

Async VFS Future

within Samba

Stefan Metzmacher <metze@samba.org>

Samba Team / SerNet

2019-09-23

<https://samba.org/~metze/presentations/2019/SDC/>

Topics

- ▶ The Evolution of Async IO
- ▶ Async SMB2 Query Directory
- ▶ Current Impersonation Model
- ▶ Fail with tevent_wrapper
- ▶ Modern VFS for SMB2/3
- ▶ Future Impersonation Model
- ▶ Make every VFS operation async
- ▶ Questions?

- ▶ Samba 2.2.0 was the first release that added a vfs abstraction
 - ▶ It supported 34 operations, basically posix like syscalls
 - ▶ `opendir()`, `open()`, `close()`, ...
 - ▶ And a few NT like calls like `[f]{get,set}_nt_acl()`
 - ▶ It only supported one module per share.
- ▶ Samba 3.0.0 made the SMB_VFS layer more flexible
 - ▶ Each share can have a chain of vfs modules specified
 - ▶ Modules like 'audit' or 'recycle' are stacked on top of the default
 - ▶ `shadow_copy` and quota operations were added

The Evolution of Async IO (1)

- ▶ Samba 3.0.20 added support for the posix aio api
 - ▶ `aio_read()`, `aio_write`, `aio_suspend()`, `aio_return()`
 - ▶ It uses realtime signals for completions
 - ▶ The glibc wrappers use a mutex per file descriptor
 - ▶ Only one pending io is possible per file descriptor
- ▶ Samba 3.3.0 added the `aio_fork` module
 - ▶ It uses shared memory
 - ▶ A unix socketpair/fd-passing is used for requests and completions
 - ▶ Supports multiple pending io requests per file descriptor
 - ▶ While having a bit more overhead
- ▶ Samba 3.6.6 added the `aio_pthread` module
 - ▶ It uses a generic pthreadpool layer
 - ▶ Only single (blocking) syscalls in the helper threads
 - ▶ Uses a pipe for the completions

The Evolution of Async IO (2)

- ▶ Samba 4.0.0 changed the way from emulating posix aio
 - ▶ It uses tevent_req based _send/_recv function pairs
 - ▶ We aim to have just one async programming model
 - ▶ struct tevent_req *SMB_VFS_PREAD_SEND(mem_ctx, ev, ...)
 - ▶ tevent_req_set_callback(req, smb_layer_pread_done, smbreq);
 - ▶ int SMB_VFS_PREAD_RECV(struct tevent_req *req, ...)
- ▶ Samba 4.0.0 aio_thread moved into the default backend
 - ▶ pthreadpool based async pread, pwrite and fsync are the default now
 - ▶ It uses the generic pthreadpool_tevent_job_send/recv()
 - ▶ No vfs module needs to be configured
- ▶ Samba still has an aio_thread module
 - ▶ But it only implements async open(O_CREAT|O_EXCL) on Linux
 - ▶ As it is a path based operation we need to get the impersonation right
 - ▶ Only Linux supports per thread credentials
 - ▶ But you need to bypath glibc as it implicitly keeps all threads in sync

The Evolution of Async IO (3)

- ▶ Samba 4.0.0 added aio_linux
 - ▶ Based on io_prep_pread(), io_submit() and io_getevents()
 - ▶ Uses eventfd() for the completions
 - ▶ It doesn't do real async io because Samba doesn't use O_DIRECT
 - ▶ See [bug #13128](#) for more details
 - ▶ It was removed again in Samba 4.9.0
- ▶ Samba 4.12.0 will most likely get an io_uring module
 - ▶ Linux 5.1 introduced a new ring buffer based io_uring interface
 - ▶ It avoids syscalls/context switches as much as possible
 - ▶ It supports async io even for buffered io
 - ▶ See <https://lwn.net/Articles/778411/>
 - ▶ A patch is available at https://gitlab.com/samba-team/samba/merge_requests/529
 - ▶ It compiles fine, but wasn't tested yet at runtime

- ▶ SMB2 Query Directory needs to return meta data for each entry
 - ▶ On a unix system `readdir()` only returns types and names
 - ▶ For SMB we also fetch meta data for each name
 - ▶ We need the write time from `locking.tdb`
 - ▶ We need DOSATTRs typically from `xattrs`
 - ▶ We need the result from `stat()`
- ▶ Samba 4.7.0 optimized fetching the write time.
 - ▶ In a cluster it is expensive to migrate `locking.tdb` records via `ctdb`
 - ▶ We invented `dbwrap_parse_record_send/recv()`
 - ▶ This makes it possible to batch requests to `ctdb` and reduce latency
 - ▶ Which is crucial when serving directories with a lot of entries

