Memory Allocator

Most of the memory used by FoundationDB (client or server) is allocated via a customized "fast allocator" (flow/FastAlloc.h), which allocates memory from the OS in large blocks and then keeps it for reuse within the process. If a transaction allocates some memory, then after the transaction commits, the memory is returned back to the "fast allocator" for next reuse. This means that from the OS perspective an FoundationDB process's memory footprint will generally not go down.

Specifically, the memory been allocated are in blocks of 16, 32, 64, 96, 128, 256, 512, 1024, 2048, 4096, and 8192 bytes.

Transaction Memory Usage

Transactions have a per-instance memory pool called an "arena" (flow/Arena.h) in which all memory needed for the transaction's operations are allocated. This includes temporary copies of user keys, which is what is happening in `fdb_transaction_clear()`. This arena is freed all at once when the transaction is committed (whether successful or not), reset, or destroyed. The arena's memory comes from the fast allocator, so once freed it remains in the fast allocator to be reused within the process instead of being given back to the OS.

While it would be possible to optimize `fdb_transaction_clear()` to require less temporary memory, write transactions in FoundationDB are limited to 10MB of work and 5 seconds of lifetime so it is not expected that a sane write transaction workload would cause a significant amount of memory to be allocated.

If the call to `fdb_database_create_transaction()` fails (e.g., returns error), the memory allocated is automatically returned to the fast allocator. So there is no need to call `fdb_transaction_destroy()` in this case.