# АВС, ИДЗ4, Вариант 12

## Матвеева Ольга Романовна, БПИ239

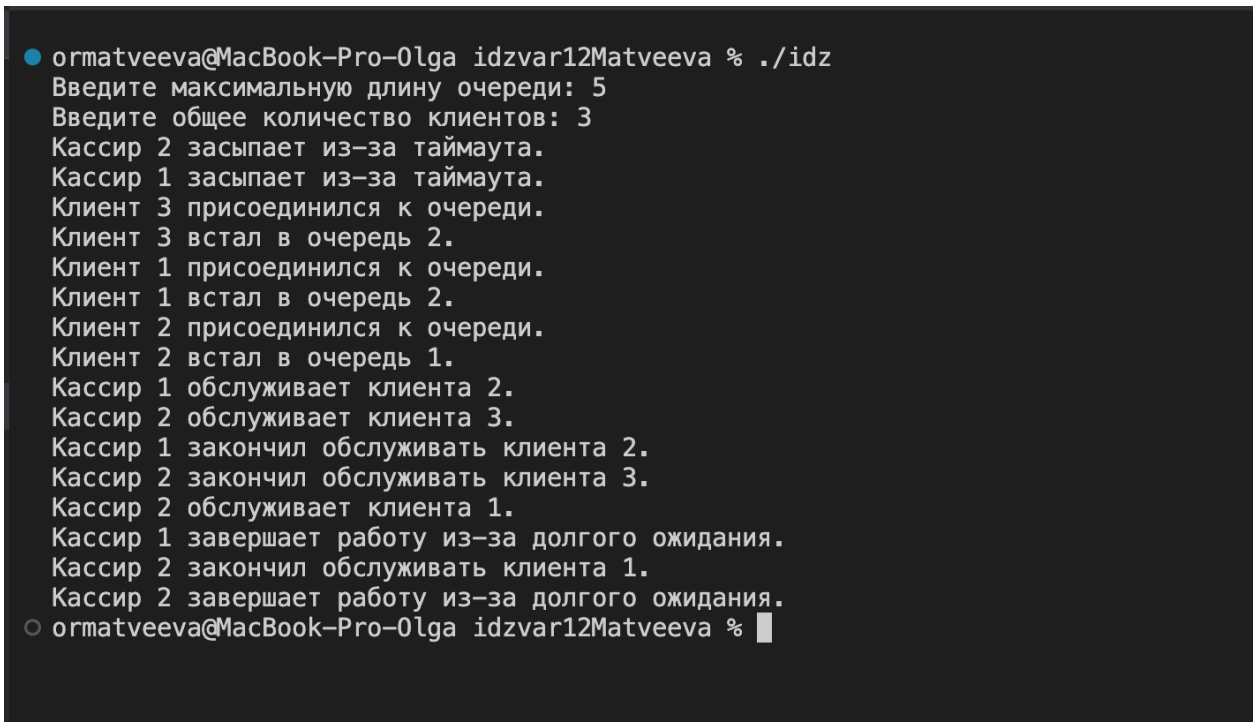
## Отчет

### Задание:

Задача о супермаркете. В супермаркете работают два кассира, покупатели заходят в супермаркет, делают покупки и становятся в очередь к случайному кассиру. Пока очередь пуста, кассир «спит», как только появляется покупатель, кассир «просыпается». Поку- патель ожидает в очереди, пока не подойдет к кассиру. Очереди имеют ограниченную длину N. Если она длиннее, то покупатель не встает ни в одну из очередей и уходит. Если одна из очередей заполнена, то покупатель встает в другую. Создать многопоточное приложение, моделирующее рабочий день супермаркета. Каждый покупатель и кассиры задаются отдельными потоками.

