Функция main().

Обзор:

2234-2237. Объявляются переменные:

Указатель на структуру ctx - контескт. Содержит все. Пока пустая. Лежит в glusterfs.h.

Int ret - проверка, нужно ли выходить из программы. Существует, как элемент ctx.

cmdlinestr - командная строка.

\*cmd - указатель на структуру \_cmd\_args. Лежит в glusterfs.h.

Cкорее всего, там содержатся команды и опции (внутри комментарии). Еще есть список структур опций транслятора.

2239. gf\_check\_and\_set\_mem\_acct.

Лежит в glusterfsd.c Проверяет, есть ли среди параметров командной строки "--no-mem-accounting". В зависимости от этого включает или не включает глобальный "счетчик" памяти (gf\_global\_mem\_acct).

Далее "функция-конструктор" для ctx.

Лежит в lib/ctx.c

ret в 0, создает еще 1 указатель на ctx, выделяет память под ctx. Если "не ctx", ret в -1 и выходим.

Устанавливает счетчик памяти, такой же как глобальный. Инициализирует заголовки списков graphs и mempool\_list структуры ctx. Устанавливаем daemon\_heads в -1.

Дальше проверка на взаимное исключения путей, если срабатывает - освобождаем память ctx. Выходим.

Если ошибка создания структуры, выдает сообщение "glusterfs context error" - gf\_msg. В этой функции проверяются дополнительно где именно ошибка, и выводится, там же пишутся логи, которые мы передаем. Вылетаем с ошибкой памяти.

Глобальная ctx указывает туда же, куда и ctx в main.

globals init - инициализирует глобальные переменные. Внутри gf\_log\_globals\_init - устанавливает значения переменных в ctx, связанные с логами, задает заголовок списка lru\_queue структуры log структуры ctx. Совершается проверка, нужно ли выходить.

(pthread\_once) - в аргумент выполняем функцию, которая ложит программу с ошибкой, если что-то не так. Если ошибка, кладем ее в логи, выходим из функции, в мэине определяется,что произошла ошибка, программа падает.

ret = glusterfs\_ctx\_defaults\_init (ctx);

cmd\_args\_t \*cmd\_args NULL <-- Объявление командной строки (command line)

ret = xlator\_mem\_acct\_init (THIS, gfd\_mt\_end) <-- функция для рета. THIS часто присваивает значения trav во всём коде этого модуля.

Также от THIS->ctx . Существует в 1262 строке while (trav) { ... } Возможно это нечто булевское.

\*\*\* xlator.h и .c - это файлы, в которых находится большое количество трансляторов. \*\*\*

Брать или не брать мьютекс здесь и в другом обработчике - gf\_print\_trace () - это большой вопрос. Принимая мьютекс в обработчик сигнала, это будет означать, что, если в процессе получается фатальный сигнал, а другой поток держит ctx-> log.log\_buf\_lock, чтобы, возможно, авторизировать сообщение в \_gf\_msg\_internal (), нарушившая нить висит на замке мьютекса навсегда, не давая выход процесса. С другой стороны, не принимая мьютекс в обработчик сигнала вызывает его изменения.

lru\_list буферированных сообщения Войти колоритный коррумпированной списка и потенциально привести к бесконечному каскаду SIGSEGVs и других вопросов

повторно entrancy.

Очевидно, что это командные функции, для авторизации сообщения через консоль и при любом не правильном вводе - ошибка и критический вылет, но так как программа работает постоянно с командной строкой, то вылет хоть и происходит, но потом снова при перезапуске будут все настройки со всеми изменениями пользователя. Однако мьютекс упрощает жизнь просто останавливая процесс и не создавая вылета. Ошибка есть, но блокируется системой и остаётся незамеченной, кроме как сообщения в консоли, что что-то не так.

Продолжение: ret делается -1 и это нужно для того, чтобы если не произойдёт ни одна из всех следующих событий в функции, то будет ошибка. Это всё идёт через ctx-> и приписке ивента или эррора. Причём может быть только 1 событие после чего очередной goto out.

ret = parse\_cmdline (argc, argv, ctx);

Если эта опция установлена, мы хотим распечатать и проверить файл, установить возвращаемое значение (код выхода в данном случае) и выйти.

В этой функции очень много других функций проверок, например на доступ и если всё делается успешно - тогда , вроде, goto out с ретом = 1 .

Если не так, выход из функции и вывод о наличии ошибки.

2270. Послле того как пропарсили аргументы командной строки, проверяем равен ли 1 cmd\_exports, если равен, нужно cделать принтекспортс и выйти ( принтекспортс прокомментирован).

Дальше начало работы с логами. Если лог фаил не задан, то он задается ф-ей gf\_set\_log\_file\_path.

gf\_set\_log\_file\_path: Внутри если аргументов не было, сразу выходим из функции, дальше, если есть точка монтирования (mount\_point), то выделяем

путь к файлу из этого аргумента. дальше что-либо записывается в логфаил функцией gf\_asprintf и внутреними. Тоже самое с переменными volfile и volfile\_Server.

