thread, timed\_mutex, conditional\_variable, duration, atomic

# Задание: эмуляция системы доставки товаров «производитель-склад-потребитель»

Пояснение:

* **Поставщики** должны «развозить» по складам свои товары, пока не пристроят ВСЕ. У каждого поставщика машин (потоков) ровно столько, сколько товаров. У каждого поставщика есть предпочитаемый им склад (ближайший). Но склад может принимать товар от другого поставщика или выдавать товар потребителю. Поэтому поставщик ждет заданный интервал времени и, если за это время склад не освободился, едет на следующий склад и т.д., пока не пристроит товар.  
  Количество поставщиков и их характеристики задает программист.
* **Складов** может быть несколько (сколько задаст программист). Все склады объединены в одну структуру данных (посредством статического массива или контейнера). Емкость каждого склада ограничена (задает программист). Поэтому если поставщик хочет пристроить свой товар на этот склад, а места нет, то поставщик ждет заданный интервал времени, и если в течение этого времени место на складе не освобождается (потребителем), то едет на другой склад.
* **Потребители** должны забрать ВСЕ товары со всех складов. У каждого потребителя только одна машина (для упрощения задачи). Так как товары на разных складах могут появляться с маленьким промежутком или даже одновременно => потребитель обзванивает склады и пытается забрать товар с первого непустого склада => этот склад считается просто предпочтительным! Но склад не бронирует товар для конкретного потребителя. Потребителей одновременно может быть несколько => "кто первым успел". Если на складе товаров не оказалось, или склад занят приемкой товара от поставщика, или обслуживает другого потребителя, потребитель ждет заданный интервал времени. Если интервал истек, а склад занят обслуживанием других клиентов или пуст, потребитель должен снова обзванивать склады и отправляться на тот склад, где появился товар. Если товаров ни на одном из складов не осталось, все потребители должны завершиться.
* Когда все товары разъехались по потребителям (каждый раз, когда потребитель получает очередной товар, он отправляет уведомление), нужно распечатать информацию по потребителям – кто сколько и каких товаров принял.

Ниже приведено **мое** виденье решения задачи => инакомыслие и творчество приветствуются (а также **обоснование** своего варианта + доброжелательная критика):

## Producer

class Producer{

std::vector<std::string> goods; //это "товары", которые нужно распределить по складам. Количество товаров задается параметром конструктора. Значения формируются таким образом, чтобы в будущем можно было выяснить: от каких поставщиков каждый потребитель получил товар, а возможно, и в каком порядке

size\_t idealStore; //предпочтительный склад («ближайший»). Но, если в течение заданного интервала не удается отгрузить товар на этот склад, пробует пристроить на другой и т.д. по кругу

std::chrono::duration<double> m\_dur; //максимальное время, в течение которого данный поставщик может ждать, пока освободится место на складе. Если время закончилось, а место так и не освободилось, то поставщик должен ехать на следующий склад и снова пытаться пристроить товар.

public:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Параметры конструктора (валидацию параметров делаем самостоятельно):

// prefix - префикс наименования товара (чтобы потом можно было понять - от какого поставщика покупателем получен товар),

// n - сколько товаров поставщик произвел и должен доставить на склады (совпадает с количеством машин поставщика)

// \_idealStore - предпочтительный склад

// d - максимальное время ожидания на складе

producer(const std::string& prefix, size\_t n, size\_t \_idealStore,  
 std::chrono::duration<double> d)

:idealStore(\_idealStore), m\_dur(d)

{

goods.reserve(n);

for (size\_t i=0; i <n; i++)

{

goods.push\_back(prefix + std::to\_string(i));

}

//увеличить ОБЩИЙ счетчик (store::amount) на «n»

//Критична ли в данном случае атомарность?

//Как (за счет чего?) можно повысить эффективность?

//…

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Доставка всех товаров для каждого поставщика запускается в отдельном потоке, чтобы «не тормозить» первичный поток и других поставщиков (а, может быть, и потребителей)

//У поставщика столько машин, сколько товаров => он «отправляет» машины на склад «одновременно» => каждая машина «едет» на склад в своем отдельном потоке

//Замечание 1: защита не нужна, так как поставщик - "неразделяемый".

// Замечание 2: подумайте – нужно ли ждать завершения потоков-доставщиков?

