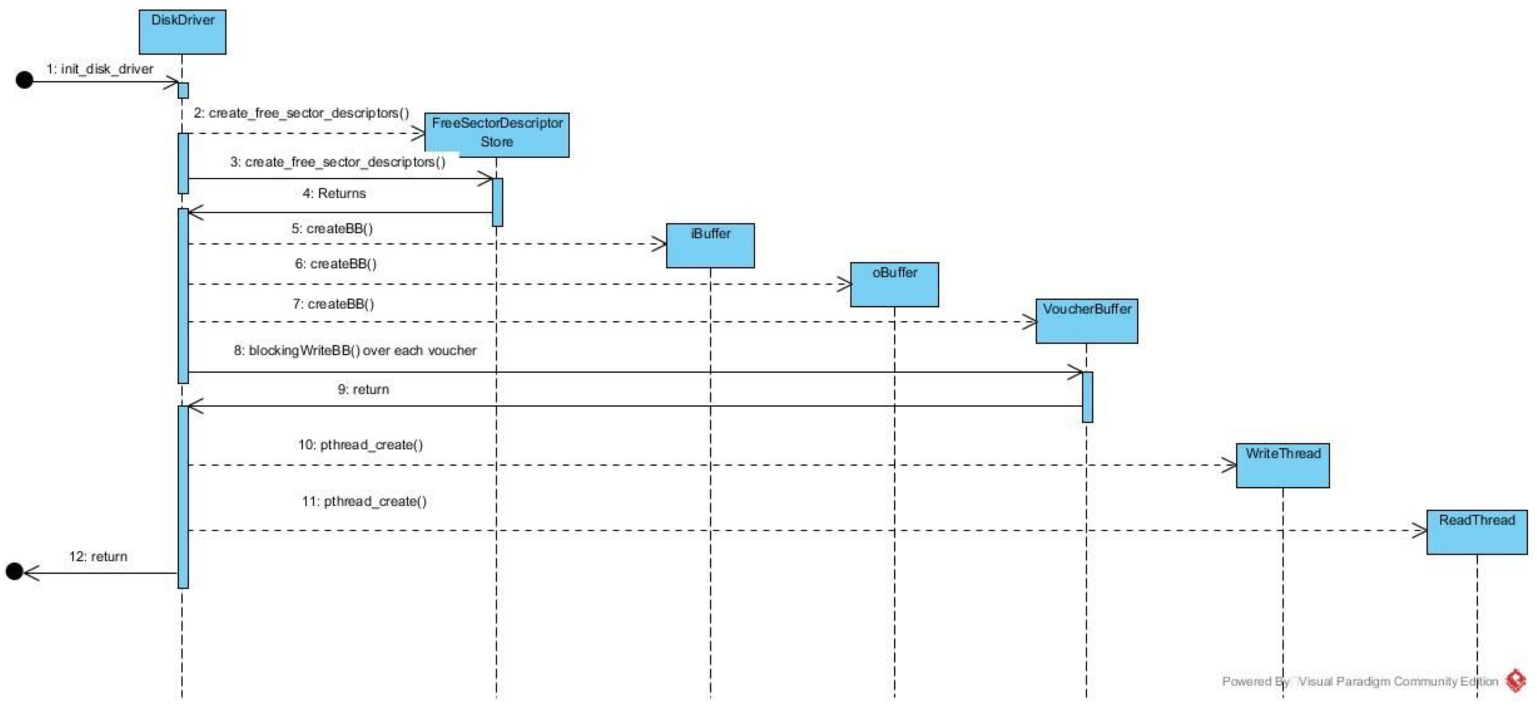
This system is working as far as I can tell. The system has a minimum and maximum allowed size for the sector descriptors. I did not get to test these bounds, though, because I could not change the test harness. It seems safe to say that the system can function at and in between the bounds. I am fairly confident that this code works without losing SectorDescriptors and Vouchers. I ran the code for two hours (and generated a whopping 12 Mb log file!) without a segfault nor infinite failures. It functioned until the end.

