Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет «Высшая школа экономики»

Факультет компьютерных наук

Департамент программной инженерии

Отчет к домашнему заданию №4
«OpenMP – приложение»
По дисциплине
«Архитектура вычислительных систем»

Работу выполнил:

Студент группы БПИ-194 Романюк А.С

Вариант 21

Москва 2020

Задание

Преподаватель проводит экзамен у группы студентов. Каждый студент заранее знает свой билет и готовит по нему ответ. Подготовив ответ, он передает его преподавателю. Преподаватель просматривает ответ и сообщает студенту оценку. Требуется создать многопоточное приложение на основе библиотеки OpenMP, моделирующее действия преподавателя и студентов. При решении использовать парадигму «клиент-сервер».

Модель

Клиенты и серверы – способ взаимодействия неравноправных потоков. Клиентский поток запрашивает сервер и ждет ответа. Серверный поток ожидает запроса от клиента, затем действует в соответствии с поступившим запросом.

Решение

Для реализации данной задачи была использована библиотека “windows.h” для взаимодействия с WINAPI, а именно для работы с событиями, а также библиотека OpenMP для распараллеливания задач и их запуска.

При разработке были использованы критические секции для обеспечения корректного вывода на экран, а также события для синхронизации действий между потоками сервера и клиентом.

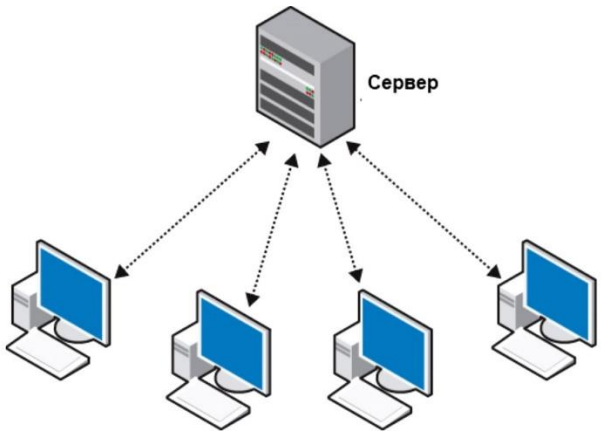

Рисунок 1 – Пример клиент-серверного приложения.

Вывод на каждом ПК может отличаться, всё зависит от количества потоков. Программа будет некорректно работать, если у процессора будет 1 поток. Приложение запускает в первом потоке: поток – сервер, а во всех остальных: потоки – клиенты. При этом, если количество студентов было введено больше, чем количество потоков у

процессора, тогда при освобождении потока будет создаваться новый клиент-сервер до тех пор, пока не создастся ровно n поток, где n – число, введённое пользователем на экране.

Чтобы полностью продемонстрировать работу с потоками была сымитирована следующая ситуация: студенты входят один за одним в аудиторию к преподавателю, причём входят они по ((КОЛ-ВО ПОТОКОВ НА ПРОЦЕССОРЕ) – 1) человек. После того, как один из студентов выйдет из аудитории – зайдёт следующий. То есть запуститься новый клиент-сервер при освобождении потока. Сам же преподаватель готовится принимать, то есть создаётся отдельный поток под преподавателя. После того как студент вошёл в аудиторию, он получает номер билета и начинает готовиться. Причём он может готовиться во время того, как другие студенты входят в аудиторию. Это было сделано для того, чтобы продемонстрировать работу главного потока и потока-студента.

Далее каждый студент смотрит, открыты ли события “serverReady”, “exchange1”, “exchange2”, “serverAnswer”, а также выполняется проверка адресации в пространство процессора данных о студенте. Запросы выполняются с задержкой в *defaultWait* мс. То есть клиент смотрит, готов ли сервер к работе.

Как только сервер стал доступен – студент берёт билет и начинает готовиться случайное время [*minStudentPrep*; *maxStudentPrep*) мс. Дальше студент ждёт готовности преподавателя, если преподаватель готов – он ожидает, пока студент подготовит все данные и выполнит событие “exchange1”. После чего преподаватель принимает у студента экзамен за случайное время [*minTeacherCheck*; *minTeacherCheck*) мс.

