd
307431	root	20	0	2426196	71864	19104	R	12.8	0.0	0:06.58	nsmbd
307424	root	20	0	2426196	69400	19104	R	12.3	0.0	0:06.65	nsmbd
307409	root	20	0	2426196	64200	19104	R	12.0	0.0	0:06.68	nsmbd
307404	root	20	0	2426196	62616	19104	R	11.3	0.0	0:06.61	nsmbd
307103	root	20	0	0	0	0	T	0.3	0.0	0:00.41	ksmorker/s102-2-1a
307382	root	20	0	0	0	0	T	0.3	0.0	0:00.03	ksmorker/221-1-event
307452	root	20	0	62936	3536	3936	R	0.2	0.0	0:00.08	top
1	root	20	0	242512	10952	8176	S	0.0	0.0	0:02.04	system
1	root	20	0	0	0	0	S	0.0	0.0	0:00.13	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	ksmorker/0:00-kblockd
10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
11	root	20	0	0	0	0	S	0.0	0.0	0:00.32	ksftirq/0
12	root	20	0	0	0	0	I	0.0	0.0	0:03.17	rcu_sched
13	root	rs	0	0	0	0	S	0.0	0.0	0:00.03	migration/0
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpupri/0
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpupri/1
16	root	rs	0	0	0	0	S	0.0	0.0	0:01.35	migration/1

The screenshot shows the Windows Task Manager Performance tab. The left sidebar lists system metrics: CPU (8% 2.70 GHz), Memory (10/312 GB (3%)), Ethernet (5.0.3 Mbps R: 31.9 Gbps), and Ethernet (5.40.0 Kbps R: 64.0 Kbps). The Ethernet section is expanded, showing a throughput graph for the Mellanox ConnectX-6 Adapter. The graph shows a peak throughput of 31.9 Gbps. Below the graph, the adapter name is 'SLOT 4 Port 1', connection type is 'Ethernet', IPv4 address is '192.168.0.153', and IPv6 address is 'fe80::af53:8135::ccccca4b%19'. The bottom of the screenshot shows the Resource Monitor window, which displays various system resources like CPU, Memory, Disk, and Network usage for different processes.

IOURING_OP_SENDMSG (Part1)

4 connections, ~6.8 GBytes/s, smbdc only uses ~11% cpu, (io_wqe_work ~50% cpu) per connection, we still use >300% cpu in total

```
top - 05:45:38 up 2 days, 46 min, 2 users, load average: 3.03, 2.04, 1.01
Threads: 823 total, 3 running, 820 sleeping, 0 stopped, 0 zombie
%cpu(s): 0.1 us, 4.7 sy, 0.0 ni, 94.6 id, 0.0 wa, 0.1 hi, 0.5 si, 0.0 st
MiB Mem: 191624.1 total, 182194.6 free, 2702.6 used, 6726.9 buff/cache
MiB Swap: 1024.0 total, 1024.0 free, 0.0 used, 185554.7 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
307577	root	20	0	0	0	0	R	49.0	0.0	0:05.00	io_wqe_worker-0
307549	root	20	0	0	0	0	S	46.0	0.0	0:21.39	io_wqe_worker-0
307555	root	20	0	0	0	0	R	44.0	0.0	0:21.45	io_wqe_worker-0
307567	root	20	0	0	0	0	S	29.0	0.0	0:09.92	io_wqe_worker-1
307558	root	20	0	663100	144024	18004	S	23.2	0.1	0:09.10	smbd
307556	root	20	0	663100	144024	18004	S	19.9	0.1	0:08.95	smbd
307559	root	20	0	663100	144024	18004	S	19.5	0.1	0:08.92	smbd
307563	root	20	0	663100	144024	18004	S	19.5	0.1	0:08.86	smbd
307557	root	20	0	663100	144024	18004	S	19.2	0.1	0:09.11	smbd
307560	root	20	0	663100	144024	18004	S	19.2	0.1	0:09.38	smbd
307561	root	20	0	663100	144024	18004	S	19.2	0.1	0:09.07	smbd
307534	root	20	0	663100	144024	18004	S	10.9	0.1	0:09.00	smbd
307576	root	20	0	663100	144024	18004	S	10.9	0.1	0:05.61	smbd
307562	root	20	0	663100	144024	18004	S	10.5	0.1	0:08.93	smbd
307530	root	20	0	663100	144024	18004	D	11.3	0.1	0:05.16	smbd
307552	root	20	0	0	0	0	S	9.3	0.0	0:12.25	io_wqe_worker-0
417	root	20	0	0	0	0	I	0.3	0.0	0:03.58	kworker/0:2-event
307183	root	20	0	0	0	0	I	0.3	0.0	0:00.61	kworker/u160:2-ml
307568	root	20	0	0	0	0	I	0.3	0.0	0:00.02	kworker/29:0-event
307588	root	20	0	62964	5532	3984	R	0.3	0.0	0:00.12	top
1	root	20	0	242512	10952	8176	S	0.0	0.0	0:02.84	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.13	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H-kblou
10	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
11	root	20	0	0	0	0	S	0.0	0.0	0:00.32	kssoftirqd/0
12	root	20	0	0	0	0	I	0.0	0.0	0:03.17	rcu_sched
13	root	rt	0	0	0	0	S	0.0	0.0	0:00.03	migration/0
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/1
16	root	rt	0	0	0	0	S	0.0	0.0	0:01.38	migration/1
17	root	20	0	0	0	0	S	0.0	0.0	0:00.07	kssoftirqd/1
19	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/1:0H-kblou
21	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/2
22	root	rt	0	0	0	0	S	0.0	0.0	0:01.37	migration/2
23	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kssoftirqd/2
25	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/2:0H-kblou
26	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/3
27	root	rt	0	0	0	0	S	0.0	0.0	0:01.39	migration/3

The screenshot shows a Windows PowerShell window with the following output:

