决赛题目

题目标题：撮合模拟程序设计大赛

欢迎来到九坤第五届 UBIQUANT CHALLENGE 量化新星挑战赛第二期，在这里你将真正接触到九坤IT的工作内容，和你的团队一起攻克难关。

引子

作为一家量化公司，九坤每天都在和交易打交道，为了阐述本题需要你们完成的任务，我们首先会给出一些基本的关于撮合交易的内容。

完成交易的基本元素是一条条Order（申报），下面是一个简化后的Order结构。

```
struct order {
    int stk_code;
    int order_id;
    int direction;
    int type;
    double price;
    int volume;
};
```

接下来解释Order结构中各个字段的意义：

- stk_code代表一支股票的代码，
- order_id代表本order的id值，注意不同股票之间的order_id相互独立，order_id从1开始逐渐递增，id越小代表order越早产生，
- direction代表order是买入还是卖出，这里记direction=1代表买入，direction=-1代表卖出，
- type代表这个order的类型，一共有6种order，包括限价申报（类型0），对手方最优价格申报（类型1），本方最优价格申报（类型2），最优五档即时成交剩余撤销申报（类型3），即时成交剩余撤销申报（类型4），全额成交或撤销申报（类型5）。
- price代表当前order的bid/ask的价格，仅对于现价申报有意义，请注意，python中定义一个float，例如a=1.23，占用8个字节。
- volume代表当前order的bid/ask的数量，本题中的volume可以看作一个最小有效申报数量的倍数（例如100股），所以volume=1,2,3...总是有效的。

在这道题中，所有的order采用**连续竞价**的方式进行撮合交易。相关的成交规则可以参考[1]。

order进行撮合交易后得到的结果被称为Trade结构，下面是一个Trade的内容

```
struct trade {
    int stk_code;
    int bid_id;
    int ask_id;
    double price;
    int volume;
};
```

接下来解释Trade结构中各个字段的意义：

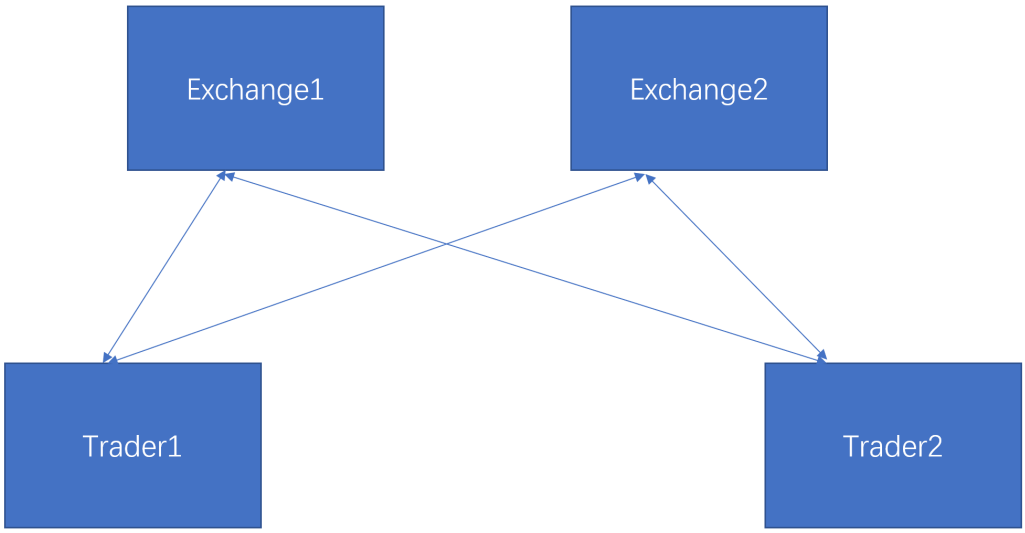
- 同Order结构，由某一支股票的order产生的trade会拥有一样的stk_code
- bid_id，代表此trade产生时对应的买方的order_id

- ask_id，代表此trade产生时对应的卖方的order_id
- price，代表撮合成功时的交易价格
- volume，代表撮合成功时的交易量

在了解了基本的关于撮合的内容后，接下来介绍本题的任务。

任务总览

本题的任务是构建一个微型交易系统进行撮合的过程，这个交易系统的框架如下图所示：



在本题中，你会得到4个服务器，其中的两个是Trader机器，另外两个是Exchange机器。Trader机器中会提供交易所需要的Order数据，Exchange机器通过网络接收这些Order然后在本机进行撮合，每次完成一笔撮合就将生成的trade结构回传到两台Trader机器上。最后在两台Trader机器上，对于每支股票（stk_code不同），都会有一个trade的序列，并且两台机器上的结果应当完全一致，我们会根据这个序列的正确性来进行给分。