Как только преподаватель принял у студента экзамен, он выставляем ему оценку равную случайному целому числу из диапазона [*minMark*; *maxMark*] и выполняет событие “serverAnswer”, и ждёт, пока студент-клиент выполнит событие “exchange2”, то есть заберёт свой студенческий билет и уйдёт. После чего студент подчищает за собой все данные и выходит из аудитории, а сервер уже готов принимать следующего студента.

Всё это происходит до тех пор, пока все студенты не будет опрошены, после чего последует сообщение о том, что экзамен окончен и программа завершится.

Стоит также упомянуть работу с критической секцией.

```
void showMessage(string msg) {
#pragma omp critical
    cout << msg << endl;
      }
```

Каждый вывод на экран в потоках сопровождается работой с критической секцией для корректного отображения данных.

Код программы

```
#include <functional>
```

```cpp
#include <iostream>
#include <windows.h>
#include <vector>
#include "omp.h"
#include <chrono>

/**
 * Задача: Задача про экзамен. Преподаватель проводит экзамен у группы
 *         студентов. Каждый студент заранее знает свой билет и готовит по нему
 *         ответ. Подготовив ответ, он передает его преподавателю. Преподаватель
 *         просматривает ответ и сообщает студенту оценку. Требуется создать
 *         многопоточное приложение, моделирующее действия преподавателя и
 *         студентов. При решении использовать парадигму «клиент-сервер».
 *
 *  Модель: Клиенты и серверы – способ взаимодействия неравноправных потоков. Клиентский
 *          поток запрашивает сервер и ждет ответа. Серверный поток ожидает запроса от
клиента,
 *          затем действует в соответствии с поступившим запросом.
 *
 * Вариант: 21
 * @author Романюк Андрей, БПИ-194
 */

using namespace std;
static const unsigned int defaultWait = 100; // Задержка между запросами в мс.
static const unsigned int minStudentPrep = 4000; // Минимальное время подготовки студента
к ответу в мс.
static const unsigned int maxStudentPrep = 10000; // Максимальное время подготовки
студента к ответу в мс.
static const unsigned int minTeacherCheck = 3000; // Минимальное время проверки
преподавателем студента в мс.
static const unsigned int maxTeacherCheck = 5000; // Максимальное время проверки
преподавателем студента в мс.
static const unsigned int minMark = 1; // Минимальная оценка.
static const unsigned int maxMark = 10; // Максимальная оценка.
static const unsigned int countTickets = 100; // Количество билетов.
static const unsigned int maxStudents = 273; // Максимальное количество студентов.
Реальные цифры с ФКН ПИ 2 курс :)
int leftStudents;


class request {
public:
    int ticket;
    int studNumber;

    request(int ticket, int studNumber) : ticket(ticket), studNumber(studNumber) {
        //
    }
};

class response {
public:
    int mark;
    int studNumber;

    response(int mark, int studNumber) : mark(mark), studNumber(studNumber) {
        //
    }

};

request *generalRequest = nullptr;
response *generalResponse = nullptr;
```

```
// Вывод сообщения
void showMessage(string msg) {
#pragma omp critical
    cout << msg << endl;
}

// Функция потока студента(клиент)
void studentThread(request *r) {
    char buf[256];

    // События для синхронизации с сервером
    HANDLE serverReady, exchange1, exchange2, serverAnswer;

    // Открытие событий сервера
    while ((serverReady = OpenEvent(EVENT_ALL_ACCESS, FALSE, LPCSTR("serverReady"))) ==
nullptr ||
           (exchange1 = OpenEvent(EVENT_ALL_ACCESS, FALSE, LPCSTR("exchange1"))) ==
nullptr ||
           (exchange2 = OpenEvent(EVENT_ALL_ACCESS, FALSE, LPCSTR("exchange2"))) ==
nullptr ||
           (serverAnswer = OpenEvent(EVENT_ALL_ACCESS, FALSE, LPCSTR("serverAnswer"))) ==
nullptr)
        Sleep(defaultWait);


    wsprintfA(buf, "[Client] Студент №%d получил билет %d и начал готовиться.", r-
>studNumber, r->ticket);
    showMessage(buf);

    // Студент готовится:
    Sleep(minStudentPrep + rand() % (maxStudentPrep - minStudentPrep));

    // Когда готов, ожидает готовности сервера (преподавателя):
    WaitForSingleObject(serverReady, INFINITE);
    // Если сервер готов - садится отвечать:

    wsprintfA(buf, "[Client] Студент №%d с билетом %d садится отвечать.", r->studNumber,
r->ticket);
    showMessage(buf);

    generalRequest = r;

    // Подает событие, что данные готовы:
    SetEvent(exchange1);

    // Ожидает ответа сервера:
    WaitForSingleObject(serverAnswer, INFINITE);

    wsprintfA(buf, "[Client] Студент №%d получил оценку %d.", generalResponse-
>studNumber, generalResponse->mark);
    showMessage(buf);

    //Подает сигнал серверу, что область общей памяти свободна и можно принимать
следующего
    SetEvent(exchange2);

    // Овобождение ресурсов, занятых клиентом
    generalRequest = nullptr;
    delete (r);
    CloseHandle(serverAnswer);
    CloseHandle(exchange1);
    CloseHandle(exchange2);
    CloseHandle(serverReady);
```

```
    wsprintfA(buf, "[Client] Студент №%d выходит из аудитории.", r->studNumber);
    showMessage(buf);
}

// Функция потока преподавателя (сервер)
void teacherThread() {
    char buf[256];
    HANDLE serverReady = CreateEvent(nullptr, FALSE, FALSE, LPCSTR("serverReady"));
    HANDLE exchange1 = CreateEvent(nullptr, FALSE, FALSE, LPCSTR("exchange1"));
    HANDLE exchange2 = CreateEvent(nullptr, FALSE, FALSE, LPCSTR("exchange2"));
    HANDLE serverAnswer = CreateEvent(nullptr, FALSE, FALSE, LPCSTR("serverAnswer"));
    // Пока экзамен не кончился:
    while (leftStudents > 0) {
        // Установить событие что сервер готов
        SetEvent(serverReady);

        // Ожидаем пока кто-то из ожидающих клиентов заполнит общую область памяти
        WaitForSingleObject(exchange1, INFINITE);

        wsprintfA(buf, "[Server] Преподаватель начал принимать билет %d у студента №%d",
generalRequest->ticket,
                  generalRequest->studNumber);
        showMessage(buf);
        // Преподаватель принимает у студента рандомное время
        Sleep(rand() % (maxTeacherCheck - minTeacherCheck) + minTeacherCheck);

        generalResponse = new response(rand() % (maxMark - minMark + 1) + minMark,
generalRequest->studNumber);
        // Выставляет оценку за экзамен
        wsprintfA(buf, "[Server] Преподаватель поставил оценку %d студенту №%d",
generalResponse->mark,
                  generalResponse->studNumber);
        showMessage(buf);

        // Устанавливает событие, что сервер ответил (экзамен сдан):
        SetEvent(serverAnswer);
        // Ожидает, пока клиент не подаст сигнал, что он прочитал переданные данные
        WaitForSingleObject(exchange2, INFINITE);
        --leftStudents;
        delete (generalResponse);
    }

    // Освобождение ресурсов, занятых сервером
    delete (generalRequest);
}

/**
 * Метод для генерации рандомно числа с разным сидом для потоков.
 * Так как два потока с лёгкостью могли получать одинаковые значения, то было принято
решение генерарировать новый сид
 * по их айдишнику и с некой битовой магией.
 * @param lim - Максимальное число, не включая его.
 * @param threadId - айдишник треда.
 * @return
 */
int random(int lim, int threadId) {
    srand((threadId << 5) | 4096);
    return rand() % lim;
}

// Выход:
int exit() {
    cout << "Экзамен окончен. Всем спасибо." << endl;
    delete (generalRequest);
    delete (generalResponse);
```

```
    system("pause");
    exit(0);
}

int main() {
    setlocale(LC_ALL, "Russian");
    int n;
    cout << "Введите количество студентов: ";
    cin >> n;

    while (n <= 0 || n >= maxStudents) {
        if (n <= 0)
            cout << "Правильно. Какие очные экзамены во время пандемии?" << endl;
        else if (n >= maxStudents)
            cout << "У нас на програмной инженерии точно не больше " << maxStudents << "
человек. Видимо, Вы ошиблись."
                 << endl;

        cout << "Введите повторно, пожалуйста: ";
        cin >> n;
    };

    leftStudents = n;
    cout << "Студенты начинают входить в аудиторию и готовиться." << endl;
#pragma omp parallel
    {
        if (omp_get_thread_num() == 0) {
            teacherThread();
            exit();
        }
#pragma omp for
        for (int i = 0; i < n + 1; ++i) {
            studentThread(new request(random(countTickets, i), i));
        }
    }
}
```