```
complete : 0=0.0%, 4=100.0%, 0=0.1%, 10=0.1%, 32=0.0%, 64=0.0%, >=64=0.0%
Issued rwts: total=64720,0,0,0 short=0,0,0 dropped=0,0,0
latency : target=0, window=0, percentile=100.00%, depth=16

Run status group 0 (all jobs):
REQID: bw=5390KIB/s (5650MB/s), 4096KIB/s-5390KIB/s (4295MB/s-5650MB/s), io=253KIB (2710
PS C:\Users\Administrator> & "cd \Program Files\Foxit Software\Foxit Reader; group_reporting=1 --name=foi
^l --thread --rwread --size=100M --bw=4M --numJobs=1 --time_based=1 --quiet=5m --direct
File_test: (g=0) rwread, bs=(R) 4096KIB-4096KIB, (W) 4096KIB-4096KIB, (T) 4096KIB-4096KIB,
...
File-3.22
Starting 2 threads
Jobs: 2 (F=2): [R(2)][15.3K][r=6816KIB/s][r=1704 IOPS][eta 0m:14s]
```

The Task Manager Performance tab shows the following system metrics:

- CPU: 16% 2.78 GHz
- Memory: 12/312 GB (2%)
- Ethernet: S: 17.4 Mbps R: 57.5 Gbps
- Ethernet: S: 32.0 Kbps R: 96.0 Kbps

The Ethernet section provides detailed throughput information:

- Adapter name: SLOT 4 Port 1
- Connection type: Ethernet
- IPv4 address: 192.168.0.153
- IPv6 address: fe80::fa5a:b155::ccccca4db::19
- Send: 17.4 Mbps
- Receive: 57.5 Gbps

IORING_OP_SENDMSG (Part2)

The major problem still exists, memory copy done by `copy_user_enhanced_fast_string()`

The screenshot shows the Windows Task Manager Performance tab. The system resources are as follows:

- CPU:** 16% 2.78 GHz
- Memory:** 12/512 GB (2%)
- Ethernet:** 15.7 Mbps R: 57.5 Gbps
- Ethernet:** 40.0 Kbps R: 96.0 Kbps

The Ethernet section also displays a throughput graph and the following details:

- Send: 15.7 Mbps
- Receive: 57.5 Gbps
- Adapter name: SLOT 4 Port 1
- Connection type: Ethernet
- IPv4 address: 192.168.0.153
- IPv6 address: fe80::d5a5815

The background of the screenshot shows a Windows PowerShell terminal window with the following output:

```
complete : 0=0.0%, 4=100.0%, 0=0.1%, 16=0.1%, 32=0.0%, 64=0.0%, >64
issued ruts: total=64728,0,0,0 short=0,0,0,0 dropped=0,0,0,0
latency : target=0, window=0, percentile=100.00%, depth=16