- ▶ Samba 4.10 optimizes fetching the DOSATTRs/xattrs
 - ▶ On some filesystem `getxattr()` is much more expensive than `stat()`
 - ▶ We added `SMB_VFS_GETXATTRAT_SEND/RECV()`
 - ▶ And `SMB_VFS_GET_DOS_ATTRIBUTES_SEND/RECV()` on top
 - ▶ This lowers the overall latency a lot for such filesystems
 - ▶ It's off by default in order to avoid overhead for fast filesystems
- ▶ `SMB_VFS_GETXATTRAT_SEND/RECV()` without `getxattrat()`
 - ▶ There's no `getxattrat()` syscall yet
 - ▶ We simulate it with `fchdir()` and `getxattr()` with a relative path
 - ▶ With our `pthreadpool` each thread needs its current working directory
 - ▶ On Linux we can use `unshare(CLONE_FS)`
 - ▶ Some container solutions reject `unshare()` without looking at the flags
- ▶ `SMB_VFS_GET_FILE_INFO_SEND/RECV()` might be the future goal
 - ▶ This would get a mask to request individual aspects
 - ▶ This would abstract `statx()` and `getxattr()` into one helper thread
 - ▶ And also include the write time from `locking.tdb` if requested

- ▶ The SMB layer calls `change_to_user_and_service()`
 - ▶ This applies to the main process thread for the connection
 - ▶ It is called before dispatching each request
 - ▶ This changes the `uid`, `gid` and groups (in a cached way)
 - ▶ It changes to the share root directory
 - ▶ Sets the global state for parameter substitutions like `%U`, `%L`, ...
- ▶ The SMB_VFS layer relies on already performed impersonation
 - ▶ It doesn't actively need to take care of it
 - ▶ Special cases use `become_root()/unbecome_root()`
 - ▶ `change_to_user_and_service_by_fsp()` for `OFFLOAD_WRITE()`
 - ▶ `SMB_VFS_OFFLOAD_WRITE()` operates on two `fsps`
 - ▶ These may not use the same share
- ▶ Path based operations are typically replayed completely
 - ▶ Async opens, e.g, waiting for `oplock` breaks, are replayed at the SMB layer
 - ▶ We reparse the request buffer and redo the impersonation
 - ▶ There's no natural way to keep state for the overall request

Fail with `tevent_wrapper` (1)

- ▶ The natural way to keep state for async operations
 - ▶ We have the wellknown `tevent_req` based `_send/_recv` model
 - ▶ The impersonation may change during the async processing
 - ▶ VFS modules could no longer rely on being correctly impersonated
 - ▶ And doing that by hand is waiting for security problems to happen
- ▶ I developed a `tevent_wrapper` infrastructure
 - ▶ The design was to allow hooks before and after each `tevent` handler
 - ▶ The SMB server used that in order to do impersonation
 - ▶ It only passed down a per user `tevent_context` wrappers
 - ▶ This way the SMB_VFS modules were always in the correct state
 - ▶ And it was not possible to forget the impersonation

- ▶ I developed a pthreadpool wrapper infrastructure
 - ▶ The design was to allow hooks before and after each syscall
 - ▶ The SMB server used that in order to do impersonation
 - ▶ It passed down per user pthreadpool context wrappers
 - ▶ This way the SMB_VFS modules could easily use the pthreadpool
 - ▶ And it was not possible to forget the impersonation
- ▶ The implementation was very complex
 - ▶ It was not really obvious when impersonation happens
 - ▶ Especially when simulating become_root()
 - ▶ The pthreadpool code was using lockless thread interaction
 - ▶ It was in master for a while, but got reverted before 4.10.0rc1
 - ▶ Instead we added explicit impersonation in the few required places

Modern VFS for SMB2/3

- ▶ Deprecation of SMB1 in 4.11
 - ▶ The world is clearly moving away from SMB1
 - ▶ So we are, SMB1 is now disabled by default
 - ▶ But it is not yet possible to remove it completely
- ▶ SMB2/3 is a handle based protocol
 - ▶ Create takes a full pathname (relative to the share root)
 - ▶ Everything else operates on a handle returned by Create
 - ▶ QueryInfo(NormalizedNameInformation) returns a full pathname
 - ▶ QueryDirectory returns relative pathnames
 - ▶ SetInfo(File{Link,Rename}Information) takes a full target pathname
- ▶ The SMB_VFS layer can be simplified a lot
 - ▶ Modern operating systems have handle based syscalls
 - ▶ We added SMB_VFS.RENAMEAT(), SMB_VFS.LINKAT(),
 - ▶ SMB_VFS.MKNODAT(), SMB_VFS.READLINKAT(),
 - ▶ SMB_VFS.SYMLINKAT(), SMB_VFS.MKDIRAT()
 - ▶ More calls will follow
 - ▶ That should allow us to remove a lot of legacy code

- ▶ Some SMB_VFS backends don't use posix syscalls
 - ▶ glusterfs and ceph use userspace libraries instead of syscalls
 - ▶ These would also work custom impersonation
 - ▶ File descriptor based syscalls also do not need impersonation
- ▶ We will move the impersonation from top to bottom
 - ▶ We no longer do generic impersonation at the SMB layer
 - ▶ Each SMB_VFS module needs to do impersonation where required
 - ▶ We provide simple and easy to understand helper functions
 - ▶ Every SMB_VFS call gets an explicit impersonation token passed
 - ▶ This makes it obvious for module writers that our strategy has changed

