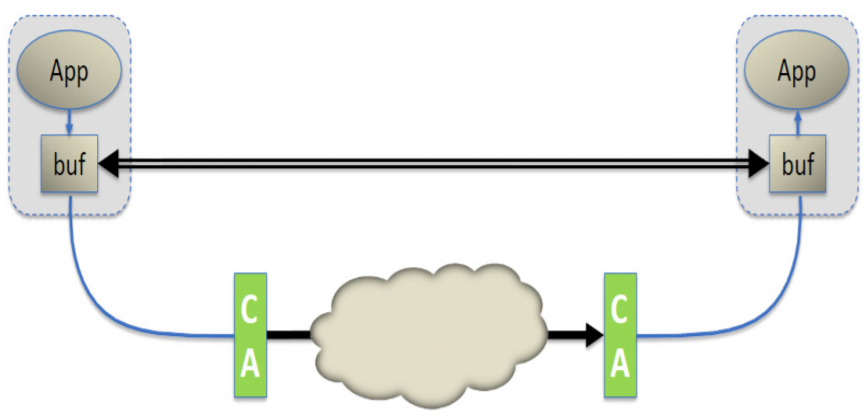
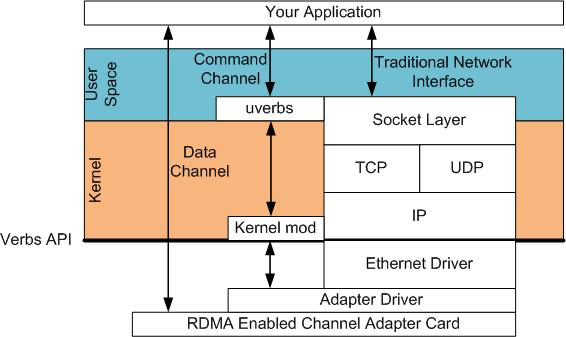
RDMA Tutorial

Механизм RDMA (Remote Dynamic Memory Access) [1] позволяет произвести прямое копирование участка памяти приложения одной машины в участок памяти приложения, работающего на другой машине. В отличие от стандартного обмена с использованием существующих сетевых протоколов, этот механизм позволяет существенным образом снизить задержку готовности данных и увеличить пропускную способность обмена между хостами.



Источник: https://www.mellanox.com/related-docs/prod\_software/RDMA\_Aware\_Programming\_user\_manual.pdf

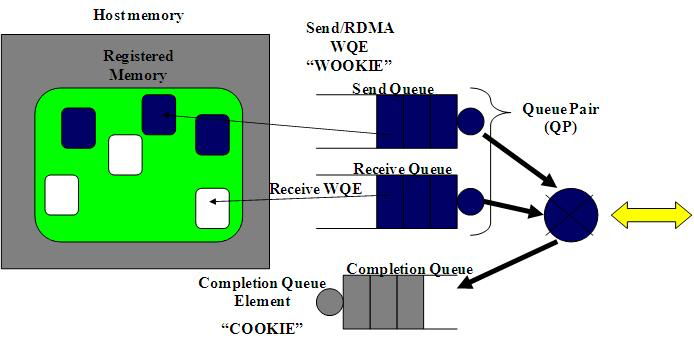
Для работы механизма RDMA необходимы соответствующие сетевые адаптеры, называемые Host Channel Adapter, HCA. Такой адаптер берет на себя всю работу по организации канала между шиной PCI Express и участком памяти приложения. С точки зрения приложения, механизм обмена информацией выглядит как обмен буферным участком памяти, выделенным для этой задачи.



Источник: https://zcopy.files.wordpress.com/2010/10/sample-rdma-protocol-stack.jpg

Обмен данными с использованием технологии RDMA отличается от технологии Ethernet.

Операции обмена данными RDMA начинаются с «закрепления» области памяти. Эта операция сообщает ядру операционной системы что участок памяти принадлежит конкретному приложению. После этого необходимо сообщить HCA, что можно подготовить канал обмена между картой и этой областью памяти (так называемая регистрация области памяти приложения в карте HCA). Теперь можно использовать данный участок памяти во всех операциях RDMA.



Источник: https://zcopy.files.wordpress.com/2010/10/qp-basic1.jpg

Технология RDMA для обмена данными использует три очереди – очередь отправки и приема (Send queue, SQ и Receive queue, RQ) отвечающие за планирование работы рабочей очереди (Work queue, WQ) и очередь завершения (Completion queue, CQ), используемая для уведомления о том, что инструкции, помещенные в рабочие очереди, были завершены. Очереди отправки и приема создаются парами (Queue pair, QP).