Run status group 0 (all jobs):
READ: bw=5396MiB/s (5658MB/s), 4096KiB/s-5396KiB/s (4295MB/s-5658MB/s),
PS C:\Users\Administrator> & 'C:\Program Files\Fio\Fio.exe' --group_report
01 --thread --rwdread --size=100M --bs=4M --numjobs=2 --time_based=1 --run
fio_test: (p=0): rwdread, bs=(R) 4096KiB-4096KiB, (W) 4096KiB-4096KiB, (T)
...
fio-3.22
starting 2 threads
Jobs: 2 (r=2): [R(2)][22.0%][r=6811KiB/s][r=1702 IOPS][eta 03m:54s]
```

IORING_OP_SENDMSG + IORING_OP_SPLICE (Part1)

16 connections, ~8.9 GBytes/s, smbdc ~5% cpu, (io_wq_work 3%-12% cpu filesystem->pipe->socket), only ~100% cpu in total.

The Windows client was still the bottleneck with "Set-SmbClientConfiguration -ConnectionCountPerRssNetworkInterface 16"

The screenshot displays the Windows Task Manager interface. The 'Performance' tab is active, showing system metrics: CPU at 25% (2.70 GHz), Memory at 15.512 GB (7%), and Ethernet at 73.7 Mbps R / 75.1 Gbps. The 'Ethernet' section is expanded, showing a Mellanox ConnectX-6 Adapter with a throughput of 75.1 Gbps. The 'Send' section shows 73.7 Mbps, and the 'Receive' section shows 75.1 Gbps. The adapter name is 'SGT 4 Port 1', connection type is 'Ethernet', IPv4 address is '192.168.0.153', and IPv6 address is 'fe80::a5d31755ccca4b710'. The task manager also shows a list of processes, with 'io_wq_work' processes consuming significant CPU resources.

PID	USER	PR	NI	VIRT	RES	MEM %	SRU	%CPU	MEMBR	TIME+	COMMAND
312117	root	20	0	0	0	0.5	12.3	0.0	0:01:26	16	io_wq_work-9
312109	root	20	0	0	0	0.5	11.0	0.0	0:00:58	16	io_wq_work-9
312125	root	20	0	0	0	0.5	6.6	0.0	0:00:19	16	io_wq_work-9
312826	root	20	0	0	0	0.5	6.6	0.0	0:00:07	16	io_wq_work-9
312836	root	20	0	0	0	0.5	6.6	0.0	0:00:04	16	io_wq_work-9
312132	root	20	0	0	0	0.5	6.0	0.0	0:00:59	16	io_wq_work-1
312135	root	20	0	0	0	0.5	6.0	0.0	0:01:04	16	io_wq_work-9
312122	root	20	0	0	0	0.5	5.6	0.0	0:00:50	16	io_wq_work-1
311994	root	20	0	457660	24080	10424.5	5.3	0.0	0:00:07	smbd	
312879	root	20	0	0	0	0.5	3.0	0.0	0:00:46	16	io_wq_work-9
312892	root	20	0	0	0	0.5	3.0	0.0	0:00:44	16	io_wq_work-9
312108	root	20	0	0	0	0.5	3.0	0.0	0:00:48	16	io_wq_work-9
312106	root	20	0	0	0	0.5	3.0	0.0	0:00:41	16	io_wq_work-9
312109	root	20	0	0	0	0.5	3.0	0.0	0:00:44	16	io_wq_work-9
312112	root	20	0	0	0	0.5	3.0	0.0	0:00:41	16	io_wq_work-9
308304	root	20	0	2986356	108452	54668.5	2.7	0.1	1:10:13	perl	
312895	root	20	0	0	0	0.5	2.7	0.0	0:00:46	16	io_wq_work-9
312115	root	20	0	0	0	0.5	2.7	0.0	0:00:37	16	io_wq_work-9
312145	root	20	0	0	0	0.5	2.7	0.0	0:00:18	16	io_wq_work-1
312862	root	20	0	0	0	0.5	2.3	0.0	0:00:37	16	io_wq_work-9
312869	root	20	0	0	0	0.5	2.3	0.0	0:00:35	16	io_wq_work-9
312183	root	20	0	0	0	0.5	2.3	0.0	0:00:15	16	io_wq_work-9
312151	root	20	0	62964	5532	3084.8	0.2	0.0	0:00:03	tap	
308276	root	20	0	62812	5404	3044.5	0.2	0.0	3:57:64	tap	
310569	root	20	0	0	0	0.1	0.5	0.0	0:00:02	worker/012-event	
311821	root	20	0	0	0	0.1	0.5	0.0	0:00:18	worker/u160:2-m1	
311830	root	20	0	0	0	0.1	0.5	0.0	0:00:18	worker/u160:0-m1	
311894	root	20	0	0	0	0.1	0.5	0.0	0:00:42	worker/u160:3-m1	
1	root	20	0	242512	10952	8176.5	0.0	0.0	0:03:35	systemd	
2	root	20	0	0	0	0.5	0.0	0.0	0:00:20	kthreadd	
3	root	0	-20	0	0	0.1	0.0	0.0	0:00:00	rcu_gp	
4	root	0	-20	0	0	0.1	0.0	0.0	0:00:00	rcu_bh	
8	root	0	-20	0	0	0.1	0.0	0.0	0:00:00	worker/0:00-kb1to	
18	root	0	-20	0	0	0.1	0.0	0.0	0:00:00	ms_perpu_wq	
11	root	20	0	0	0	0.5	0.0	0.0	0:00:39	ksftirq/0	
12	root	20	0	0	0	0.1	0.0	0.0	0:07:94	rcu_sched	
13	root	rt	0	0	0	0.5	0.0	0.0	0:00:05	migration/0	
14	root	20	0	0	0	0.5	0.0	0.0	0:00:00	cpulp/0	
15	root	20	0	0	0	0.5	0.0	0.0	0:00:00	cpulp/1	
16	root	rt	0	0	0	0.5	0.0	0.0	0:01:40	migration/1	
17	root	20	0	0	0	0.5	0.0	0.0	0:00:00	ksftirq/1	
18	root	0	-20	0	0	0.1	0.0	0.0	0:00:00	worker/1:00-kb1lockd	
21	root	20	0	0	0	0.5	0.0	0.0	0:00:00	cpulp/2	
22	root	rt	0	0	0	0.5	0.0	0.0	0:01:40	migration/2	
23	root	20	0	0	0	0.5	0.0	0.0	0:00:01	ksftirq/2	
25	root	0	-20	0	0	0.1	0.0	0.0	0:00:00	worker/2:00-kb1lockd	
36	root	20	0	0	0	0.1	0.0	0.0	0:00:00	cpulp/3	

smbclient IORING_OP_SENDMSG/SPLICE (network)

4 connections, ~11 GBytes/s, smbdc 8.6% cpu, with 4 io-wqework threads (pipe to socket) at ~20% cpu each.

smbclient is the bottleneck here too