下面具体介绍各个部分。

模拟撮合

正如之前所说，本题的输入是已经准备好的一系列Order，这些Order被保存在两台Trader机器上。我们会提供5个矩阵作为数据，一个记作Price矩阵，一个被记作Order Id矩阵，一个被记作Volume矩阵，一个记作Type矩阵，最后一个记做Direction矩阵，这5个矩阵的维度均为$1000 \times 1000 \times 1000$，每个矩阵的前$500 \times 1000 \times 1000$在Trder1上，后$500 \times 1000 \times 1000$都在Trader2上。

由这些矩阵来生成Order的方式如下：对于某个Order，其在所有矩阵中的坐标均是（x, y, z），并且stk_code = (x % 10) + 1。举个例子，在Trader1机器上坐标为（103，102，677）的位置，从5个矩阵中分别取出对应的order_id，price，volume，type，direction，然后根据x计算出对应的stk_code=4，就得到了一个order。这些矩阵用hdf5的方式存储。由于存在涨跌幅限制，除了这5个矩阵我们还会给出相应的前收盘价，储存在price矩阵中（两个price矩阵中都有）。本题限制涨跌幅范围为10%，最小变动单位为0.01。

本题的输出是每一个stk_code所对应的Trade序列，在每个Trader机器上都要有全量的Trade序列，直接保存成二进制的形式，在保存时请不要加入填充，保存下来的每一个trade的大小应该为24字节（c++使用**attribute**((packed)), python可以使用struct库的'='选项）。对于输入输出我们会给出相应的example code。

在Trader机器上得到了输入流之后，Trader机器将这些机器传输到Exchange机器上进行撮合交易。整个撮合交易的规则遵循深交所的交易规则。值得注意的是，在Order之间在进行撮合交易时必须严格按照Order Id的顺序进行交易（Order Id越小代表这个Order的实际报单时间越早），但是在给出的数据矩阵中给出的order并不是保序的，order id为1的order可能和order id为2的order在不同的Trader机器上。在进行撮合前**请务必确认order按照order id递增的顺序进入撮合。**Exchange机器上进行撮合生成了trade之后需要把trade序列回传到两台Trader机器。

同时，在正常交易时我们会根据之前的撮合结果结合策略信号来生成报单，本题不涉及交易策略，为了模拟这个过程，我们额外提供了一个$10 \times 100 \times 4$的hook矩阵（两台Trader机器上均存在），这个矩阵中的某个元素（x, y, z），（x+1）代表这个元素对应的stk_code，y代表100个特殊的order，这100个order会根据之前生成的trade信息来判断这个order是否需要进入撮合。hook矩阵中每一行的意义如下[self_order_id, target_stk_code, target_trade_idx, arg]，有一个self_order_id代表这个特殊order的order_id，这个order需要根据已有的trade序列进行判断是否进入撮合，target_stk_code代表目标trade序列所属的stk_code，target_trade_idx代表需要的trade在这个trade序列中的idx（从1开始），最后的arg是一个用于判断的信号量，判断规则如下：如果目标的trade的volume值（撮合成交量）**小于等于**arg，那么这个order才会进入撮合。

Example：对于stk_code = 2，他的第一个特殊order的order id=1000，拥有这个order的Trader会从hook中读取这个order需要比较的trade，这个trade有可能是stk_code=7的第10个trade（保证不会出现数据死锁），如果这个trade一直没有从Exchange机器回传，那么stk_code=2的这支股票只能一直等待。在收到trade后，根据trade中的volume信息（成交量）来判断这个order是否发送到Exchange。如果判断不发送，那么此时对于处理stk_code=2的Exchange，他收到的order的id顺序为998，999，1001....，即不处理1000这个order。

网络状况

本题中，两台Trader机器之间没有网络连接，两台Exchange机器之间也没有网络连接，Trader机器和Exchange机器之间有网络连接。**请使用tcp的方式进行网络传输。**

对于一台Trader机器和一台Exchange机器，我们仅仅开放三条可行的通道，即指定Trader机器的某一个port，通过这个port只能连接Exchange机器的某一个指定端口，**对于一对Trader和Exchange，仅仅有三个这样的端口对**，即Trader和Exchange之间仅仅可以建立三条tcp连接（在决赛正式开始前我们会在邮件里通知每支队伍的服务器ip和开放的通道）。因为指定了tcp连接两端的端口，在建立tcp连接时connect端需要通过bind来指定一个port，而不是使用操作系统分配的port（参考nc命令的-p选项）。