Установка пути может вернуть ошибку, тогда выводим ее. Дальше утсновка финальных параметров логов разными функциями. Все эти функции - обыкновенные "поля",

с некоторыми проверками. После того, как все установлено, на сцене функция gf\_log\_init (видимо, уже включение логов, т.е создание логфайла.).

gf\_log\_init : Внутри всяческие проверки, которые выходят из функции, если что не так. Если все проверки пройдены и мы до сих пор внутри, создаем файл, проверим, создался ли он, если не создался, выдаем ошибку, выходим. Потом закрываем файл . Присваиваем gf\_log\_logfilev значение logfile структуры log, выходим из функции.

Возвращаемся в мэин.

Дальше идут просто {}. В них мы просто берем и аргументы командной строки записываем в ctx->cmdlinestr.

gf\_proc\_dump\_init() (запуск дампа процессов) – внутри

сreate\_fuse\_mount(ctx) - тут начитается работа с системой(файловой) fuse. А теперь идет проверка, указан ли mount\_point, если не указан, то выходим.

Дальше еще на какую-то ошибку проверка. Теперь выделяем память под транслятор master. Если не выделилась, вылетаем на метку ошибки. Присваеваем master->name - fuse функцией gf\_strdup(выделяет память под новый массив чаров, копирует в него аргумент, возвращает новый массив).

Если не получилось - на ошибку.

Устанавливаем тип транслятора set\_type. Внутри функция set\_type\_virtual : там проверки, связанные с транслятором и типом. устанавливаем тип (если

установился хорошо, возвращаем что все ок, если нет, то -1).

Если что-то пошло не так, вызывается ф-я xlator\_dynload , выходим из ф-и.

Если не установился, выдаем ошибку, идем на error.

У транслятора есть своя структура ctx, присваиваем ей просто ctx. Инициализируем настройки транслятору функцией get\_new\_dict(внутри хз что). В общем дальше там установка настроек fuse и прочих, потом xlatur\_init, в котором мы выделяем память под транслятор и пытаемся создать volume, если не получается - выдаем ошибку.

2305. Функция daemonize(). В ней идёт функция подгрузки файлов(pid\_file через командную строку). Снова создаётся cmd\_args и проверяется на наличие и отсутствие в командной строке каких-то записей и если есть то по метке снова к файлу подкачки и инжектирование таймера событий в log(?), а также установка СИГНАЛОВ (это еще одна из библиотек // это общая установка для всех потоков и блокировка эти сигналов без sigwaiter нитей, а также возврат к сигналам получаемых обрабаткой другими потоками, и установка обработчиков сигналов).

Затем ret = pipe (ctx->daemon\_pipe), где сказано, что если в итоге ret не получит deamon\_pipe, то "родитель” не будет получать ждать для получения статуса “ребёнка" pipe() был создан в xlator.

Также после этой функции идёт кейс, где ret либо -1, либо 0. Причём если -1, то происходит закрытие daemon\_pipe[0], daemon\_pipe[1] и вывод об ошибке.

При 0 включается режим "ребёнка" и закрывается только daemon\_pipe[0] и происходит "режим записи" (вполне возможно с консоли), а если рет не равен 0 и 1, то включается режим "родителя", где уже происходит только "чтение" и родитель ждёт статуса ребёнка, после чего выводит сообщение либо всё ок, либо ошибка.

2309. ctx->env = syncenv\_new

(0, 0, 0) Контекс-> Событие = функция(размер стека = 0, procmin = 0, procmax = 0) procmin и procmax я не нашёл, что означает, но эти переменные в

syncenv\_new() подвергаются проверкам. Глянув дальше, это переведётся как Process Min\Max.

2316. Стоит приписка, что делать эту операцию строго после deamonize(). Происходит проверка cmd->global\_timer\_wheel -это тот таймер, который, если вдруг истёк, то проделывает операцию.

ret = glusterfs\_global\_timer\_wheel\_init(ctx) ). Тут уже происходит подключение к следующей библиотеке tw.c\.h , причём это единственная функция в ней, в которой по видимому при истечении времени и не получении статус даётся ошибка, а при успехе дальше продолжаем работу в main-е. Функция тесно связана с библиотекой xlator и командой global\_timer\_wheel

2322. glusterfs\_volumes\_init() <-- функция, где происходит полное подключение к серверу от клиента. Подключаются проверки на наличие сокетов, volume файлов. Родитель (сервак) принимает пакет с размером ret (sizeof (ret)) в функции emancipate (освобождение) и ребёнок (клиент) "break free from the parent" (освобождается от родителя)

2326. ret = event\_dispatch (ctx->event\_pool); и попадаем в библиотеку event.c\.h , где имеется последняя маленькая функция об успехе видимо. Интересный факт: тут почти во всех функциях ивента встречается одна и тажа функция GF\_VALIDATE\_OR\_GOTO ("event", event\_pool, out); Разработчики могли бы её сократить и сэкономить много строк кода. Сделать более структурировано и было б понятнее и экономнее, что даёт оптимизацию и читаемость.

-----------------------