```
getting file 1506.dat of size 2097152000 as /dev/null [2771312.2 KiB/bytes/sec] [average 276678.9 KiB/bytes/sec]
getting file 1506.dat of size 2097152000 as /dev/null [3185089.5 KiB/bytes/sec] [average 322387.9 KiB/bytes/sec]
getting file 1506.dat of size 2097152000 as /dev/null [3186123.7 KiB/bytes/sec] [average 321906.6 KiB/bytes/sec]
getting file 1506.dat of size 2097152000 as /dev/null [2824827.2 KiB/bytes/sec] [average 282085.4 KiB/bytes/sec]
getting file 1506.dat of size 2097152000 as /dev/null [3293391.1 KiB/bytes/sec] [average 329082.5 KiB/bytes/sec]
getting file 1506.dat of size 2097152000 as /dev/null [2782888.3 KiB/bytes/sec] [average 278038.3 KiB/bytes/sec]
getting file 1506.dat of size 2097152000 as /dev/null [3236283.4 KiB/bytes/sec] [average 323095.8 KiB/bytes/sec]
getting file 1506.dat of size 2097152000 as /dev/null [3215878.2 KiB/bytes/sec] [average 322392.8 KiB/bytes/sec]
getting file 1506.dat of size 2097152000 as /dev/null [2790190.4 KiB/bytes/sec] [average 282836.8 KiB/bytes/sec]
getting file 1506.dat of size 2097152000 as /dev/null [3185089.5 KiB/bytes/sec] [average 321907.4 KiB/bytes/sec]
getting file 1506.dat of size 2097152000 as /dev/null [2787813.8 KiB/bytes/sec] [average 278094.5 KiB/bytes/sec]
getting file 1506.dat of size 2097152000 as /dev/null [3218793.1 KiB/bytes/sec] [average 322021.8 KiB/bytes/sec]

top - 02:41:58 up 17 days, 17:38, 1 user, load average: 3.87, 0.22, 3.35
tasks: 877 total, 5 running, 872 sleeping, 0 stopped, 0 zombie
rtps(s): 0.1 us, 4.4 sy, 0.0 ni, 93.5 id, 0.0 wa, 0.0 hi, 1.7 si, 0.0 st
Mem Mem : 388888.7 total, 327333.7 free, 3813.5 used, 68941.4 buff/cache
Mem Swap: 1828.0 total, 732.0 free, 287.0 used, 131866.0 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR  %CPU  %MEM    time+  Command
 298180 root      20   0 375680 37680 18852  99.2  0.0  9:25.55 smbclient
 298185 root      20   0 375664 36180 17816  99.8  0.0  8:30.87 smbclient
 298187 root      20   0 375682 36880 18896  88.1  0.0  8:44.80 smbclient
 298188 root      20   0 375652 37096 18208  86.4  0.0  8:49.28 smbclient
 388100 root      20   0 31540 7872 3412.5  2.0  0.0 180:03.15 ktop
      230 root      20   0 0 0 0  0.5 1.3  0.0  5:56.39 ksoftirqd/45
 298176 root      20   0 249536 8076 5130.5  1.3  0.0  9:13.28 lftp

top - 02:41:57 up 3 days, 21:43, 5 users, load average: 1.11, 0.09, 0.82
tasks: 877 total, 1 running, 876 sleeping, 0 stopped, 0 zombie
rtps(s): 0.1 us, 3.4 sy, 0.0 ni, 97.6 id, 0.0 wa, 0.1 hi, 0.9 si, 0.0 st
Mem Mem : 388824.1 total, 377248.5 free, 3895.5 used, 13328.1 buff/cache
Mem Swap: 1828.0 total, 1824.0 free, 0.0 used, 188675.2 avail Mem

  PID USER      PR  NI  VIRT  RES  SHR  %CPU  %MEM    time+  Command
388116 root      20   0 0 0 0  0.5 21.2  0.0  7:51.81 io_wqeworker-0
388133 root      20   0 0 0 0  0.5 20.3  0.0  0:53.32 io_wqeworker-0
388139 root      20   0 0 0 0  0.5 17.9  0.0  0:40.39 io_wqeworker-0
388121 root      20   0 0 0 0  0.5 17.3  0.0  0:38.48 io_wqeworker-0
388116 root      20   0 458888 21264 17852.5  8.6  0.0  0:40.53 smbd

smplc: 706 of mount 'cyclic', 4900 KiB, (root count approx.): 35388328236 lost: 0/18 drop: 0/21096
overhead: shared object | symbol
  3.00% [kernel] | [k] do_tcp_sendpages
  2.93% [kernel] | [k] raw_spin_lock_bh
  4.80% [kernel] | [k] copy_page_to_iter
  3.29% [kernel] | [k] page_cache_pipe_buf_release
  3.05% [kernel] | [k] __x86_rtpoline_ras
  3.04% [kernel] | [k] page_cache_pipe_buf_confirm
  2.89% [kernel] | [k] native_sendmsg_spin_lock_slowpath
  2.89% [kernel] | [k] show_file_root_iter
  2.79% [kernel] | [k] inet_sendpage
  2.63% [kernel] | [k] tcp_sendpage