Пользователь помещает инструкции в рабочие очереди, которые сообщают HCA, какие буферы он хочет отправить или получить. Эти инструкции представляют собой небольшие структуры, называемые рабочими запросами или элементами рабочей очереди (Work Queue Elements, WQE, WOOKIE). WQE в первую очередь содержит указатель на буфер. WQE, помещенный в очередь отправки, содержит указатель на сообщение, которое нужно отправить. Указатель в WQE в очереди приема содержит указатель на буфер, в который может быть помещено входящее сообщение из сети.

RDMA — это асинхронный транспортный механизм. Таким образом, мы можем поставить в очередь несколько отправляемых или принимаемых WQE одновременно. Когда WQE обрабатывается, данные перемещаются. После завершения транзакции создается элемент очереди завершения (Completion Queue Element, CQE, COOKIE).

**В RDMA доступны следующие операции**

1. Отправка / немедленная отправка

Операция отправки позволяет отправлять данные в очередь приема удаленного получателя. Получатель должен предварительно разместить адрес буфера приложения для приема данных. Отправитель не контролирует, где данные будут находиться на удаленном хосте. Вместе с адресом буфера дополнительно может быть передано 4 байта данных. Эти данные предоставляются удаленному получателю как часть уведомления о получении и не содержится в буфере данных.

1. Получение

Удаленный хост получает уведомление о том, что получен адрес буфера (возможно с дополнительными 4 байтовыми данными). Получатель (приложение) отвечает за обслуживание буфера приема.

1. RDMA чтение

Операция считывания области памяти удаленного хоста. Перед выполнением этой операции удаленная сторона должна предоставить соответствующие разрешения доступа к своей памяти. Операция RDMA чтения (и записи) выполняется без какого-либо уведомления удаленной стороны.

1. RDMA запись / RDMA запись с значением

Операция аналогична RDMA чтению.

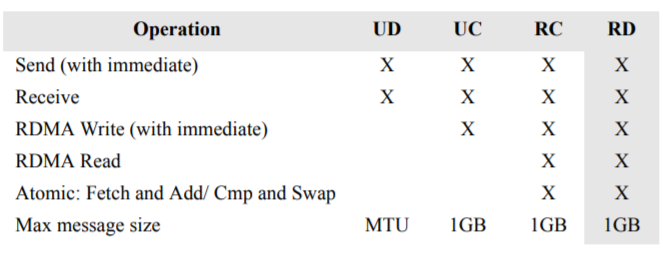
1. Атомарная выборка, добавление, сравнение и обмен

Это атомарные расширения операций RDMA. Операции атомарной выборки и добавления увеличивает значение по указанному виртуальному адресу на указанную величину. Значение, существующее до увеличения, возвращается вызывающей стороне.

Атомарное значение и своп будут поэлементно сравнивать значение по указанному виртуальному адресу с указанным значением, и, если они равны, указанное значение будет сохранено.

**Транспортные режимы RDMA**

RDMA поддерживает следующие транспортные режимы



Источник: https://www.mellanox.com/related-docs/prod\_software/RDMA\_Aware\_Programming\_user\_manual.pdf

1. Надежное соединение (Reliable Connection, RC)

QP связана только с одной QP. Сообщения, передаваемые очередью отправки одной QP надежно доставляются в очередь приема другой QP. Пакеты доставляются по порядку. RC-соединение очень похоже на TCP-соединение.

1. Ненадежное соединение (Unreliable Connection, UC)

QP связана только с одной QP. Соединение ненадежное, поэтому пакеты могут быть потеряны. Сообщения с ошибками повторно не отправляются, и обработка ошибок должна обеспечиваться протоколом более высокого уровня.

1. Ненадежная дейтаграмма (Unreliable Datagram, UD)

QP может передавать и принимать сообщения с одним пакетом в / от любой другой UD QP. Доставка не гарантируется, а доставленные пакеты могут быть отброшены получателем. Поддерживаются многоадресные сообщения. Соединение UD очень похоже на соединение UDP.

**Основные понятия, используемые в технологии RDMA**

1. Запрос на передачу (Send Request, SR)

SR определяет, сколько данных будет отправлено и куда. Для реализации SR используется структура struct ibv\_send\_wr.