One peculiarity you may find is in redeem_voucher, which purposefully has a data race with either one of the worker threads. There is commentary showing why this is not detrimental, except for perhaps the busy waiting. There is a rationale for the granularity of the busy wait as well.

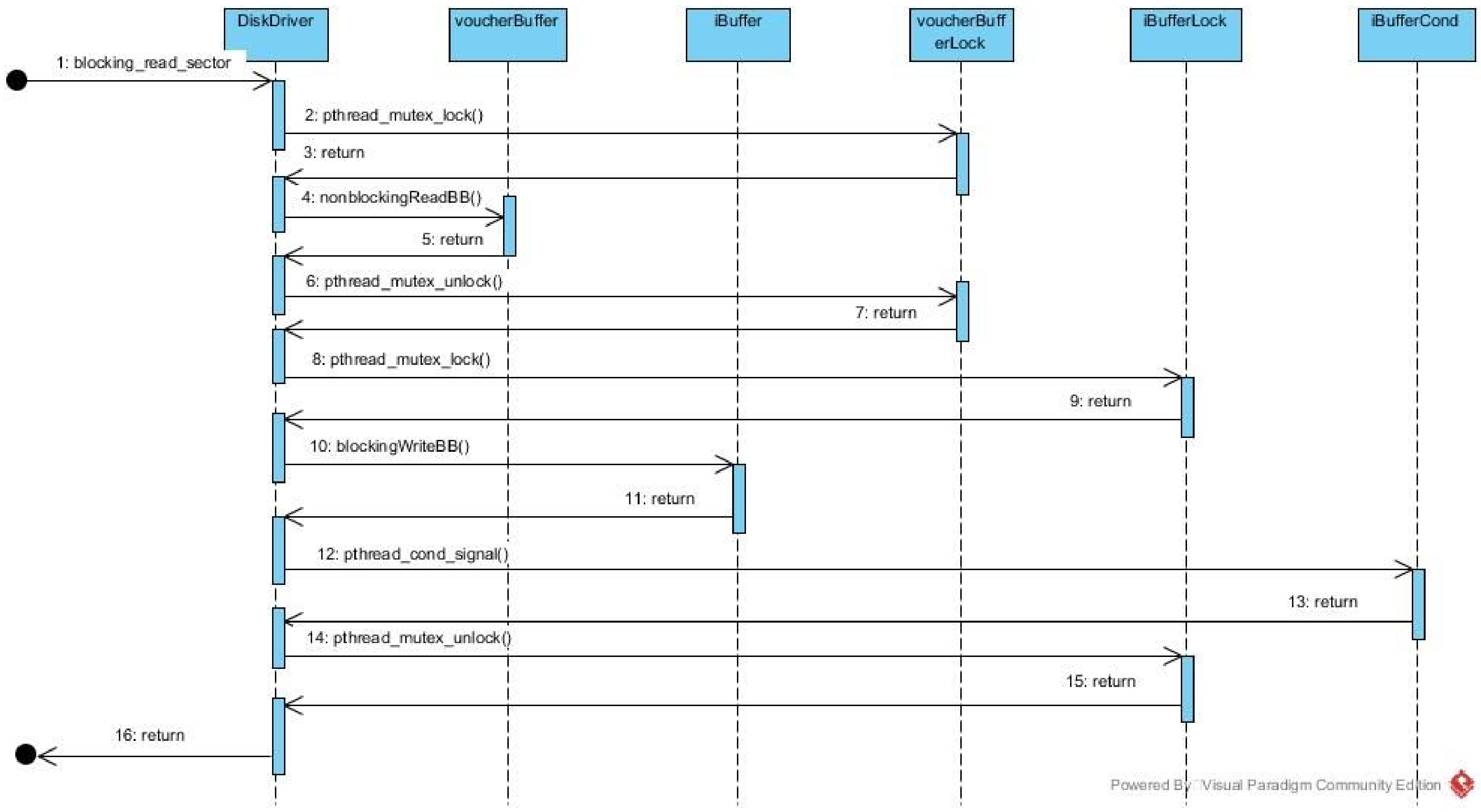
Another peculiarity you may find is in the designs shown below. There are only eight diagrams. This is due to the fact that init_disk_driver, bounded_read_sector, bounded_write_sector, and redeem_voucher do not have bad day scenarios. The init_disk_driver function in the system can force the system to crash, but that does not appear to be the intended meaning of “bad day scenario”. If that is included, then the meaning did not get across.