同时，在比赛的过程中，我们会通过软件的方式控制每条通道的网络状态，来模拟实际情况下物理链路的波动。运行时两台服务器之间的网络会发生的情况：

- **网络拥塞**
- **网络断链**
- **服务器端口重映射（例如本来的10.192.108.1:12345 -> 10.192.108.2:54321 切换为10.192.108.1:60023 -> 10.192.108.2:12112）**
- **网络出现延迟、丢包、包乱序、重包**

以上情况中，只有**服务器端口重映射**会在发生时进行通知，其余情况均需要自行发现处理。

在设计网络传输组件时，请先想好每种情况的应对方法，在**确保正确的情况下，能够快速的发送数据完成任务。**初始时，每条channel在两个方向上的带宽均为1.5MB/s，在比赛期间会出现波动。

关于比赛形式：

- 相信你们小队的人员已经完全确定，你们一共有两周的时间，这两周时间内我们会提供四台服务器供你们测试，并且会提供给你们用于测试的数据。
- 我们会组织线下的正式比赛，在线下的比赛中部署自己的程序，然后统一开始自己的程序进行任务。
- 在进行任务的过程中，也许你们的程序会因为某些情况而报错，卡死。进行比赛时，你们需要判断出现了什么情况，重启你的程序，或者临时修改你们的程序，因此你们的程序需要做好充分的log来应对随时可能出现的问题。只要结果正确，无论在任务执行的过程中你们怎么样操作自己的程序我们都不会关心。
- 整个线下比赛时，进行任务的时间限制为最长45min。

Hint

- 整个任务可以分为算法、传输、IO三大块。
- 算法方面请大家仔细阅读深交所规则，来编写撮合的算法代码，请仔细阅读深交所规则（尤其是第三章）的内容。
- 传输方面。在成功建立出tcp连接的情况下，有这些内容需要注意：
    - IO多路复用
    - 在某些通道带宽变化时，如何不让某个通道的传输影响整个任务的进度
    - 如果在传输的过程中网络突然闪断了，如何在重连时判断已经发送的进度
    - 如果出现了必须重启应用的情况，能否接着之前的进度进行传输
- IO方面。
    - 如何快速的获取有序数据。
    - 如果需要备份一部分order或者trader数据，应当采取怎样的方式存储，同时在缓存下来的一部分数据可能会面临数据竞争的问题，如何解决。
    - 理论上IO应当远比网络快，所以优先优化传输组件是一个更好的选择。

评分标准：

评分分为四部分，基本分100分，奖励分20分，惩罚分（待定），设计分40分：

- 基本分：最后的trade序列的正确性。如果仅有部分正确，按照正确的trade结构的百分比给分，并且无法拥有奖励分。
- 奖励分：根据完成的时间，程序的cpu使用峰值均值，程序的内存峰值均值，来进行奖励，奖励分的前提是trade序列完全正确
- 设计分：根据每支队伍提交的报告来进行给分。主要由两方面来进行给分，第一部分异常恢复，第二部分是可扩展性。
- 惩罚分：每支队伍选出一名“运维”，“运维”可以在比赛期间随意控制程序，在限时进行任务的过程中出现“运维”自己无法解决的问题，需要场外队友帮助的情况，每次出现一次进行惩罚。（由于线上比赛相对比较难以操作，此条可能失效）

关于提交

两周时间结束后每支队伍需要提交代码和报告到主办方邮箱内，主办方会在决赛当天将这些提交的文件拷贝到比赛环境中。

- 代码提交：每支队伍都需要提交全部的代码，如果使用了第三方库请使用submodule的形式放在代码中。
- 报告提交：每支队伍都需要提交一份简要的报告，报告中需要阐述自己程序的运行逻辑，程序的框架，对于异常情况的处理和恢复方法部分，我们会根据这份报告来给出设计分。

关于决赛服务器

由于资源有限，我们不太可能为每个队伍提供完全独立的4台服务器。一共20支队伍，我们一共提供20台服务器，并且在分配时保证每台服务器都由4支队伍使用，分别作为Trader1，Trader2，Exchange1，Exchange2。以此保证公平。

决赛时提供的服务器为16核，物理内存64G，硬盘大小200G的服务器，一台服务器上有4队，因此我们会对于cpu核内存进行限制，最多使用4个核，实际使用物理内存最多16G，请注意如果使用了过大的内存你的程序会被系统自动kill掉。硬盘使用不做限制，参赛队伍可以随意使用。

【1】深交所规则.pdf

【2】存储大矩阵的方式采用hdf5格式存储，python端比较常用，c++端参考 https://portal.hdfgroup.org/display/HDF5/Learning+HDF5。