Тестирование

```
Введите количество студентов:4
 Студенты начинают входить в аудиторию и готовиться.
[Client] Студент №2 получил билет 23 и начал готовиться.
[Client] Студент №4 получил билет 32 и начал готовиться.
[Client] Студент №3 получил билет 27 и начал готовиться.
[Client] Студент №1 получил билет 18 и начал готовиться.
[Client] Студент №4 с билетом 32 садится отвечать.
[Server] Преподаватель начал принимать билет 32 у студента №4
[Server] Преподаватель поставил оценку 8 студенту №4
[Client] Студент №4 получил оценку 8.
[Client] Студент №4 выходит из аудитории.
[Client] Студент №2 с билетом 23 садится отвечать.
[Server] Преподаватель начал принимать билет 23 у студента №2
[Server] Преподаватель поставил оценку 1 студенту №2
[Client] Студент №2 получил оценку 1.
[Client] Студент №2 выходит из аудитории.
[Client] Студент №3 с билетом 27 садится отвечать.
[Server] Преподаватель начал принимать билет 27 у студента №3
[Server] Преподаватель поставил оценку 5 студенту №3
[Client] Студент №3 получил оценку 5.
[Client] Студент №3 выходит из аудитории.
[Client] Студент №1 с билетом 18 садится отвечать.
[Server] Преподаватель начал принимать билет 18 у студента №1
[Server] Преподаватель поставил оценку 9 студенту №1
[Client] Студент №1 получил оценку 9.
[Client] Студент №1 выходит из аудитории.
Экзамен окончен. Всем спасибо.
```

