

Code Notes:

Seafile Components: <http://manual.seafile.com/develop/server-components.html>

Seafile Data Model: http://manual.seafile.com/develop/data_model.html

Seafile Sync Algorithm: http://manual.seafile.com/develop/sync_algorithm.html

COMPONENTS OF SEAFILE:

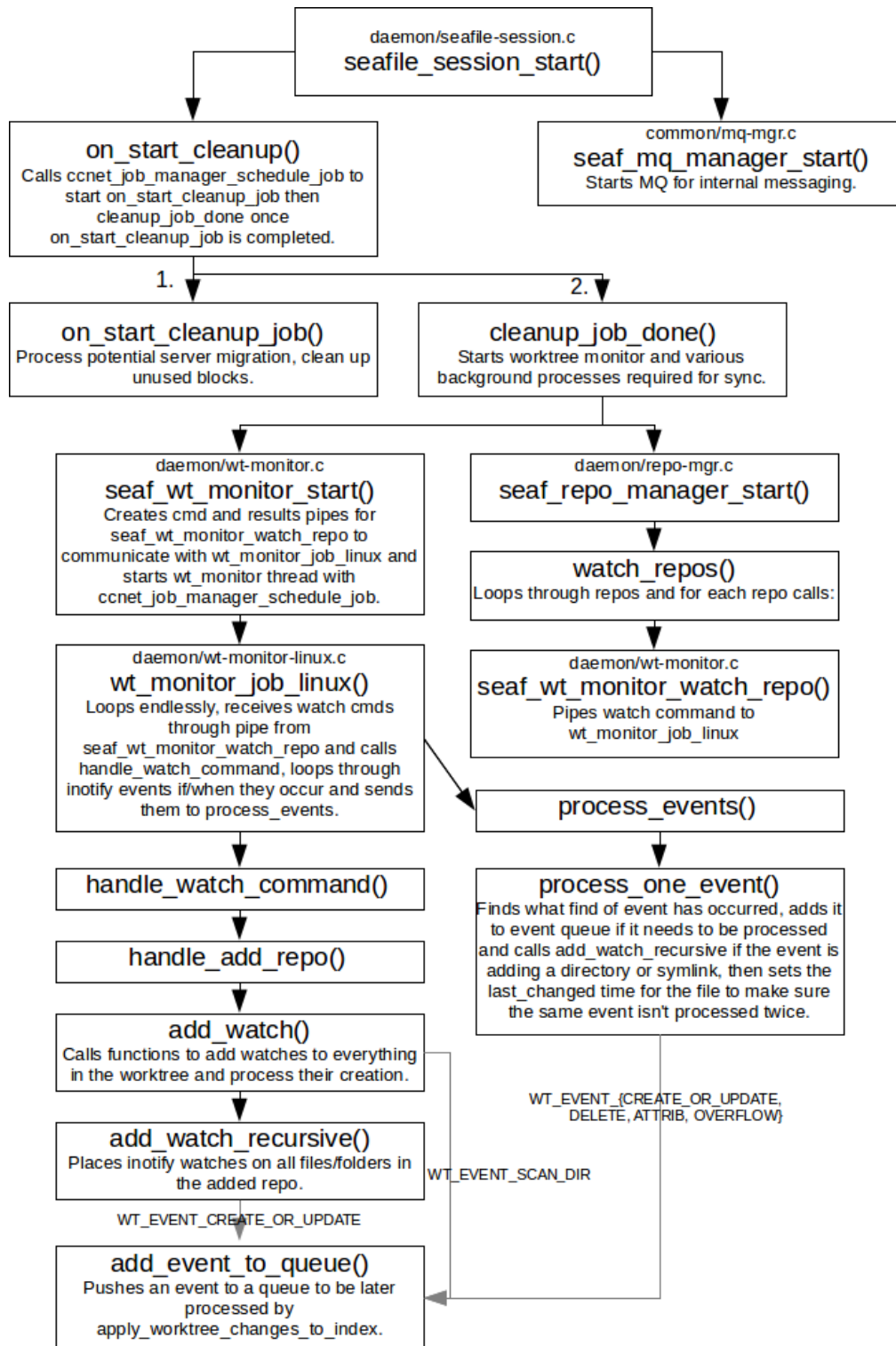
- Seafile Daemon is the daemon that runs on the client machine and processes added/removed/modified files and sends/receives the main file data (but not commit data and file metadata).
- Ccnet runs on both the client and server and handles internal messaging for the processes and threads, and transfers the commit data and file metadata from client to server and vice versa.
- Seaf-cli is the command line interface used on the client, it issues commands to Seafile Daemon and Ccnet.
- Seafile Server runs on the server, it is the application which does most of the tasks on the server.
- Seahub runs on the server and is the web frontend for seafile, it is needed to access it both from the website it generates and through the command line on the client.