1. Запрос на получение (Receive Request, RR)

RR определяет буферы, в которые должны приниматься данные для операций, отличных от RDMA. Если буфера не определены, и передатчик пытается выполнить операцию отправки или запись RDMA с немедленным получением, будет отправлено сообщение об ошибке по причине неготовности (RNR). Для реализации RR используется структура struct ibv\_recv\_wr.

1. Очередь завершения (Completion Queue, CQ)

Очередь завершения — это FIFO очередь, которая содержит завершенные рабочие запросы рабочей очереди (WQ). Каждая запись в очереди говорит о завершении определенного запроса из очереди WR. Очередь завершения - это механизм для уведомления приложения об информации о завершенной работе.

Запросы (статус, код операции, размер, источник) CQ имеют n записей очереди завершения (CQE). Количество CQE указывается в момент создания CQ. При запросе CQE удаляется из CQ.

CQ может обслуживать очереди отправки, очереди приема или и то, и другое.

Очередь WR из нескольких QP могут быть связаны с одним CQ. Для реализации CQ используется структура struct ibv\_cq.

1. Регистрация памяти (Memory Registration, MR)

Регистрация памяти — это механизм, который позволяет приложению описывать набор виртуально непрерывных ячеек памяти или набор физически непрерывных ячеек памяти для сетевого адаптера как виртуально непрерывный буфер с использованием виртуальных адресов. В процессе регистрации страницы памяти закрепляются (чтобы страницы не выгружались и сохранилось физическое <-> виртуальное отображение).

Во время регистрации ОС проверяет права зарегистрированного блока. В процессе регистрации таблица виртуальных адресов записывается в физический сетевой адаптер. При регистрации памяти выставляются разрешения для региона. Разрешения - локальная запись, удаленное чтение, удаленная запись, атомарная запись и привязка.

Каждый MR имеет удаленный и локальный ключ (r\_key, l\_key). Локальные ключи используются локальным HCA для доступа к локальной памяти, например, во время операции приема. Удаленные ключи передаются удаленному HCA, чтобы разрешить удаленному процессу доступ к системной памяти во время операций RDMA.

Один и тот же буфер памяти может регистрироваться несколько раз (даже с разными правами доступа), и каждая регистрация приводит к другому набору ключей. Для реализации регистрации памяти используется структура struct ibv\_mr.

1. Окно памяти (Memory Window, MW)

MW позволяет приложению более гибко управлять удаленным доступом к своей памяти. Окна памяти предназначены для ситуаций, когда приложение:

* хочет предоставлять и отзывать права удаленного доступа к зарегистрированному региону динамическим образом с меньшими потерями производительности, чем при отмене регистрации / регистрации или перерегистрации;
* хочет предоставить разные права удаленного доступа разным удаленным агентам и / или предоставить эти права на различные диапазоны в пределах зарегистрированного региона.

Операция связывания MW с MR называется привязкой. Различные MW могут перекрывать один и тот же MR (событие с разными разрешениями доступа).

1. Вектор адресов (Address Vector, AV)

Вектор адресов — это объект, который описывает маршрут от локального узла к удаленному. В каждом UC/RC QP есть адресный вектор в контексте QP. В UD QP вектор адресов должен быть определен в каждом запросе SR. Для реализации используется структура struct ibv\_ah.

1. Заголовок глобальной маршрутизации (Global Routing Header, GRH)

GRH используется в IB и RoCE для маршрутизации между подсетями.

Когда глобальная маршрутизация используется в UD QP, GRH будет содержаться в первых 40 байтах буфера приема. Эта область используется для хранения информации о глобальной маршрутизации, поэтому соответствующий вектор адресов может быть сгенерирован для ответа на полученный пакет. Если GRH используется с UD, RR всегда должно быть доступно 40 дополнительных байтов для этого GRH. Для реализации используется структура struct ibv\_grh.

1. Домен защиты (Protection Domain, PD)

Объект, компоненты которого могут взаимодействовать только друг с другом. Этими компонентами могут быть AH, QP, MR и SRQ.

Домен защиты используется для связывания QP с областями памяти и окнами памяти (MW) в качестве средства для включения и управления доступом сетевого адаптера к памяти хост-системы.

PD также используются для связывания пар очередей ненадежных датаграмм с дескрипторами адресов, как средства контроля доступа к пунктам назначения UD. Для реализации домена защиты используется структура struct ibv\_pd.