For a higher level overview, try: perf top --sort comm,dso
```

smbclient IORING_OP_SENDMSG/SPLICE (loopback)

8 connections, ~22 GBytes/s, smbdc 22% cpu, with 4 io.wqe_work threads (pipe to socket) at ~22% cpu each.

smbclient is the bottleneck here too, it triggers the memory copy done by copy_user_enhanced_fast_string()

```
getting file \\smb.dat of size 2097152000 as /dev/null (2945758.8 Kbytes/sec) leverage 2945758.8 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2710797.3 Kbytes/sec) leverage 2041657.3 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2951888.2 Kbytes/sec) leverage 2079457.6 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2201643.2 Kbytes/sec) leverage 2201643.2 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3107738.5 Kbytes/sec) leverage 2950864.5 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2909736.8 Kbytes/sec) leverage 2714162.3 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2808334.4 Kbytes/sec) leverage 2808334.4 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3117188.0 Kbytes/sec) leverage 3080262.3 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2960838.6 Kbytes/sec) leverage 2949964.1 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3000335.4 Kbytes/sec) leverage 2943473.4 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2743218.8 Kbytes/sec) leverage 2049012.6 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3002932.1 Kbytes/sec) leverage 2082954.5 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3118717.7 Kbytes/sec) leverage 2951515.8 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3000049.0 Kbytes/sec) leverage 2951515.8 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2251839.2 Kbytes/sec) leverage 2231748.8 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2217381.9 Kbytes/sec) leverage 2292968.6 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2902548.2 Kbytes/sec) leverage 2944263.8 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3001655.1 Kbytes/sec) leverage 2943726.7 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3001655.1 Kbytes/sec) leverage 2942525.3 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3001741.7 Kbytes/sec) leverage 2011805.4 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3107738.5 Kbytes/sec) leverage 2940074.4 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3116293.9 Kbytes/sec) leverage 2951072.3 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2723487.9 Kbytes/sec) leverage 2231748.8 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2902548.2 Kbytes/sec) leverage 2945095.8 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2958385.8 Kbytes/sec) leverage 2799964.1 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3117188.0 Kbytes/sec) leverage 2940074.4 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3117188.0 Kbytes/sec) leverage 2042425.7 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2945283.2 Kbytes/sec) leverage 2020559.8 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2511884.4 Kbytes/sec) leverage 2011805.4 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3001655.1 Kbytes/sec) leverage 2094346.1 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2723487.9 Kbytes/sec) leverage 2232666.5 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2777312.2 Kbytes/sec) leverage 2790097.3 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3114148.8 Kbytes/sec) leverage 2040441.8 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3114148.8 Kbytes/sec) leverage 2940074.4 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2504048.4 Kbytes/sec) leverage 2942472.7 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (3001655.1 Kbytes/sec) leverage 2957105.8 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2978343.8 Kbytes/sec) leverage 2078306.8 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2978343.8 Kbytes/sec) leverage 2095262.7 Kbytes/sec
getting file \\smb.dat of size 2097152000 as /dev/null (2042425.7 Kbytes/sec) leverage 2231219.8 Kbytes/sec
```