We use Nginx and SQLite as a lightweight server and database respectively.

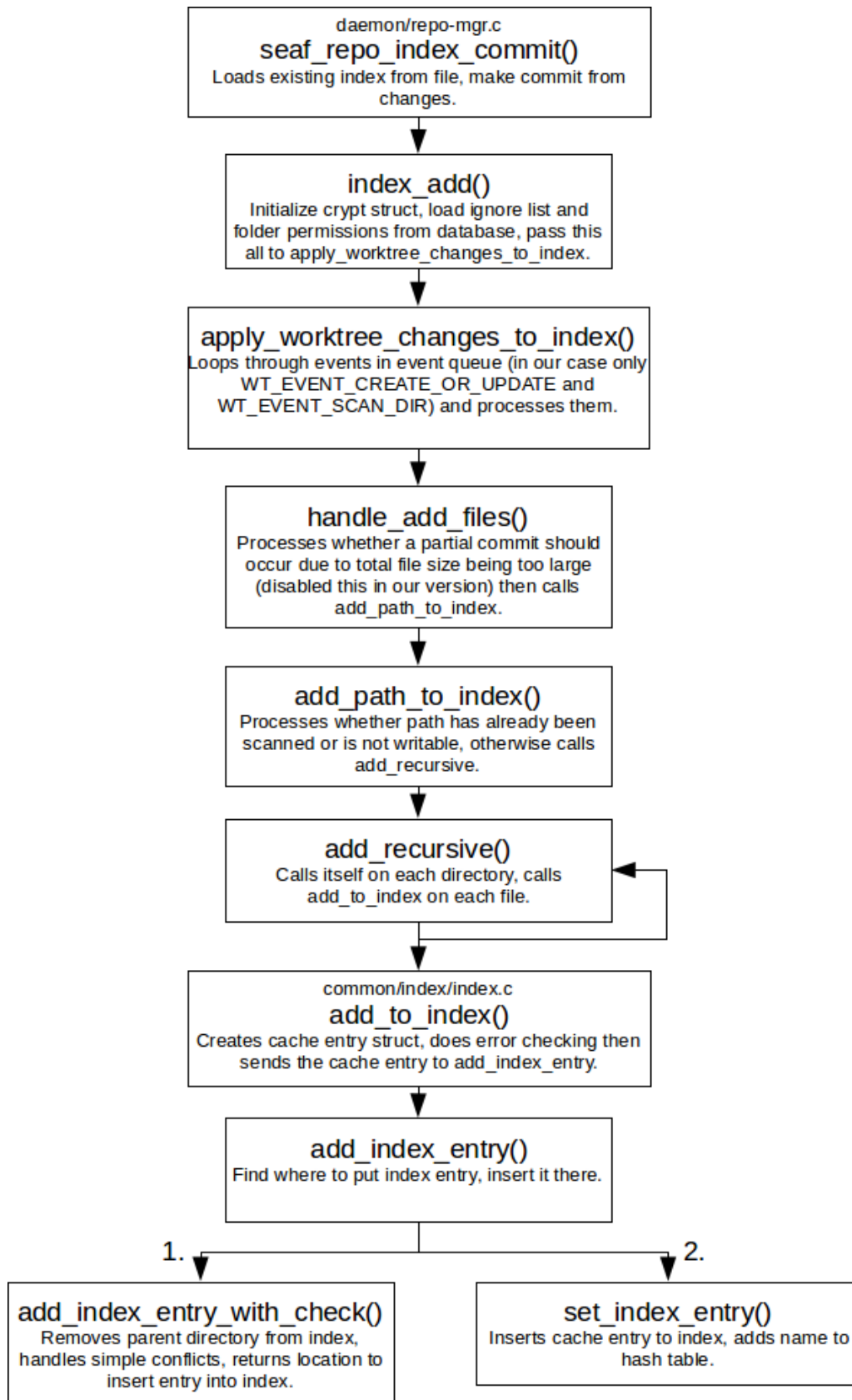
OVERVIEW OF PROGRAM FLOW FOR FILE SYNC (timed part of our sync tests):