1. Асинхронные события (Asynchronous Events, AE)

Сетевой адаптер может отправлять асинхронные события чтобы информировать SW о событиях, произошедших в системе. Существуют два типа асинхронных события:

- Аффилированные события: события, произошедшие с персональными объектами (CQ, QP, SRQ). Эти события будут отправлены в конкретный процесс.

- Не аффилированные события: события, произошедшие с глобальными объектами (сетевой адаптер, ошибка порта). Эти события будут отправлены всем процессам.

1. Сбор разбор (Scatter Gather)

Данные собираются / распределяются с использованием элементов сбора разбора, которые включают:

- Адрес: адрес локального буфера данных, из которого данные будут собираться или в который будут распространяться.

- Размер: размер данных, которые будут считываться / записываться по этому адресу.

- L\_key: локальный ключ MR, который был зарегистрирован в этом буфере.

Элементы сбора разбора реализует структура struct ibv\_sge.

1. Опрос (Polling)

Опрос CQ для завершения — это получение сведений о WR (отправка или получение), который был

опубликовал. Если у нас есть завершение с плохим статусом в WR, все остальные завершения будут плохими (и рабочая очередь будет переведена в состояние ошибки).

Каждый WR, у которого нет завершения (который был опрошен), все еще остается невыполненным.

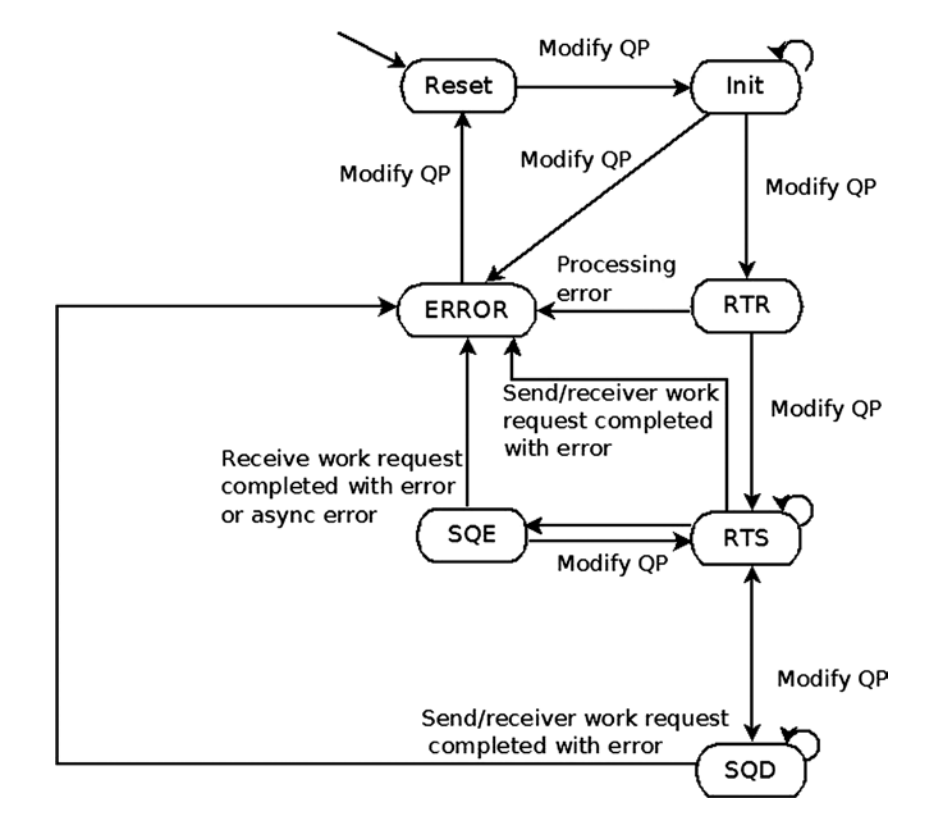
Только после того, как WR завершится, буфер отправки / приема может быть использован / повторно использован / освобожден.

Статус завершения всегда следует проверять. Когда CQE опрашивается, он удаляется из CQ. Опрос выполняется с помощью операции ibv\_poll\_cq.

1. Парная очередь (Queue Pair, QP)

QP представляет из себя пару очередей отправки и получения. Очередь оправки предназначена для отправки исходящих сообщений, а очередь получения – для получения входящих сообщений.

В связи с тем, что это ключевой объект, участвующий во взаимодействии, далее приведено более подробное описание.