Рисунок 2 – Результат создания потоков-клиентов и выполнения программы

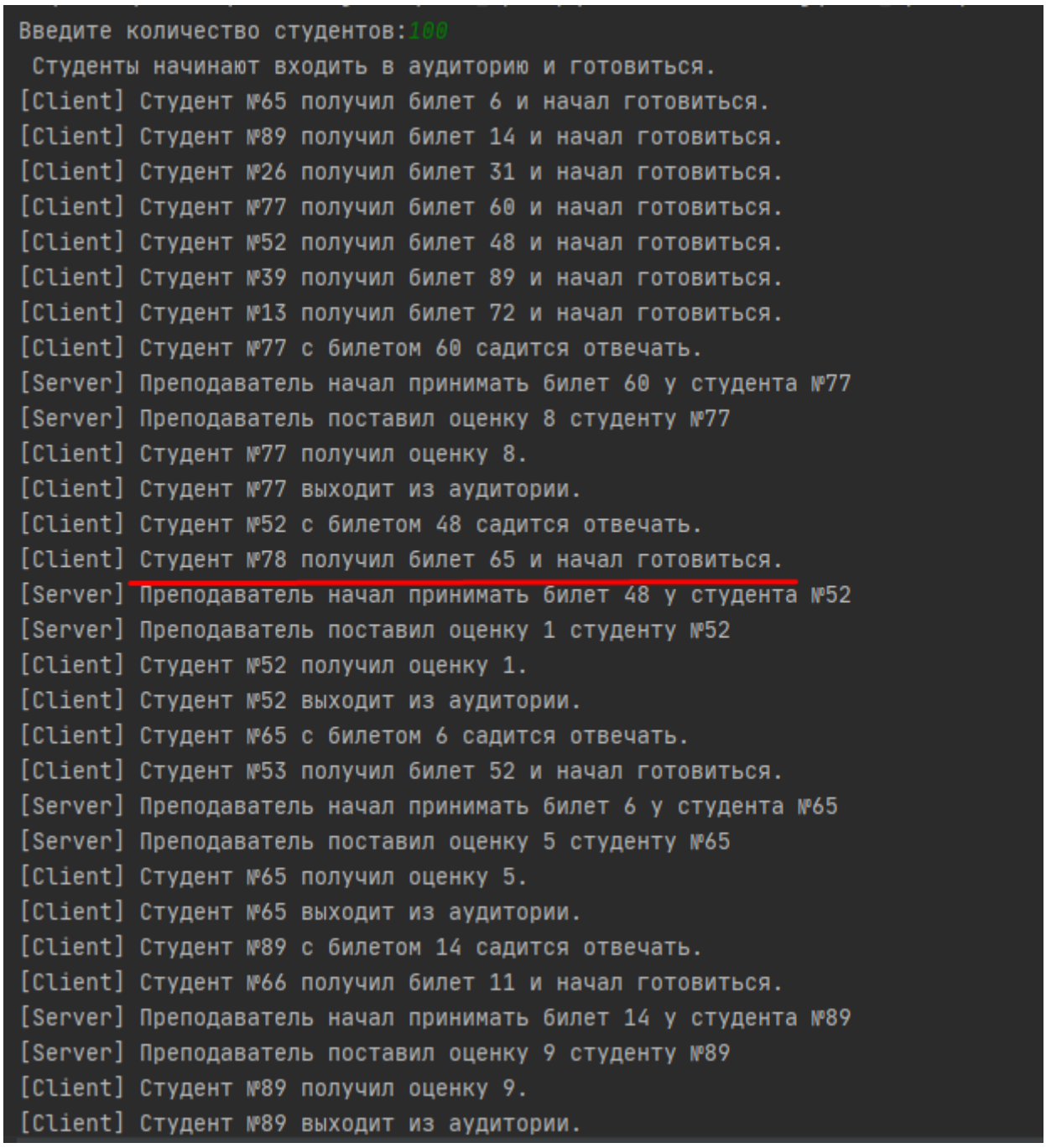

Рисунок 3 – Студент вошёл после того, как аудитория освободилась для него. То есть один из потоков закончил свою работу.