### Примеры вывода программы:

```
ormatveeva@MacBook-Pro-Olga idzvar12Matveeva % g++ -std=c++20 -o idz idz.cpp
ormatveeva@MacBook-Pro-Olga idzvar12Matveeva % ./idz
Введите максимальную длину очереди: 3
Введите общее количество клиентов: 7
Клиент 4 присоединился к очереди.
Клиент 4 встал в очередь 1.
Клиент 2 присоединился к очереди.
Клиент 2 встал в очередь 2.
Кассир 1 обслуживает клиента 4.
Кассир 2 обслуживает клиента 2.
Клиент 7 присоединился к очереди.
Клиент 7 встал в очередь 2.
Кассир Клиент Кассир 31 присоединился к очереди.
Клиент 3 встал в очередь 1.
 закончил обслуживать клиента 4.
2 закончил обслуживать клиента 2.
Кассир 2 обслуживает клиента 7.
Кассир 1 обслуживает клиента 3.
Клиент 5 присоединился к очереди.
Клиент 5 встал в очередь 2.
Клиент 6 присоединился к очереди.
Клиент 6 встал в очередь 2.
Клиент 1 присоединился к очереди.
Клиент 1 встал в очередь 1.
Кассир 1 закончил обслуживать клиента 3.
Кассир 1 обслуживает клиента 1.
Кассир 2 закончил обслуживать клиента 7.
Кассир 2 обслуживает клиента 5.
Кассир Кассир 12 закончил обслуживать клиента 1.
 закончил обслуживать клиента 5.
Кассир 2 обслуживает клиента 6.
Кассир 1 завершает работу из-за долгого ожидания.
Кассир 2 закончил обслуживать клиента 6.
Кассир 2 завершает работу из-за долгого ожидания.
ormatveeva@MacBook-Pro-Olga idzvar12Matveeva %
```




## Логика программы:

1. **Инициализация**:
    1) Программа запрашивает у пользователя максимальную длину очереди и общее количество клиентов.
    2) Создается объект супермаркета с заданными параметрами.
2. **Создание потоков**:
    1) Для каждого кассира создается поток, который запускает метод **serve()**.
    2) Для каждого клиента создается поток, который запускает метод **shop()**.
3. **Обслуживание клиентов**:
    1) Клиенты заходят в супермаркет, делают покупки и выбирают случайную очередь для ожидания.
    2) Если очередь полна, клиент пытается встать в другую очередь. Если обе очереди полны, клиент уходит.
    3) Кассиры "спят", пока нет клиентов. Как только клиент появляется в очереди, кассир "просыпается" и начинает его обслуживать.
4. **Завершение работы**:
    1) Кассиры продолжают обслуживать клиентов, пока есть клиенты в очереди. Если клиентов больше нет, через некоторое время кассиры завершают свою работу.

## Пояснения к используемым классам:

1. **Класс Queue**:
    1. Реализует очередь клиентов с ограниченной длиной.
    2. Методы:
        1) **enqueue(int customerId)**: добавляет клиента в очередь, если есть место. Если очередь полна, клиент не может встать в очередь.
        2) **dequeue()**: удаляет клиента из начала очереди и возвращает его ID. Если очередь пуста, кассир "засыпает" до появления клиентов.
        3) **isEmpty()** и **isFull()**: проверяют состояние очереди.
2. **Класс Cashier**:
    1. Представляет кассира, который обслуживает клиентов из очереди.
    2. Метод **serve()**:
        1) Кассир "спит", пока нет клиентов в очереди.
        2) Когда клиент появляется, кассир обслуживает его, симулируя время обслуживания.
        3) Если кассир долго не получает клиентов, он завершает свою работу.
3. **Класс Customer**:
    1. Представляет клиента, который заходит в супермаркет.
    2. Метод **shop()**:
        1) Клиент симулирует время шопинга.
        2) Выбирает случайную очередь (первую или вторую) для присоединения.
        3) Если выбранная очередь полна, клиент пытается встать в другую очередь. Если обе очереди полны, клиент уходит.
4. **Класс Supermarket**:
    1. Управляет очередями и кассирами.
    2. Метод **run()**:
        1) Создает потоки для кассиров и клиентов.
        2) Кассиры работают в отдельных потоках, обслуживая клиентов из своих очередей.
        3) Клиенты также создаются в отдельных потоках, которые выполняют метод **shop()**.