Источник: Linux Kernel Networking: Implementation and Theory

После создания, очередь переходит в статус Reset.

Для перевода QP в рабочее состояние, она должна перейти в статус Ready to Receive, RTR или Ready to Send, RTS. Для этого используется ibv\_modfiy\_qp().

**Структура типичного приложения RDMA**

Для взаимодействия с адаптером (Infiniband Host Chanel Adapter) в пространстве пользователя можно использовать Verbs API [1]. Для Linux систем разработкой и поддержкой Verbs API занимается Open Fabrics Enterprise Distribution (OFED, [www.openfabrics.org](http://www.openfabrics.org)) [4]. Для Windows существует проект WinOF.

При использовании Verbs API, для организации обмена сначала необходимо настроить канал команд. Настройка канала команд в RDMA происходит с использованием драйвера, работающего на уровне ядра операционной системы. Канал команд используется для создания каналов обмена данными, которые позволяют в дальнейшем осуществлять перемещение информации в обход ядра операционной системы.

Для организации взаимодействия необходимо выполнить следующую последовательность:

1. Получение перечня устройств. Сначала необходимо получить перечень InfiniBand (IB) устройств локального хоста. Каждое устройство в данном списке содержит имя и Guid. Например, имя устройство в системе может быть mthca0 или mlx4\_1.
2. Открытие выбранного устройства. Полученный список устройств перебирается на основании определенных критериев и открывается с использованием имени или Guid.
3. Произвести опрос возможностей открытого устройства. Это позволит получить информацию о поддерживаемых опциях (например APM или SRQ).
4. Выделение защитного домена для содержащего ресурсы приложения. Защитный домен (PD) позволит приложению ограничить взаимодействие только с выбранными ресурсами. Этими компонентами могут быть AH, QP, MR, MW или SRQ.
5. Зарегистрировать область памяти. RDMA работает только с зарегистрированной областью памяти. Можно зарегистрировать любой буфер в виртуальном пространстве памяти процесса. В процессе регистрации приложение устанавливает права доступа к памяти и получает локальные и удаленные ключи (lkey / rkey), которые позже будут использоваться для ссылки на этот буфер памяти.
6. Создание очереди завершения (CQ). CQ содержит завершенные рабочие запросы (WR). Каждый WR будет генерировать запись в очередь завершения (CQE), которая помещается в очередь CQ. CQE укажет, была ли WR завершена успешно или нет.
7. Создание очереди QP. Это подразумевает также создание очереди приема и передачи.
8. Поднятие QP. Созданную QP можно использовать только после прохождения ей всех необходимых статусов достигнув статуса RTS (Ready to Send).
9. Публикация запросов на работу и опрос на предмет завершения. Созданная на предыдущих шагах QP используется для обмена данными.
10. Очистка занимаемых ресурсов. Удаление QP, CQ, отмена регистрации MR, деаллокация PD, закрытие устройства.

**Практическая часть.**

В качестве HCA будем использовать двухпортовый MCX456A-ECA или однопортовый MCX455A-ECA).

Данные адаптеры установлены на Linux машине под управлением Ubuntu 20.04.

Обновляем версию микропрограммного обеспечения, для этого с сайта Mellanox устанавливаем необходимую утилиту [2].

*# wget http://www.mellanox.com/downloads/firmware/mlxup/4.17.0/SFX/linux\_x64/mlxup*

*# chmod +x mlxup*

*# ./mlxup*

*Querying Mellanox devices firmware ...*

*Device #1:*

*----------*

*Device Type: ConnectX4*

*Part Number: MCX456A-ECA\_Ax*

*Description: ConnectX-4 VPI adapter card; EDR IB (100Gb/s) and 100GbE; dual-port QSFP28; PCIe3.0 x16; ROHS R6*

*PSID: MT\_2190110032*

*PCI Device Name: 0000:01:00.0*

*Base GUID: 0000000000000123*

*Base MAC: 000000001234*

*Versions: Current Available*

*FW 12.28.2006 12.28.2006*

*PXE 3.6.0102 3.6.0102*

*UEFI 14.21.0017 14.21.0017*

*Status: Up to date*

В случае, если в строке Status отображается «Update required», запускаем утилиту с ключом online, либо отвечаем положительно на запрос Perform FW update? [y/N]: Y

*Device #1: Updating FW ...*

*Initializing image partition - OK*

*Writing Boot image component - OK*

*Done*

После обновления перезагружаем систему.