Design

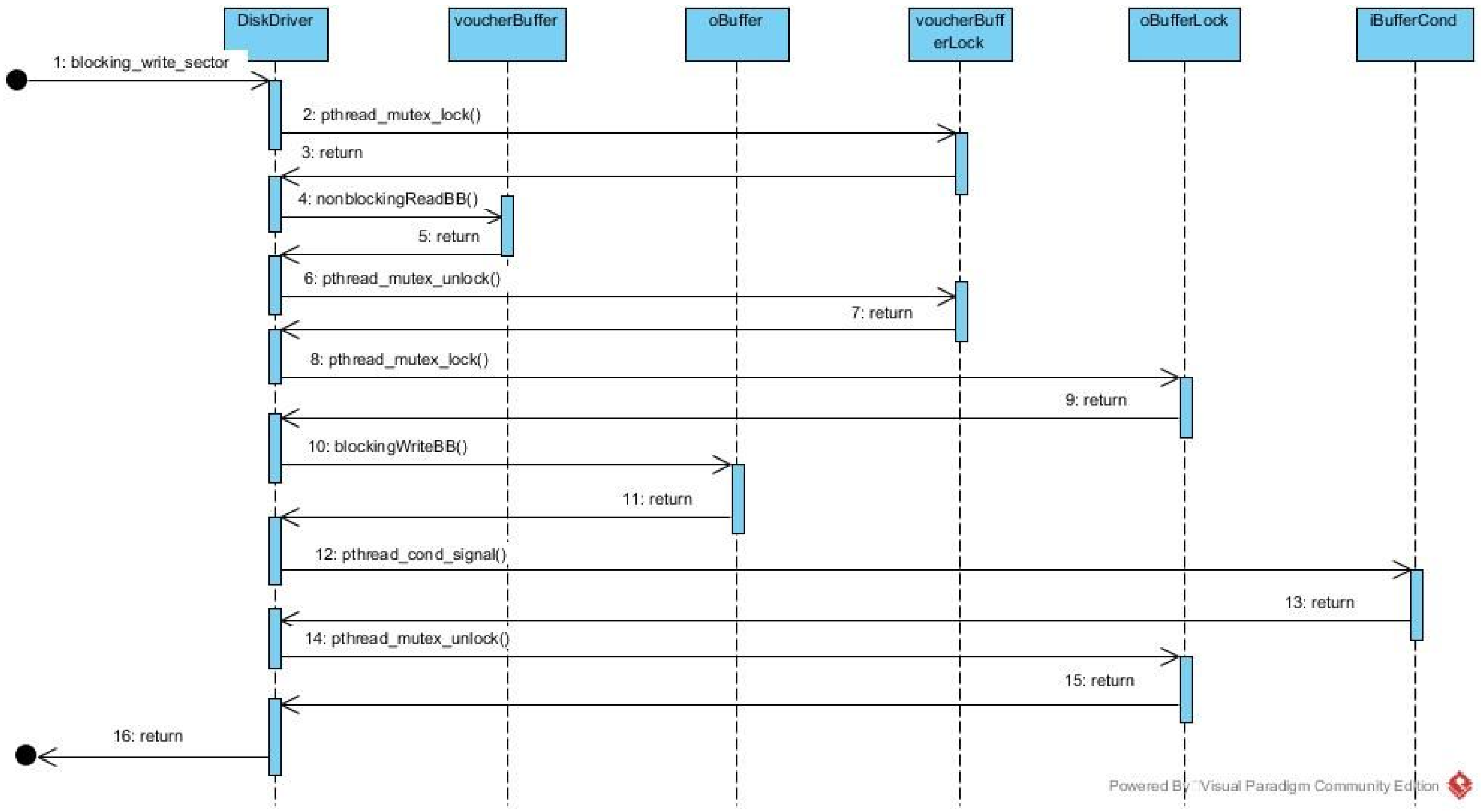
init_disk_driver



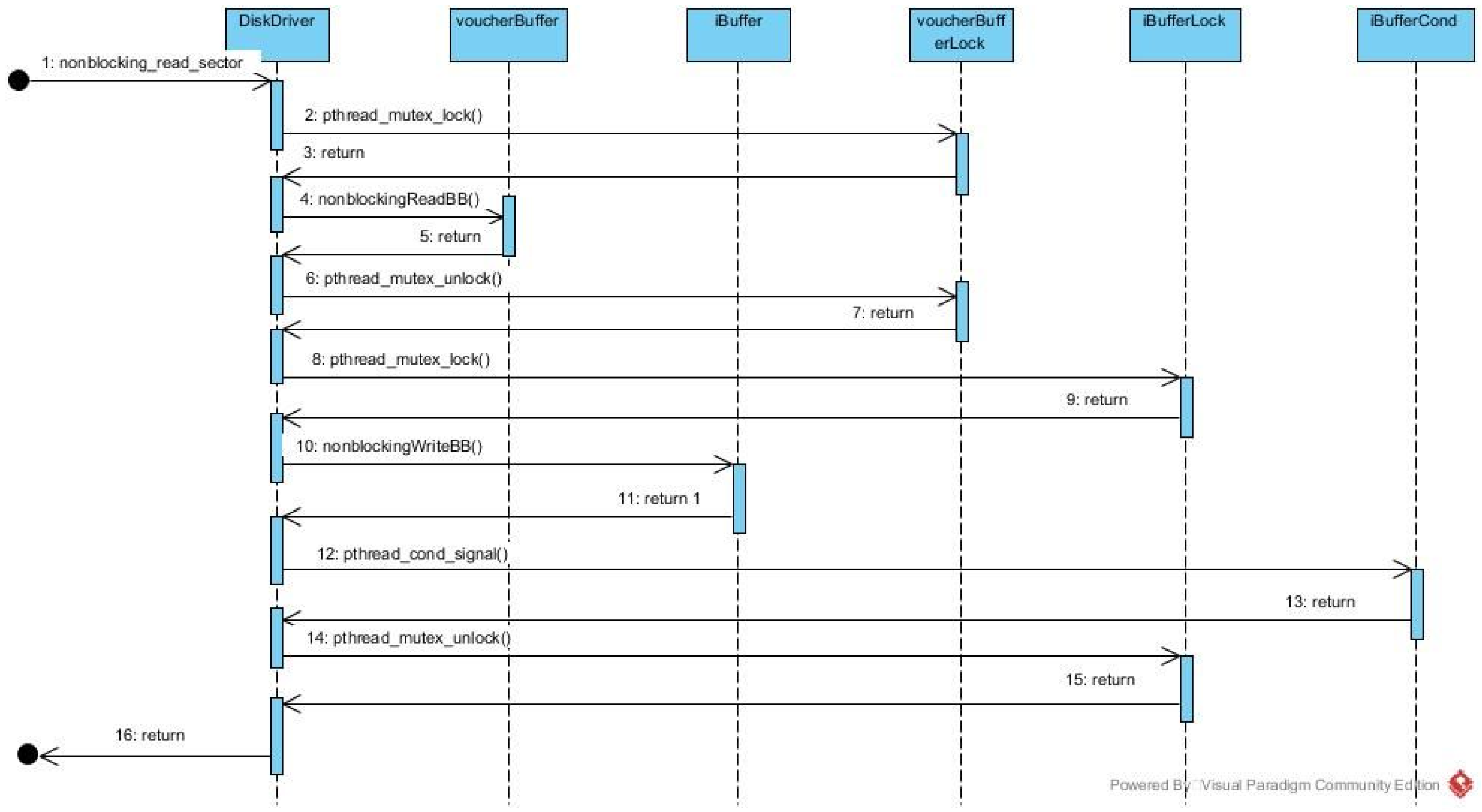
blocking_read_sector