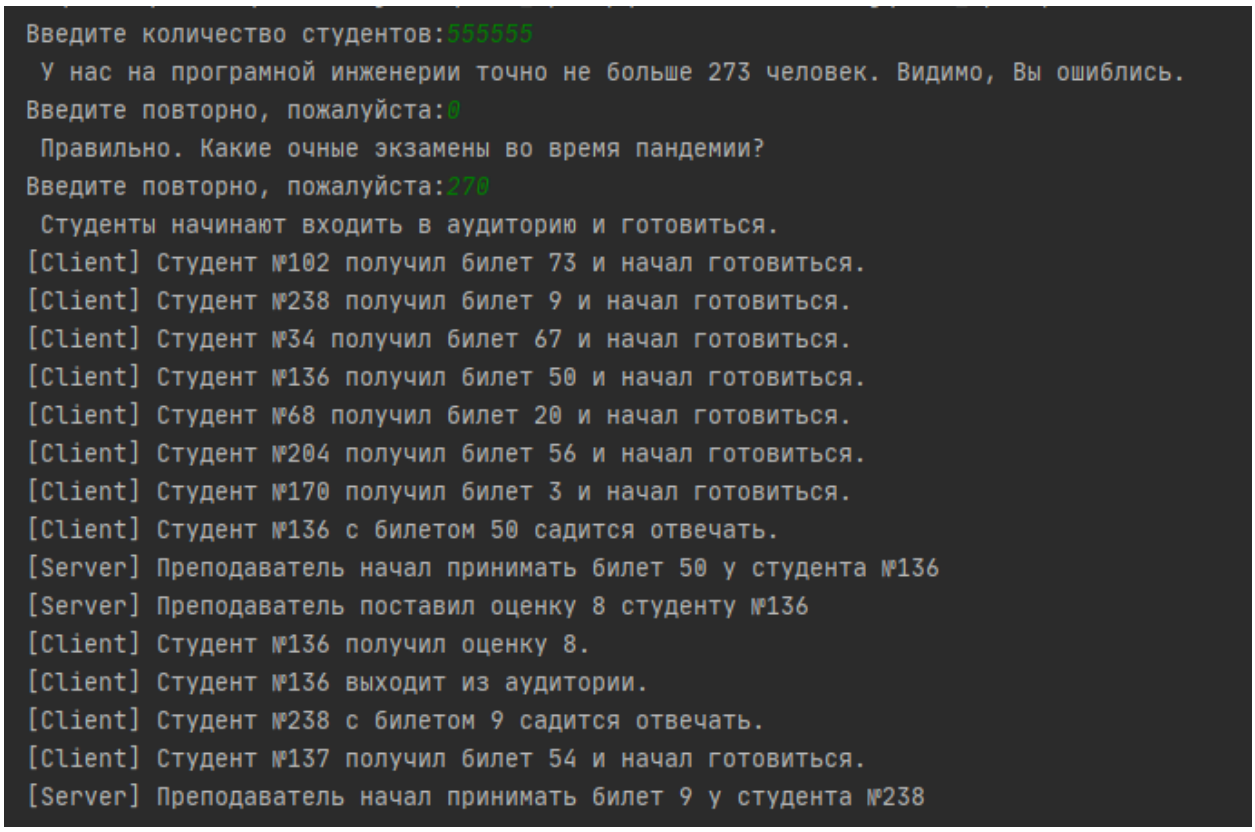

Рисунок 4 – Результат проверки на некорректный ввод.

Список используемых источников

1. Википедия (2020) «Клиент-сервер» (https://ru.wikipedia.org/wiki/Клиент_—_сервер). Просмотрено: 10.11.2020
2. Habr (2020) «Клиент-сервер шаг — за — шагом, от однопоточного до многопоточного (Client-Server step by step)» (https://habr.com/ru/post/330676/). Просмотрено: 10.11.2020
3. Metanit (2020) «Многопоточное клиент-серверное приложение TCP» (https://metanit.com/sharp/net/4.3.php). Просмотрено: 10.11.2020
4. Cyberforum (2020) «Простой клиент-сервер с многопоточностью» (https://www.cyberforum.ru/java-networks/thread1557122.html). Просмотрено: 10.11.2020
5. Docs Microsoft (2020) «Creating Threads» (https://docs.microsoft.com/en-us/windows/win32/procthread/creating-threads). Просмотрено: 10.11.2020
6. Shalatov Grost17 «Потоки (threads) в WinAPI» (http://shatalov.ghost17.ru/winapi/threads.html). Просмотрено: 10.11.2020
7. Легалов А.И.(2020) «Многопоточность. Простая многопоточная программа. Основные функции» (http://softcraft.ru/edu/comparch/practice/thread/01-simple/). Просмотрено: 10.11.2020
8. Легалов А.И.(2020) «Многопоточность. Синхронизация потоков. Методы синхронизации» (http://softcraft.ru/edu/comparch/practice/thread/02-sync/). Просмотрено: 10.11.2020
9. Легалов А.И.(2020) «Многопоточное программирование. OpenMP» (http://www.softcraft.ru/edu/comparch/practice/thread/03-openmp/) Просмотрено 25.11.2020
10. Pro-Prof.ru (2020) «Учебник по OpenMP» (https://pro-prof.com/archives/4335) Просмотрено 25.11.2020