Устанавливаем необходимые утилиты и пакеты

*# apt-get install -y rdma-core ibverbs-utils libibverbs-dev infiniband-diags mstflint*

Необходимые для сборки заголовочные файлы появляются в каталоге /usr/include/infiniband/

Перечень устройств можно посмотреть с использованием команды

# lspci | grep Mellanox

01:00.0 Infiniband controller: Mellanox Technologies MT27700 Family [ConnectX-4]

01:00.1 Infiniband controller: Mellanox Technologies MT27700 Family [ConnectX-4]

01:00.0 – первый порт устройства

Посмотреть детальные свойства 1 порта адаптера можно с использованием команды

# mstflint -d 01:00.0 q

Image type: FS3

FW Version: 12.28.2006

FW Release Date: 15.9.2020

Product Version: 12.28.2006

Rom Info: type=UEFI version=14.21.17 cpu=AMD64,AARCH64

type=PXE version=3.6.102 cpu=AMD64

Description: UID GuidsNumber

Base GUID: 0000000000000123 4

Base MAC: 000000001234 4

Image VSD: N/A

Device VSD: N/A

PSID: MT\_2190110032

Security Attributes: N/A

Или команды

# mstconfig -d 01:00:0 q

Device #1:

----------

Device type: ConnectX4

Name: MCX456A-ECA\_Ax

Description: ConnectX-4 VPI adapter card; EDR IB (100Gb/s) and 100GbE; dual-port QSFP28; PCIe3.0 x16; ROHS R6

Device: 01:00.0

Configurations: Next Boot

Свойства адаптера можно посмотреть командой

# ibstat

CA 'ibp1s0'

CA type: MT4115

Number of ports: 1

Firmware version: 12.28.2006

Hardware version: 0

Node GUID: 0x7cfe900300c6d42a

System image GUID: 0x7cfe900300c6d42a

Port 1:

State: Initializing

Physical state: LinkUp

Rate: 100

Base lid: 65535

LMC: 0

SM lid: 0

Capability mask: 0x2651e848

Port GUID: 0x7cfe900300c6d42a

Link layer: InfiniBand

Наименование устройства, которое будет использоваться в программном обеспечении – ibp1s0. Наименование зависит от различных факторов и может отличаться. Например, второй сервер с установленным двухпортовым адаптером имеет следующие наименования:

# ibstat | grep CA

CA 'ibp1s0f0'

CA 'ibp1s0f1'

В качестве среды разработки будет использоваться Visual Studio 2019, установленная на Windows 10. Соответственно, исполнение кода и отладка будут выполняться в удаленном режиме.

Для настройки Visual Studio 2019 с удаленной Linux машиной можно воспользоваться инструкцией [5].

На удаленную машину устанавливаем инструменты сборки и отладки

# apt install g++ gdb make build-essential rsync zip

На локальной машине в настройках «Каталоги VC++» должны быть указанными обязательно следующие каталоги

Включаемые каталоги - /usr/include/infiniband/

Каталоги библиотек /usr/lib/x86\_64-linux-gnu/libibverbs;

В настройках компоновщика в свойстве «Дополнительные зависимости» должна быть указана библиотека /usr/lib/x86\_64-linux-gnu/libibverbs.so.

Примеры проектов доступны по адресу [6].

**Tutorial01** - Получение перечня устройств InfiniBand HCA.

**Tutorial02** – Открытие выбранного устройства InfiniBand HCA и опрос его возможностей.

**Tutorial03** – Выделение защитного домена PD, создание очереди завершения CQ, регистрация области памяти MR, создание QP.

**Tutorial04** – Изменение состояния QP

Список используемой литературы

1. RDMA Aware programming user manual

<https://www.mellanox.com/related-docs/prod_software/RDMA_Aware_Programming_user_manual.pdf>

1. Mellanox Update and Query Utility

<https://www.mellanox.com/support/firmware/mlxup-mft>

1. NVIDIA ConnectX-4 VPI Adapter Cards User Manual

<https://docs.nvidia.com/networking/display/ConnectX4IB>

1. OpenFabrics LibIbVerbs

<https://www.openfabrics.org/downloads/libibverbs/>

1. Удаленная отладка проекта C++ в Visual Studio

<https://docs.microsoft.com/ru-ru/visualstudio/debugger/remote-debugging-cpp?view=vs-2019>

1. <https://github.com/sgunin/RDMATutorial.git>
2. Linux Kernel Networking: Implementation and Theory, Rami Rosen