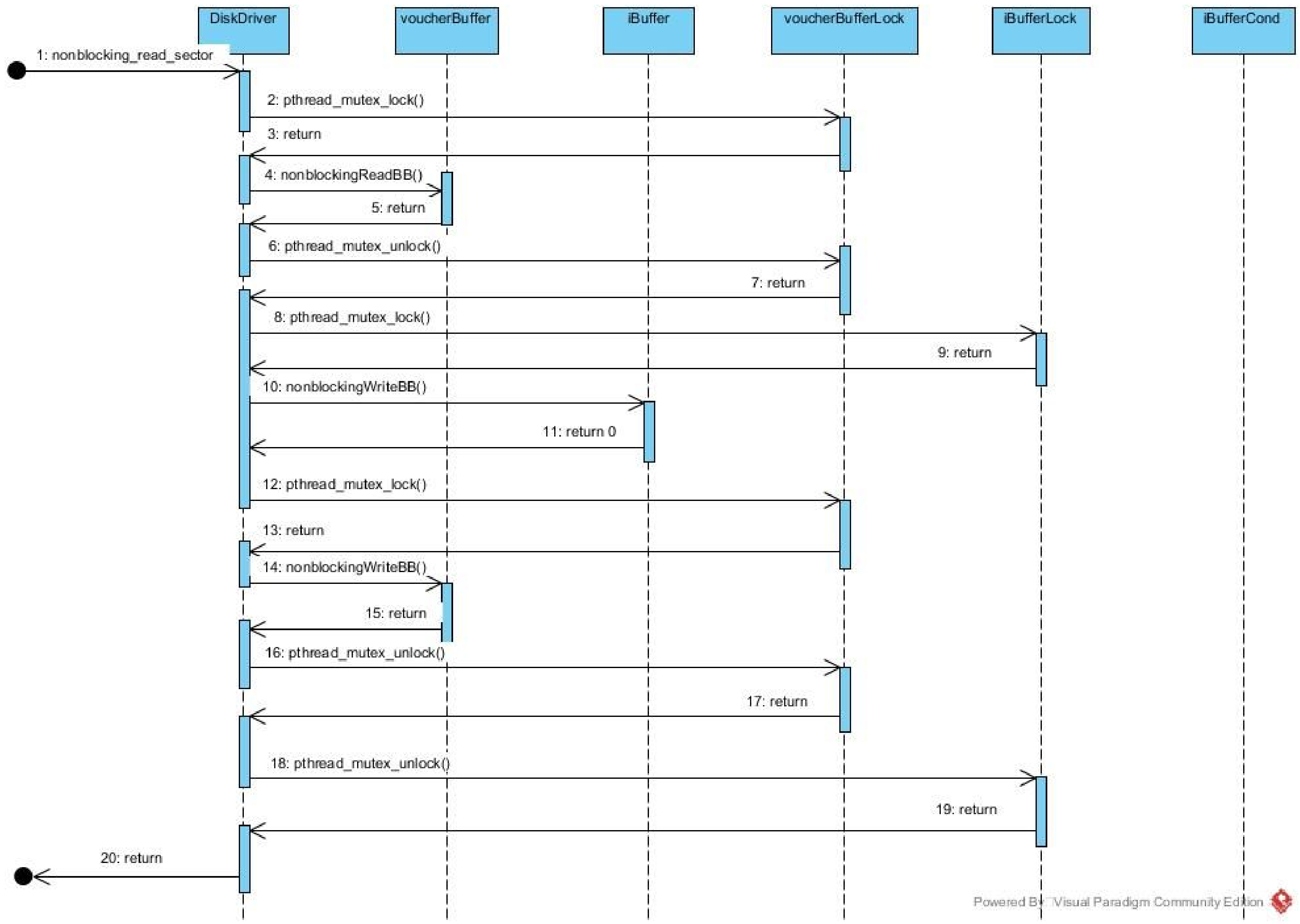
blocking_write_sector



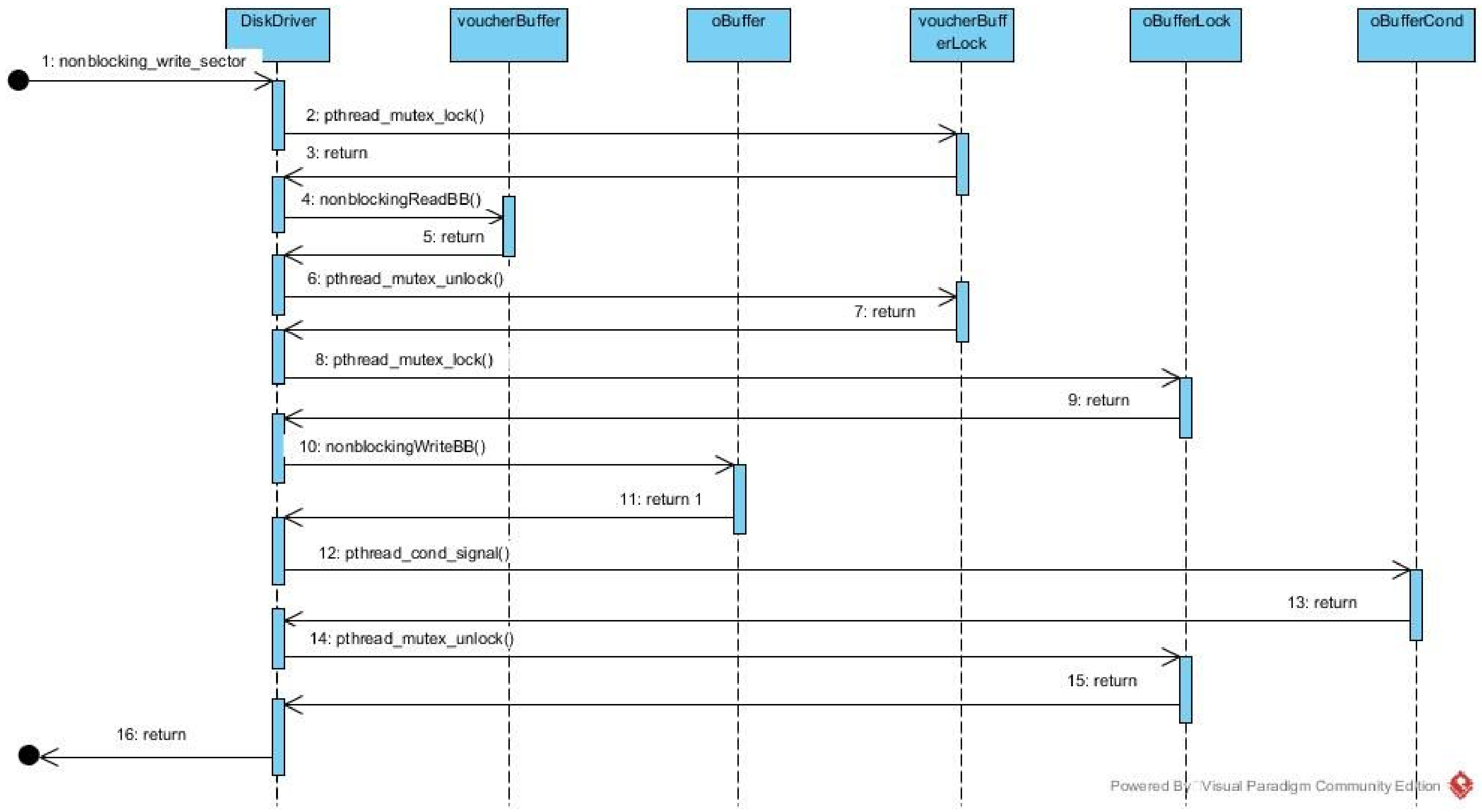