Код программы с комментариями:

```cpp
#include <iostream>
#include <pthread.h>
#include <queue>
#include <unistd.h>
#include <random>
#include <ctime>

// Класс Очередь (клиентов)
class Queue {
public:
    // Конструктор, инициализирующий максимальный размер очереди
    Queue(int maxSize) : MAX_QUEUE_SIZE(maxSize) {}

    // Метод для добавления клиента в очередь
    bool enqueue(int customerId) {
        pthread_mutex_lock(&mutex);
        if (queue.size() < MAX_QUEUE_SIZE) {
            queue.push(customerId);
            std::cout << "Клиент " << customerId << " присоединился к очереди.\n";
            pthread_cond_signal(&cond); // Будим кассира
            pthread_mutex_unlock(&mutex);
            return true; // Успешно присоединился к очереди
        } else {
            pthread_mutex_unlock(&mutex);
            return false; // Очередь полна
        }
    }

    // Метод для удаления клиента из начала очереди
    int dequeue() {
        pthread_mutex_lock(&mutex);
        while (queue.empty()) {
            struct timespec ts;
            clock_gettime(CLOCK_REALTIME, &ts);
            ts.tv_sec += 0; // 0 секунд
            ts.tv_nsec += 100000000; // 100 миллисекунд
            if (ts.tv_nsec >= 1000000000) {
                ts.tv_sec += 1;
                ts.tv_nsec -= 1000000000;
            }
            int ret = pthread_cond_timedwait(&cond, &mutex, &ts);
            if (ret == ETIMEDOUT) {
                // Произошел таймаут, возвращаем -1, чтобы указать, что клиент не был обслужен
                pthread_mutex_unlock(&mutex);
                return -1; // Указываем, что клиент не был обслужен
            }
        }
        int customerId = queue.front();
        queue.pop();
        pthread_mutex_unlock(&mutex);
        return customerId; // Возвращаем ID обслуженного клиента
```

```
    }

    // Метод для проверки, пуста ли очередь
    bool isEmpty() {
        pthread_mutex_lock(&mutex);
        bool empty = queue.empty();
        pthread_mutex_unlock(&mutex);
        return empty; // Возвращаем true, если очередь пуста
    }

    // Метод для проверки, заполнена ли очередь
    bool isFull() {
        pthread_mutex_lock(&mutex);
        bool full = queue.size() >= MAX_QUEUE_SIZE;
        pthread_mutex_unlock(&mutex);
        return full; // Возвращаем true, если очередь полна
    }

private:
    std::queue<int> queue; // Очередь клиентов
    pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // Мьютекс для защиты очереди
    pthread_cond_t cond = PTHREAD_COND_INITIALIZER; // Условная переменная для пробуждения кассиров
    const int MAX_QUEUE_SIZE; // Максимальный размер очереди
};

// Класс Кассир
class Cashier {
public:
    // Конструктор, инициализирующий ID кассира, очередь и счетчик клиентов
    Cashier(int id, Queue& queue, int& customerCount)
        : id(id), queue(queue), customerCount(customerCount) {}

    // Метод для обслуживания клиентов
    void serve() {
        struct timespec lastServeTime;
        clock_gettime(CLOCK_REALTIME, &lastServeTime);

        while (true) {
            int customerId = queue.dequeue(); // Получаем клиента из очереди
            if (customerId == -1) {
                // Если клиент не был обслужен, проверяем таймаут
                struct timespec currentTime;
                clock_gettime(CLOCK_REALTIME, &currentTime);
                long elapsedTime = (currentTime.tv_sec - lastServeTime.tv_sec) * 1000 +
                                   (currentTime.tv_nsec - lastServeTime.tv_nsec) / 1000000; //
Преобразуем в миллисекунды

                if (elapsedTime > 1000) { // Если ждем более 1000 мс
                    std::cout << "Кассир " << id << " завершает работу из-за долгого ожидания.\n";
                    break; // Выходим из цикла и завершаем работу
                }
                std::cout << "Кассир " << id << " засыпает из-за таймаута.\n";
                sleep(10); // Спим некоторое время перед повторной проверкой
```

```
                continue; // Проверяем клиентов снова
            }

            // Обновляем время последнего обслуживания
            clock_gettime(CLOCK_REALTIME, &lastServeTime);
            std::cout << "Кассир " << id << " обслуживает клиента " << customerId << ".\n";
            sleep(2); // Симулируем время обслуживания
            std::cout << "Кассир " << id << " закончил обслуживать клиента " << customerId << ".\n";
            // Уменьшаем счетчик клиентов
            pthread_mutex_lock(&countMutex);
            customerCount--;
            pthread_mutex_unlock(&countMutex);
            // Выходим, если больше нет клиентов
            if (customerCount <= 0 && queue.isEmpty()) {
                break; // Выходим из цикла, если больше нет клиентов
            }
        }
    }

    static pthread_mutex_t countMutex; // Мьютекс для защиты customerCount