1. Startup process & placing inotify watches

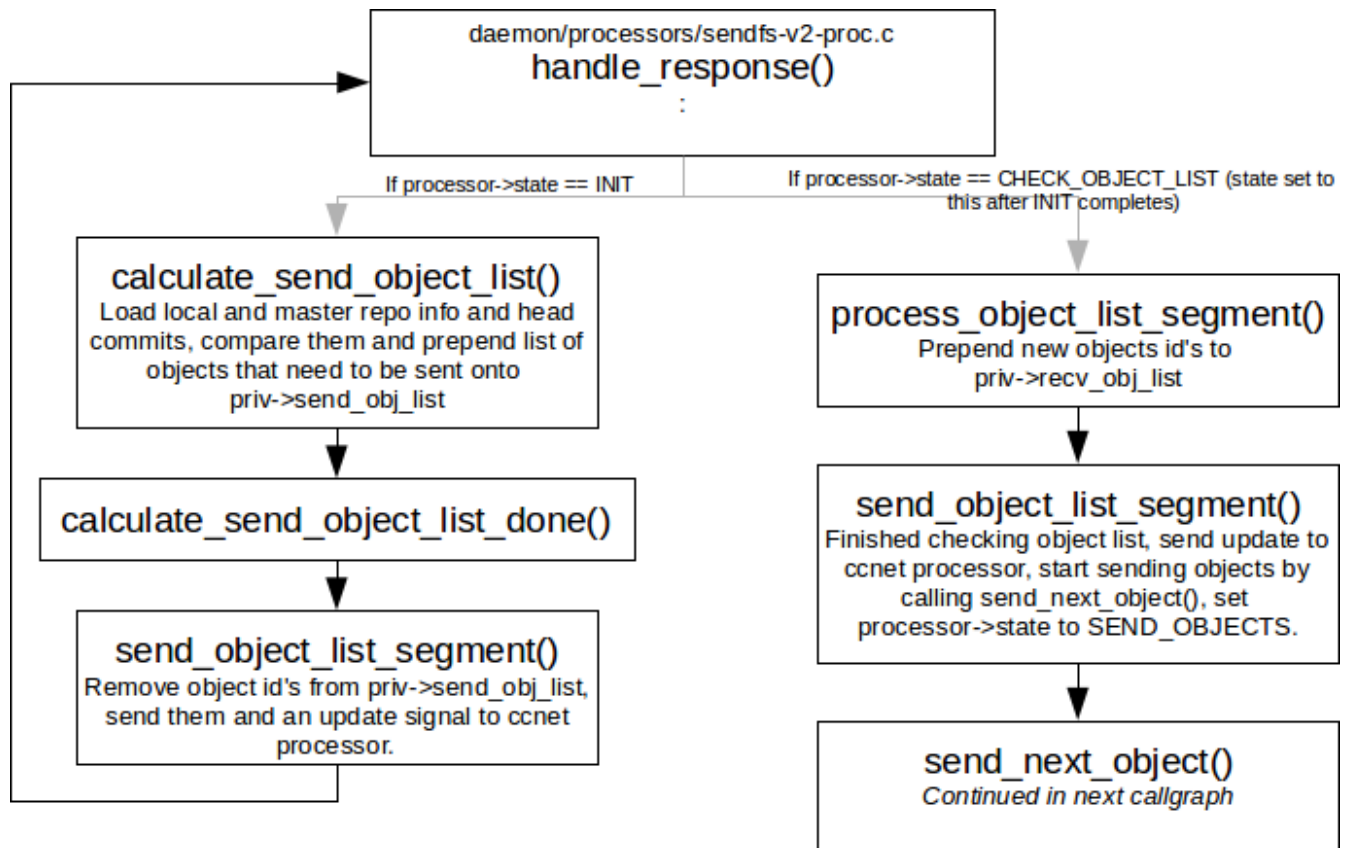
Section relevant to us:



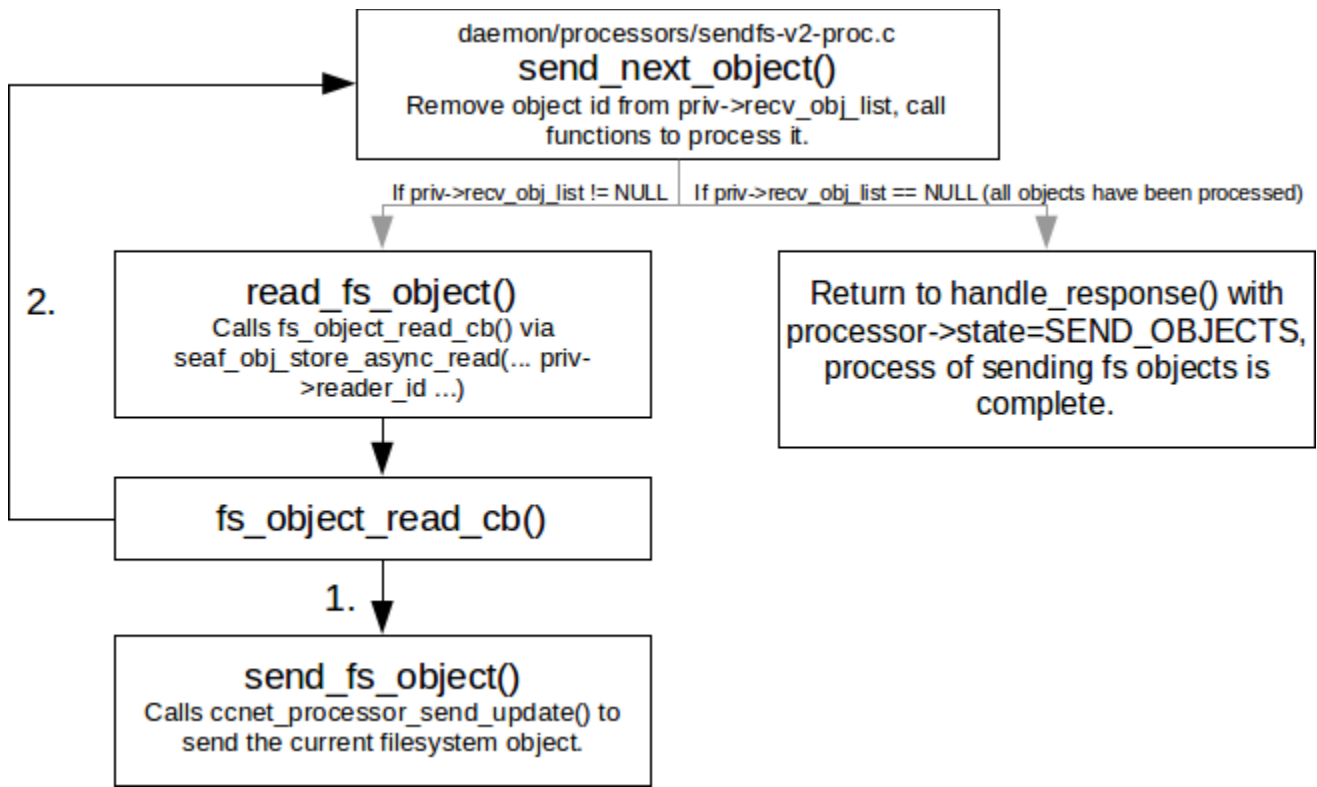
2. Update index with changes:



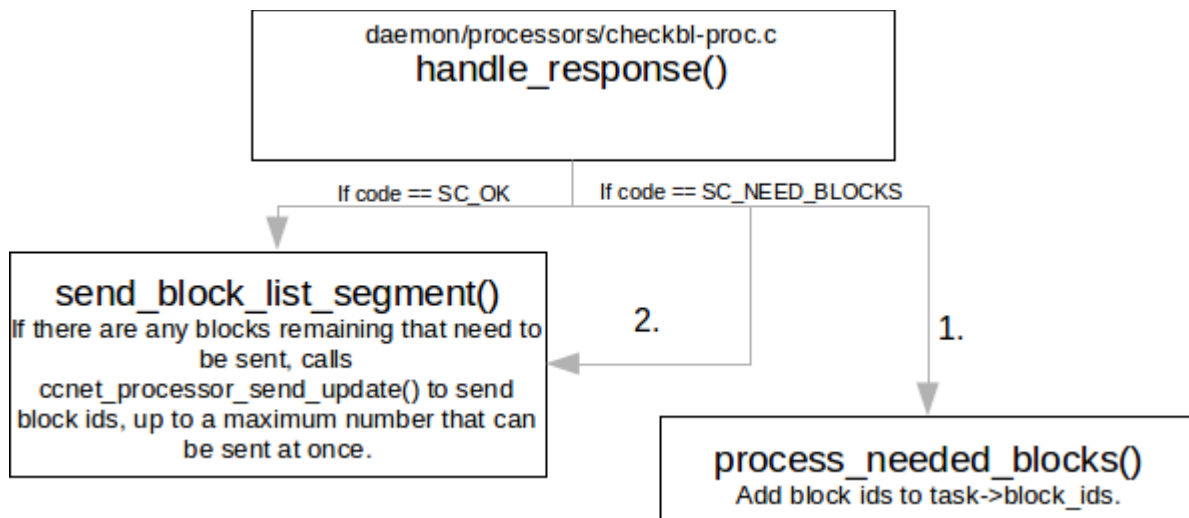
3. Send commit to server.
4. Calculate objects needed by client from server. (none in our use case as the repository on the server is empty)
5. Calculate/send objects needed by server from client. (object id's of the new files we added).



6. Calculate/send fs objects needed by server from client. (file system objects (metadata) of the new files we added).



7. Calculate/send block ids needed by server from client. (ids of data blocks of the new files we added).



8. Calculate/send blocks needed by server from client. (data blocks of the new files we added).

The main loop for this is in `client_thread_loop()` in `daemon/block-tx-client.c`

9. Cleanup.

In order to get gprof output for the Seafile client, we have to compile both seafile and ccnet with the `-pg` flag. This is already done in our build scripts.

Seafile responds to the following inotify events:

- `IN_IGNORED`
- `IN_UNMOUNT`
- `IN_Q_OVERFLOW`
- `IN_MODIFY`
- `IN_CREATE`
- `IN_DELETE`
- `IN_MOVED_FROM`
- `IN_MOVED_TO`
- `IN_CLOSE_WRITE`
- `IN_ATTRIB`

SLEEPS:

The times where `g_usleep` occurred in the program have been replaced by `G_USLEEP`, which is defined as `nothing (do {} while (0);)`. This is to prevent the sync from pausing when the upload/download limit has been passed, since with only 1 user in our tests we don't need it. `G_USLEEP` is defined in `seafile/common/common.h`, and `g_usleep` has been replaced by `G_USLEEP` in the following files:

- `seafile/common/seaf-db.c`
- `seafile/common/processors/blocktx-common-impl-v2.h`
- `seafile/daemon/http-tx-mgr.c`
- `seafile/daemon/block-tx-client.c`

The times where `usleep` occurred in the program have been replaced by `USLEEP`, which is defined as `usleep`, i.e. no change (legacy from replacing sleeps of all kinds). These are used to sleep before retrying if unable to access database, which is usually caused by multiple users trying to access it at once. `usleep` has been replaced with `USLEEP` in the following files:

- `seafile/server/http-server.c`
- `seafile/server/repo-op.c`
- `seafile/server/processors/recvbranch-proc-v2.c`