// Замечание 3: в этом потоке (в цикле) отправляем каждый товар (каждую машину) на предпочтительный склад (статический метод store::acquire(<параметры>) ) в отдельном потоке. Данный поток должен завершиться тогда, когда отправит все товары на склады! Подумайте – как запускать эти потоки? Как передавать параметры потоковой функции?

void FromProducerToStore(<параметры> );

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//…

};

Producer

goods

Store

Store

## Store

class store {//склад с ограниченной емкостью!

std::deque<std::string> goods; //это собственно текущий "склад"

const size\_t capacity; //емкость текущего склада - изменяться не может!

std::timed\_mutex m\_mutex; //защита данных на время получения/отгрузки “товаров”

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//а также перечень всех складов

static store stores[];

static std::atomic<size\_t> amount; //общее количество товаров, //которое должны доставить на склад ВСЕ поставщики.

//Так как это значение будет формироваться в конструкторах поставщиков,

//оно будет полностью сформировано на момент "отправки" товаров =>

//его нулевое значение в дальнейшем можно будет использовать

//в качестве признака завершения работы в целом

static std::condition\_variable cv; //посредством этой условной переменной будем ждать завершения процесса + рассылать уведомления

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Доставка на склад. Доставить нужно в любом случае!!! Эта функция инициируется производителем и запускается в отдельном потоке. Начиная с рекомендуемого склада, пытаемся пристроить товар по очереди на каждый склад (вызывая обычный нестатический метод store:: try\_acquire(<параметры>) конкретного склада). Если на данный склад пристроить не удалось, едем на следующий… (и так по кругу)

//

static void acquire(const std::string& s, size\_t idealStore, std::chrono::duration<double> dur);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Пробуем принять конкретно на наш склад (в течение dur). Если удалось, принимаем товар и возвращаем true.

//Замечание 1: Добавлять товары на склад могут пытаться одновременно несколько поставщиков!

//Замечание 2: Забирать товары со склада могут пытаться одновременно несколько потребителей!

bool try\_acquire(const std::string& s, std::chrono::duration<double> dur);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Выдача товара:

//У потребителя одна машина => за один раз потребитель может забрать только один товар с любого склада => для потребителя запускаем только один поток, в котором в цикле (пока на всех складах все не разберут) потребитель пробует забрать себе как можно больше товаров!

// Обзваниваем склады и находим первый, на котором не пусто. Едем на этот склад и пробуем получить "товар"! Если потребитель приехал на склад, а там уже пусто, ждем заданный таймаут. Если так ничего и не появилось, едем на другой склад.   
//Также нужно учесть ситуацию, когда все товары на всех складах закончились! => продолжать искать не имеет смысла…

static bool release(std::string& s, size\_t idealStore, std::chrono::duration<double> dur);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//Выдать товар с нашего склада покупателю. Тоже в течение dur. Если интервал ожидания закончился, а мы не смогли выдать товар (склад обслуживает другого потребителя или поставщика) или на складе пусто, возвращаем false => потребитель отправится на другой склад

Параметры:

s - адрес - куда положить товар

dur - интервал ожидания

bool try\_release(std::string& s, std::chrono::duration<double> dur) ;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//И возможно понадобится функция определения первого непустого склада

static int notEmpty(); //возвращает или индекс склада, или -1

//Увеличения статического атомарного счетчика

static void IncreaseAmount(int n);

//Ожидание завершения процесса

static void Done();

};

# consumer

Так как товары на разных складах могут появляться с маленьким промежутком или даже одновременно => потребитель обзванивает склады и пытается забрать товар с первого непустого склада => но этот склад считается просто предпочтительным! Так как другой потребитель может «успеть раньше»! Склад не бронирует товар для конкретного потребителя => "кто первым успел"!

class consumer {//будет получать товары со склада

std::vector<std::string> goods; //емкость не ограничена! Чем больше, тем лучше!

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

public:

void FromStoreToConsumer();

void ShowGoods()const;

};

# main()

int main()

{

//Создаем производителей:

std::vector<producer> vP = { {"1", 10, 0,std::chrono::duration<double>(1)} ,

{"2", 20, 1,std::chrono::duration<double>(2)},

{"3", 30, 2, std::chrono::duration<double>(3)} };

//Создаем потребителей:

std::vector<consumer> vC(10);

//Запускаем поставщиков. Каждый поставщик в свою очередь будет распределять доставку каждого товара на склад в отдельном потоке

for (auto& p : vP)

{

…

}

//Запускаем потребителей

for (auto& c : vC)

{

…

}

//Ждем завершения процесса

//Используем условную переменную

store::Done(); //в статическом методе посредством условной переменной

//ожидаем, когда на всех складах больше не

//останется ни одного товара

//Когда все товары со всех складов будут изъяты, нужно подвести итоги!

//

for (auto& c : vC)

{

c.ShowGoods();

}

}