```
copy -d 10:30:30 up 4 days, 21:02, 0 users, load average: 0.15, 0.39, 0.44
tasks: 812 total, 34 running, 903 sleeping, 0 stopped, 0 zombie
(Mem): 0.3 vs. 31.2 sy, 0.0 mi, 00.1 id, 0.0 wa, 0.0 hi, 2.1 si, 0.0 st
MiB Mem: 381024.1 total, 170825.4 free, 3316.2 used, 13382.0 buff/cache
MiB Swap: 1024.0 total, 1024.0 free, 0.0 used, 100403.2 avail Mem
```

pid	user	ppid	ni	pid	uid	gid	svu	spid	smem	time	command
322783	root	20	0	326733	root	12708	0	22.8	0.0	1:26.29	io_uring
322784	root	20	0	300036	root	12708	0	01.5	0.0	1:26.30	smbclient
322785	root	20	0	300040	root	12708	0	01.5	0.0	1:26.30	smbclient
322786	root	20	0	376244	root	19468	0	79.8	0.0	1:23.73	smbclient
322787	root	20	0	376236	root	12720	79.8	0.0	0.0	1:24.42	smbclient
322781	root	20	0	376243	root	12702	79.8	0.0	0.0	1:24.74	smbclient
322786	root	20	0	300040	root	19468	0	79.8	0.0	1:25.83	smbclient
322789	root	20	0	376140	root	12732	79.1	0.0	0.0	1:24.33	smbclient
322822	root	20	0	0	0	22.8	0.0	0.0	0.0	0:14.00	io_wqe_worker-0
322827	root	20	0	0	0	22.5	0.0	0.0	0.0	0:12.77	io_wqe_worker-0
322862	root	20	0	0	0	22.8	0.0	0.0	0.0	0:14.36	io_wqe_worker-0
322836	root	20	0	0	0	22.2	0.0	0.0	0.0	0:12.96	io_wqe_worker-0
322772	root	20	0	458260	root	12706	22.5	0.0	0.0	0:12.45	waitd
322796	root	20	0	0	0	22.2	0.0	0.0	0.0	0:14.00	io_wqe_worker-0
322800	root	20	0	0	0	21.5	0.0	0.0	0.0	0:14.13	io_wqe_worker-0
322822	root	20	0	0	0	22.5	0.0	0.0	0.0	0:12.86	io_wqe_worker-0
322818	root	20	0	244764	root	6005	5	-2.2	0.0	0:12.75	io_wqe_worker-0
322833	root	20	0	0	0	0	5.2	0.0	0.0	1:21.29	lfflag
322854	root	20	0	0	0	0	5.2	0.0	0.0	0:02.28	io_wqe_worker-0
322854	root	20	0	0	0	0	5.2	0.0	0.0	0:02.50	io_wqe_worker-0
322842	root	20	0	0	0	0	5.6	0.0	0.0	0:02.78	io_wqe_worker-0
322851	root	20	0	0	0	0	5.6	0.0	0.0	0:02.49	io_wqe_worker-0
322860	root	20	0	0	0	0	5.6	0.0	0.0	0:02.54	io_wqe_worker-0
322862	root	20	0	0	0	0	5.6	0.0	0.0	0:02.79	io_wqe_worker-0
322870	root	20	0	1031718	root	54344	4.2	1.1	1.4	0:20.50	perf
322836	root	20	0	0	0	0	5.2	0.0	0.0	0:02.61	io_wqe_worker-0
322839	root	20	0	0	0	0	4.2	0.0	0.0	0:02.77	io_wqe_worker-0
322860	root	20	0	0	0	0	4.0	0.0	0.0	0:02.52	io_wqe_worker-0
322865	root	20	0	0	0	0	4.0	0.0	0.0	0:02.68	io_wqe_worker-0
322866	root	20	0	0	0	0	4.0	0.0	0.0	0:02.66	io_wqe_worker-0
322867	root	20	0	0	0	0	4.0	0.0	0.0	0:02.57	io_wqe_worker-0
322845	root	20	0	0	0	0	3.6	0.0	0.0	0:02.50	io_wqe_worker-0
322856	root	20	0	0	0	0	3.6	0.0	0.0	0:02.33	io_wqe_worker-0
322858	root	20	0	0	0	0	3.6	0.0	0.0	0:02.52	io_wqe_worker-0

```
Sample: 50 of event 'cycles', 1000 Hz, Event count (approx.): 5267050929 Inst: 0/0 drop: 0/0
Overhead: Shared object:
 0.4% [kernel] | [k] copy_user_enhanced_fast_string
 0.4% [kernel] | [k] native_queue_splice_lock_sleep
 1.0% [kernel] | [k] packet_rcv
 1.0% [kernel] | [k] do_tcp_sendpages
 1.0% [kernel] | [k] raw_spin_lock_hb
 1.0% [kernel] | [k] prh_fill_curr_blockIsra0
 1.0% [kernel] | [k] raw_spin_lock
 0.0% [kernel] | [k] copy_page_to_iter
 0.0% [kernel] | [k] sock_release_data
 0.0% [kernel] | [k] check_object_size
or a higher level overview, try: perf top --sort com.dsz
```

	1575379260	3151875040	472614016	638215680/2077603440
127.0.0.1		io_127.0.0.1		1816 1816 1816
		sw		0 0 0
tk:	com: 2264210	peak: 6.56k		rates: 1816 1816 1816
sk:	0 0 0	0 0 0		0 0 0 0
TOTAL:	2264210	6.56k		1816 1816 1816

More loopback testing on brand new hardware

- ▶ Recently I re-did the loopback read tests IORING_OP_SENDMSG/SPLICE (from /dev/shm/)
 - ▶ 1 connection, ~10-13 GBytes/s, smbd 7% cpu, with 4 iou-wrk threads at 7%-50% cpu.
 - ▶ 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 iou-wrk threads at 3%-35% cpu.
- ▶ I also implemented SMB2 writes with IORING_OP_RECVMSG/SPLICE (tested to /dev/null)
 - ▶ 1 connection, ~7-8 GBytes/s, smbd 5% cpu, with 3 io-wrk threads at 1%-20% cpu.
 - ▶ 4 connections, ~10 GBytes/s, smbd 15% cpu, with 12 io-wrk threads at 1%-20% cpu.
- ▶ I tested with a Linux Kernel 5.13
 - ▶ In both cases the bottleneck is clearly on the smbclient side
 - ▶ We could apply similar changes to smbclient and add true multichannel support
 - ▶ It seems that the filesystem->pipe->socket path is much better optimized

More loopback testing on brand new hardware

- ▶ Recently I re-did the loopback read tests IORING_OP_SENDFMSG/SPLICE (from /dev/shm/)
 - ▶ 1 connection, ~10-13 GBytes/s, smbd 7% cpu, with 4 iou-wrk threads at 7%-50% cpu.
 - ▶ 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 iou-wrk threads at 3%-35% cpu.
- ▶ I also implemented SMB2 writes with IORING_OP_RECVMSG/SPLICE (tested to /dev/null)
 - ▶ 1 connection, ~7-8 GBytes/s, smbd 5% cpu, with 3 io-wrk threads at 1%-20% cpu.
 - ▶ 4 connections, ~10 GBytes/s, smbd 15% cpu, with 12 io-wrk threads at 1%-20% cpu.
- ▶ I tested with a Linux Kernel 5.13
 - ▶ In both cases the bottleneck is clearly on the smbclient side
 - ▶ We could apply similar changes to smbclient and add true multichannel support
 - ▶ It seems that the filesystem->pipe->socket path is much better optimized

More loopback testing on brand new hardware

- ▶ Recently I re-did the loopback read tests IORING_OP_SENDFMSG/SPLICE (from /dev/shm/)
 - ▶ 1 connection, ~10-13 GBytes/s, smbd 7% cpu, with 4 iou-wrk threads at 7%-50% cpu.
 - ▶ 4 connections, 24-30 GBytes/s, smbd 18% cpu, with 16 iou-wrk threads at 3%-35% cpu.
- ▶ I also implemented SMB2 writes with IORING_OP_RECVMSG/SPLICE (tested to /dev/null)
 - ▶ 1 connection, ~7-8 GBytes/s, smbd 5% cpu, with 3 io-wrk threads at 1%-20% cpu.
 - ▶ 4 connections, ~10 GBytes/s, smbd 15% cpu, with 12 io-wrk threads at 1%-20% cpu.
- ▶ I tested with a Linux Kernel 5.13
 - ▶ In both cases the bottleneck is clearly on the smbclient side
 - ▶ We could apply similar changes to smbclient and add true multichannel support
 - ▶ It seems that the filesystem->pipe->socket path is much better optimized

The road to upstream (TEVENT_FD_ERROR)

- ▶ We need support for TEVENT_FD_ERROR in order to monitor errors
 - ▶ When using IORING_OP_SEND,RECVMSG we still want to notice errors
 - ▶ This is the main merge request:
 - ▶ https://gitlab.com/samba-team/samba/-/merge_requests/2793
 - ▶ This merge request converts Samba to use TEVENT_FD_ERROR:
 - ▶ https://gitlab.com/samba-team/samba/-/merge_requests/2885
 - ▶ (It also simplifies other places in the code without io_uring)

The road to upstream (samba_io_uring abstraction 1)

API glue to tevent:

```
void samba_io_uring_ev_register(void);

const struct samba_io_uring_features *samba_io_uring_system_features(void);

struct samba_io_uring *samba_io_uring_ev_context_get_ring(struct tevent_context *ev);

const struct samba_io_uring_features *samba_io_uring_get_features(
    const struct samba_io_uring *ring);

ev = tevent_context_init_byname(mem_ctx, "samba_io_uring_ev");
```

- ▶ samba_io_uring abstraction factored out of vfs_io_uring:
 - ▶ samba_io_uring_ev_hybrid tevent backend (glued on epoll backend)
 - ▶ It means every layer getting the tevent_context can use io_uring
 - ▶ No #ifdef's just checking if the required features are available

The road to upstream (samba_io_uring abstraction 1)

API glue to tevent:

```
void samba_io_uring_ev_register(void);

const struct samba_io_uring_features *samba_io_uring_system_features(void);

struct samba_io_uring *samba_io_uring_ev_context_get_ring(struct tevent_context *ev);

const struct samba_io_uring_features *samba_io_uring_get_features(
    const struct samba_io_uring *ring);

ev = tevent_context_init_byname(mem_ctx, "samba_io_uring_ev");
```

- ▶ samba_io_uring abstraction factored out of vfs_io_uring:
 - ▶ samba_io_uring_ev_hybrid tevent backend (glued on epoll backend)
 - ▶ It means every layer getting the tevent_context can use io_uring
 - ▶ No #ifdef's just checking if the required features are available

The road to upstream (samba_io_uring abstraction 2)

generic submission/completion api:

```
void samba_io_uring_completion_prepare(struct samba_io_uring_completion *completion,
    void (*completion_fn)(struct samba_io_uring_completion *completion,
        void *completion_private,
        const struct io_uring_cqe *cqe),
    void *completion_private);

void samba_io_uring_submission_prepare(struct samba_io_uring_submission *submission,
    void (*submission_fn)(struct samba_io_uring *ring,
        struct samba_io_uring_submission *submission,
        void *submission_private),
    void *submission_private,
    struct samba_io_uring_completion *completion);

struct io_uring_sqe *samba_io_uring_submission_sqe(struct samba_io_uring_submission *
    submission);

size_t samba_io_uring_queue_submissions(struct samba_io_uring *ring,
    struct samba_io_uring_submission *submission);
```

▶ Using it ...

- ▶ convert `vfs_io_uring`
- ▶ use it in `smb2_server.c`
- ▶ In future use it in other performance critical places too.

The road to upstream (samba_io_uring abstraction 2)

generic submission/completion api:

```
void samba_io_uring_completion_prepare(struct samba_io_uring_completion *completion,
    void (*completion_fn)(struct samba_io_uring_completion *completion,
        void *completion_private,
        const struct io_uring_cqe *cqe),
    void *completion_private);

void samba_io_uring_submission_prepare(struct samba_io_uring_submission *submission,
    void (*submission_fn)(struct samba_io_uring *ring,
        struct samba_io_uring_submission *submission,
        void *submission_private),
    void *submission_private,
    struct samba_io_uring_completion *completion);

struct io_uring_sqe *samba_io_uring_submission_sqe(struct samba_io_uring_submission *
    submission);

size_t samba_io_uring_queue_submissions(struct samba_io_uring *ring,
    struct samba_io_uring_submission *submission);
```

▶ Using it ...

- ▶ convert `vfs_io_uring`
- ▶ use it in `smb2_server.c`
- ▶ In future use it in other performance critical places too.

The road to upstream (smb2_server.c)

- ▶ Refactoring of smb2_server.c
 - ▶ add optional IORING_OP_SENDMSG, IORING_OP_RECVMSG support
- ▶ There are structural problems with splice from a file
 - ▶ I had a discussion with the Linux developers about it:
 - ▶ The page content from the page cache may change unexpectedly
 - ▶ <https://lists.samba.org/archive/samba-technical/2023-February/thread.html#137945>
 - ▶ We may not be able to use IORING_OP_SENDMSG/SPLICE by default
 - ▶ Maybe IORING_OP_RECVMSG/SPLICE is possible
- ▶ At least we can have only 1 one copy instead of two:
 - ▶ IORING_OP_SENDMSG_ZC is able to avoid copying to the socket
 - ▶ we get an extra completion once the buffers are not needed anymore
 - ▶ This gives good results, between with and without IORING_OP_SENDMSG/SPLICE
 - ▶ But I don't have numbers as it doesn't work on loopback
 - ▶ Within VM's improvement can be seen

The road to upstream (smb2_server.c)

- ▶ Refactoring of smb2_server.c
 - ▶ add optional IORING_OP_SENDMSG, IORING_OP_RECVMSG support
- ▶ There are structural problems with splice from a file
 - ▶ I had a discussion with the Linux developers about it:
 - ▶ The page content from the page cache may change unexpectedly
 - ▶ <https://lists.samba.org/archive/samba-technical/2023-February/thread.html#137945>
 - ▶ We may not be able to use IORING_OP_SENDMSG/SPLICE by default
 - ▶ Maybe IORING_OP_RECVMSG/SPLICE is possible
- ▶ At least we can have only 1 one copy instead of two:
 - ▶ IORING_OP_SENDMSG_ZC is able to avoid copying to the socket
 - ▶ we get an extra completion once the buffers are not needed anymore
 - ▶ This gives good results, between with and without IORING_OP_SENDMSG/SPLICE
 - ▶ But I don't have numbers as it doesn't work on loopback
 - ▶ Within VM's improvement can be seen

The road to upstream (smb2_server.c)

- ▶ Refactoring of smb2_server.c
 - ▶ add optional IORING_OP_SENDMSG, IORING_OP_RECVMSG support
- ▶ There are structural problems with splice from a file
 - ▶ I had a discussion with the Linux developers about it:
 - ▶ The page content from the page cache may change unexpectedly
 - ▶ <https://lists.samba.org/archive/samba-technical/2023-February/thread.html#137945>
 - ▶ We may not be able to use IORING_OP_SENDMSG/SPLICE by default
 - ▶ Maybe IORING_OP_RECVMSG/SPLICE is possible
- ▶ At least we can have only 1 one copy instead of two:
 - ▶ IORING_OP_SENDMSG_ZC is able to avoid copying to the socket
 - ▶ we get an extra completion once the buffers are not needed anymore
 - ▶ This gives good results, between with and without IORING_OP_SENDMSG/SPLICE
 - ▶ But I don't have numbers as it doesn't work on loopback
 - ▶ Within VM's improvement can be seen

Future Improvements

- ▶ I have a prototype for a native `io_uring` tevent backend:
 - ▶ The idea is to avoid `epoll` and only block in `io_uring_enter()`
 - ▶ But the semantics of `IORING_OP_POLL_ADD,REMOVE` are not useable
 - ▶ <https://lists.samba.org/archive/samba-technical/2022-October/thread.html#137734>
 - ▶ We may get an `IORING_POLL_CANCEL_ON_CLOSE` in future
 - ▶ And a usable `IORING_POLL_LEVEL`
- ▶ We can use `io_uring` deep inside of the `smbclient` code
 - ▶ The low layers can just use `samba_io_uring_ev_context_get_ring()`
 - ▶ And use it available without changing the whole stack

Future Improvements

- ▶ I have a prototype for a native `io_uring` tevent backend:
 - ▶ The idea is to avoid `epoll` and only block in `io_uring_enter()`
 - ▶ But the semantics of `IORING_OP_POLL_ADD,REMOVE` are not useable
 - ▶ <https://lists.samba.org/archive/samba-technical/2022-October/thread.html#137734>
 - ▶ We may get an `IORING_POLL_CANCEL_ON_CLOSE` in future
 - ▶ And a usable `IORING_POLL_LEVEL`
- ▶ We can use `io_uring` deep inside of the `smbclient` code
 - ▶ The low layers can just use `samba_io_uring_ev_context_get_ring()`
 - ▶ And use if available without changing the whole stack

Questions? Feedback!

- ▶ Stefan Metzmacher, metze@samba.org
- ▶ <https://www.sernet.com>
- ▶ <https://samba.plus>

Slides: <https://samba.org/~metze/presentations/2023/SambaXP/>