nonblocking_read_sector good



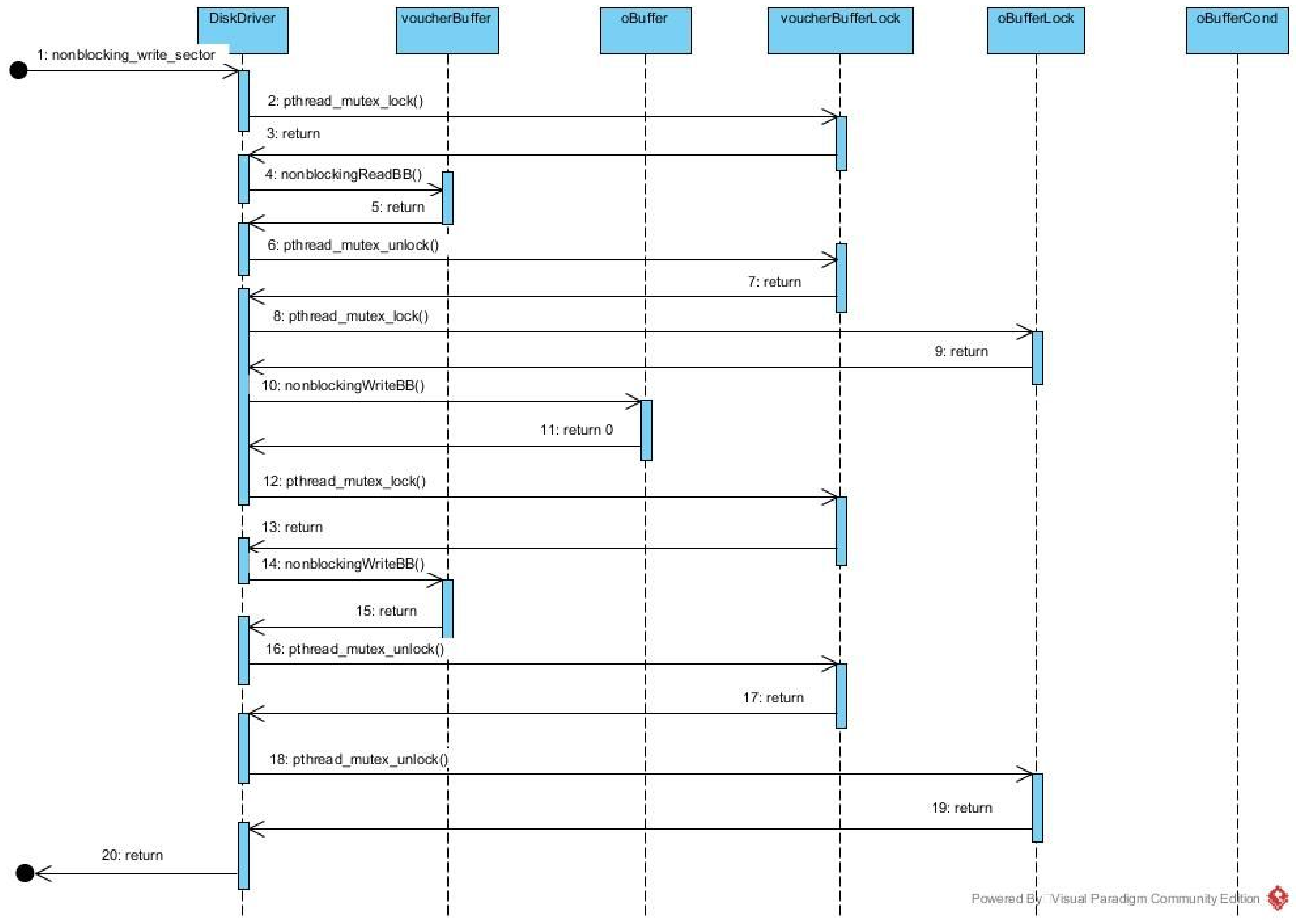
nonblocking_read_sector bad



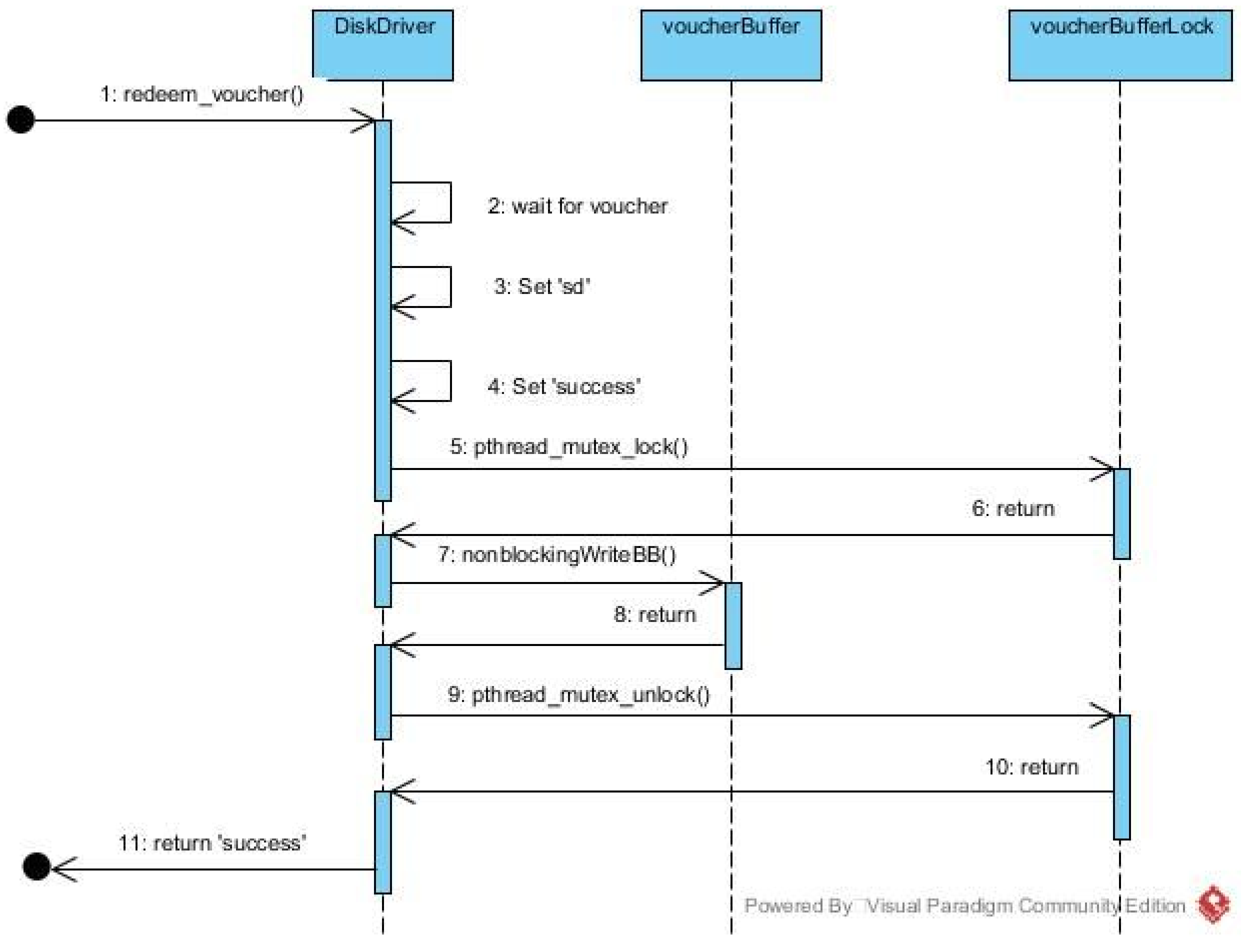
nonblocking_write_sector good



nonblocking_write_sector bad



redeem_voucher


DiskDriver
voucherBuffer
voucherBufferLock
1: redeem_voucher()
2: wait for voucher
3: Set 'sd'
4: Set 'success'
5: pthread_mutex_lock()
6: return
7: nonblockingWriteBB()
8: return
9: pthread_mutex_unlock()
10: return
11: return 'success'
Powered By Visual Paradigm Community Edition