private:
    int id; // ID кассира
    Queue& queue; // Ссылка на очередь
    int& customerCount; // Ссылка на счетчик клиентов
};

// Инициализация статического мьютекса
pthread_mutex_t Cashier::countMutex = PTHREAD_MUTEX_INITIALIZER;

// Класс Клиент
class Customer {
public:
    // Конструктор, инициализирующий ID клиента и ссылки на очереди
    Customer(int id, Queue& queue1, Queue& queue2) : id(id), queue1(queue1), queue2(queue2) {}

    // Метод для симуляции процесса покупок
    void shop() {
        sleep(rand() % 3); // Симулируем время шопинга

        // Генерируем случайное число для выбора очереди (0 или 1)
        int chosenQueue = rand() % 2; // 0 - первая очередь, 1 - вторая очередь

        // Пытаемся присоединиться к выбранной очереди
        if (chosenQueue == 0) {
            if (!queue1.enqueue(id)) { // Если первая очередь полна
                std::cout << "Клиент " << id << " не смог встать в очередь 1, пробует 2.\n";
                if (!queue2.enqueue(id)) { // Пытаемся присоединиться ко второй очереди
                    // Если обе очереди полны, клиент уходит
                    std::cout << "Клиент " << id << " ушел, потому что обе очереди полны.\n";
                }
            } else {
                std::cout << "Клиент " << id << " встал в очередь 1.\n";
```

```
            }
        } else {
            if (!queue2.enqueue(id)) { // Если вторая очередь полна
                std::cout << "Клиент " << id << " не смог встать в очередь 2, пробует 1.\n";
                if (!queue1.enqueue(id)) { // Пытаемся присоединиться к первой очереди
                    // Если обе очереди полны, клиент уходит
                    std::cout << "Клиент " << id << " ушел, потому что обе очереди полны.\n";
                }
            } else {
                std::cout << "Клиент " << id << " встал в очередь 2.\n";
            }
        }
    }

private:
    int id; // ID клиента
    Queue& queue1; // Ссылка на первую очередь
    Queue& queue2; // Ссылка на вторую очередь
};

// Класс Супермаркет
class Supermarket {
public:
    // Конструктор, инициализирующий очереди и счетчик клиентов
    Supermarket(int maxQueueSize, int customerCount)
        : queue1(maxQueueSize), queue2(maxQueueSize), customerCount(customerCount) {
        cashier1 = new Cashier(1, queue1, customerCount);
        cashier2 = new Cashier(2, queue2, customerCount);
    }

    // Метод для запуска симуляции супермаркета
    void run() {
        pthread_t cashierThread1, cashierThread2;
        pthread_create(&cashierThread1, nullptr, [](void* arg) -> void* {
            Cashier* cashier = static_cast<Cashier*>(arg);
            cashier->serve(); // Запускаем обслуживание клиентов кассиром
            return nullptr;
        }, cashier1);

        pthread_create(&cashierThread2, nullptr, [](void* arg) -> void* {
            Cashier* cashier = static_cast<Cashier*>(arg);
            cashier->serve(); // Запускаем обслуживание клиентов кассиром
            return nullptr;
        }, cashier2);

        // Создаем клиентов
        for (int i = 0; i < customerCount; ++i) {
            Customer* customer = new Customer(i + 1, queue1, queue2); // Передаем обе очереди
            pthread_t customerThread;
            pthread_create(&customerThread, nullptr, [](void* arg) -> void* {
                Customer* customer = static_cast<Customer*>(arg);
                customer->shop(); // Запускаем процесс покупок клиента
                delete customer; // Освобождаем память после завершения клиента
```

```
                return nullptr;
            }, customer);
            pthread_detach(customerThread); // Отсоединяем поток
        }

        // Ждем завершения потоков кассиров
        pthread_join(cashierThread1, nullptr);
        pthread_join(cashierThread2, nullptr);
    }

    // Деструктор для освобождения ресурсов
    ~Supermarket() {
        delete cashier1;
        delete cashier2;
    }

private:
    Queue queue1; // Первая очередь
    Queue queue2; // Вторая очередь
    Cashier* cashier1; // Первый кассир
    Cashier* cashier2; // Второй кассир
    int customerCount; // Общее количество клиентов
};

// Главная функция
int main() {
    srand(time(0)); // Инициализируем генератор случайных чисел
    int maxQueueSize; // Переменная для хранения максимальной длины очереди
    int customerCount; // Переменная для хранения общего количества клиентов

    std::cout << "Введите максимальную длину очереди: ";
    std::cin >> maxQueueSize; // Ввод максимальной длины очереди
    std::cout << "Введите общее количество клиентов: ";
    std::cin >> customerCount; // Ввод общего количества клиентов

    Supermarket supermarket(maxQueueSize, customerCount); // Создаем объект супермаркета
    supermarket.run(); // Запускаем симуляцию супермаркета

    return 0; // Завершение программы
}
```