- ▶ Introducing lpcfg_substitution to avoid global state
 - ▶ It is complex to keep the global state for substitutions like %U, %L
 - ▶ We have 54 global and 27 per share options with substitution support
 - ▶ We can remove the substitution support for some of them
 - ▶ The rest will be converted to require an explicit lpcfg_substitution

Creation functions for the new impersonation model (A unique 64-bit cache-id is assigned):

```
NTSTATUS smb_vfs_impersonation_create(TALLOC_CTX *mem_ctx,
                                     const struct auth_session_info *session_info,
                                     const struct lpcfg_substitution *substitution,
                                     struct smb_vfs_impersonation **imp);
struct smb_vfs_impersonation *smb_vfs_impersonation_ref(TALLOC_CTX *mem_ctx,
                                                         const struct smb_vfs_impersonation *imp);
const struct auth_session_info *smb_vfs_impersonation_session_info(
    const struct smb_vfs_impersonation *imp);
const struct lpcfg_substitution *smb_vfs_impersonation_substitution(
    const struct smb_vfs_impersonation *imp);
```

Impersonation helper functions for the new impersonation model (they use the cache-id to avoid overhead):

```
NTSTATUS smb_vfs_impersonate_unix_token(const struct smb_vfs_impersonation *imp);
void smb_vfs_impersonation_cache_reset(void);
```

- ▶ Introducing simple syscall wrapper and blacklist defines
 - ▶ It would still be complex if modules have to impersonate explicitly
 - ▶ `smb_vfs_impersonate_unix_token()` will typically be hidden
 - ▶ SMB_VFS modules won't ever call syscalls directly

The syscall wrappers and defines to detect direct syscalls:

```
#define __SMB_VFS_IMPERSONATE_UNIX_TOKEN_CHECK_ERRNO(__imp, __ret_errno) do { \
    NTSTATUS status; \
    status = smb_vfs_impersonate_unix_token(__imp); \
    if (!NT_STATUS_IS_OK(status)) { \
        errno = __ret_errno; \
        return -1; \
    } \
} while(0)

static inline int smb_vfs_sys_renameat(const struct smb_vfs_impersonation *imp,
                                       int olddirfd, const char *oldpath,
                                       int newdirfd, const char *newpath)
{
    __SMB_VFS_IMPERSONATE_UNIX_TOKEN_CHECK_ERRNO(imp, EPERM);
    return renameat(olddirfd, oldpath, newdirfd, newpath);
}

#define renameat __error_please_use_smb_vfs_sys_renameat
```

Make every VFS operation async (1)

- ▶ We would like to have all operations async
 - ▶ We have OEMs who use Samba as a gateway to cloud storage
 - ▶ Others may also need HSM were tapes or slow disks are used
- ▶ Modern storage is very fast
 - ▶ NVMe SSDs and Persistent Memory requires minimal overhead
 - ▶ Maintaining tevent_req states at multiple levels adds overhead
 - ▶ Going async is not needed and a waste of resources

3 calls per operation, STATUS_DRIVER_BLOCKED (or EWOULDBLOCK) triggers the async path:

(Modules can implement `sync_fn` and/or `send/recv_fn`)

```
int (*mkdirat_sync_fn)(struct vfs_handle_struct *handle,
                      const struct smb_vfs_impersonation *imp,
                      struct files_struct *dirfsp,
                      const struct smb_filename *smb_fname,
                      mode_t mode);

struct tevent_req *(*mkdirat_send_fn)(TALLOC_CTX *mem_ctx,
                                     struct vfs_handle_struct *handle,
                                     const struct smb_vfs_impersonation *imp,
                                     struct files_struct *dirfsp,
                                     const struct smb_filename *smb_fname,
                                     mode_t mode);

int (*mkdirat_recv_fn)(struct tevent_req *req, struct vfs_aio_state *state);
```


Make every VFS operation async (2)

- ▶ Things get more complicated with database locks
 - ▶ For various operations we need to have our open file database locked
 - ▶ This prevents races, e.g. in case multiple low level operations are needed
- ▶ Updating the byte range lock database is such an operation
 - ▶ Samba 4.11 brings the possibility to implement async backends
 - ▶ For now we use a different model without tevent_req
 - ▶ `SMB_VFS_BRL_LOCK_WINDOWS()` can return `NT_STATUS_RETRY`

The `SMB_VFS` call is unchanged, but we now have helper functions to identify the request and remember state for it:

```
NTSTATUS (*brl_lock_windows_fn)(struct vfs_handle_struct *handle,
                              struct byte_range_lock *br_lck,
                              struct lock_struct *plock);
TALLOC_CTX *brl_req_mem_ctx(const struct byte_range_lock *brl);
const struct GUID *brl_req_guid(const struct byte_range_lock *brl);
```

Questions?

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

→ SerNet/SAMBA+ sponsor booth