Министерство науки и высшего образования

Российской Федерации

**ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)**

**Факультет дистанционного образования (ФДО)**

Кафедра автоматизированных систем управления (АСУ)

Контрольная работа № 1 по дисциплине

«ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ ПРОЦЕССЫ»

Вариант – 6

Выполнил: студент

специальности 09.03.01

группа з-435П11-1

Бубенщиков Олег Юрьевич

23.05.2020

Проверил: доцент каф. АСУ ТУСУР

Романенко Владимир Васильевич

«\_\_\_»\_\_\_\_\_\_\_\_2020г.

г. Томск 2020

Оглавление

[1. ЗАДАНИЕ 1 3](#_Toc41157789)

[2. ЗАДАНИЕ 2 4](#_Toc41157790)

[2.1 Требования к среде выполнения 4](#_Toc41157791)

[2.2 Входные данные 4](#_Toc41157792)

[2.2.1 Описание файлов проекта 4](#_Toc41157793)

[2.2.2 Описание структуры входного файла 4](#_Toc41157794)

[2.2.3 Формат структуры «Студент» 5](#_Toc41157795)

[2.3 Выходные данные 5](#_Toc41157796)

[2 4. Компиляция 5](#_Toc41157797)

[2.5 Запуск и аргументы командной строки 5](#_Toc41157798)

[2.6 Описание способов реализации 6](#_Toc41157799)

[2.6.1 Способ №1 6](#_Toc41157800)

[2.6.2 Способ №2 6](#_Toc41157801)

[2.6.3 Способ №3 6](#_Toc41157802)

[2.6.4 Способ №4 6](#_Toc41157803)

[2.7 Тестирование и результаты работы 7](#_Toc41157804)

[2.8 Выводы 8](#_Toc41157805)

[2.9 Список используемых источников 8](#_Toc41157806)

[2.10 Листинг программы 8](#_Toc41157807)

# 1. ЗАДАНИЕ 1

Тема задания: «Реализация алгоритмов планирования использования процессорного времени».

Цель: освоить реализацию алгоритмов планирования использования ресурсов с вытесняющей и невытесняющей многозадачностью, с абсолютным и относительным приоритетом. Освоить реализацию механизмов безопасности и синхронизации потоков, а также механизмов исключения тупиковых ситуаций.

Вариант 6. Ресурс – оборудование (станки) на заводе. Атрибуты – наименование оборудования (станка), а также количество изделий (деталей) P (P>=1), которое оно может обрабатывать одновременно. Количество станков – S (S>=1). Атрибуты деталей – наименование, количество, а также список оборудования (причём заданный в требуемом порядке обработки)

Алгоритмы планирования:

1. LCFS, nonpreemptive;
2. MLQ, абсолютный приоритет.

Для блокировки доступа к оборудованию (станкам) использовать сеть Петри.

# 2. ЗАДАНИЕ 2

Тема задания: «Реализация многопоточной обработки данных».

Цель: освоить реализацию многопоточной обработки данных, а также пула потоков и механизма асинхронного ввода/вывода.

В работе необходимо реализовать многопоточную обработку массива структур данных (из N элементов) четырьмя способами:

1. При помощи массива из M потоков (M <= N), используя для синхронизации объект ядра – семафор.
2. При помощи массива из M потоков (M <=N), используя для синхронизации сеть Петри, моделирующую семафор.
3. При помощи пула из M потоков (M <=N), используя системный пул потоков или асинхронные потоки ввода/вывода.
4. При помощи пула из M потоков (M <=N), моделируя его при помощи сети Петри.

Вариант 6. Структура содержит анкетные данные студентов (ФИО, группа, дата рождения, номер комнаты в общежитии). Требуется вывести в выходной файл данные о студентах, которые родились в заданном месяце M.

## 2.1 Требования к среде выполнения

1. Язык программирования: C++
2. Компилятор g++ с поддержкой стандарта C++11.
3. Операционная система: ОС Linux.
4. Системные характеристики ОС на которой выполнялось задание:
5. Linux: Ubuntu 18.04 x86\_64 GNU/Linux
6. RAM: 8Gb
7. CPU: 4

## 2.2 Входные данные

## 2.2.1 Описание файлов проекта

* task2.cpp Исходный файл программы.
* sem.key Файл-ключ для межпроцессорной коммуникации в ядре системы.
* config2.txt Содержит атрибуты управления работой программы, а также пример структуры «Студент» со всеми заполненными полями.
* config2-1.txt Содержит пример списка студентов с динамической генерацией атрибутов структуры «Студент».

## 2.2.2 Описание структуры входного файла

PA 1 Выбранный способ обработки массива.

M 8 Количество параллельных потоков (или 0).

MONTH 3 Заданный месяц.

PT 1000 Пауза после обработки каждого элемента массива, мс.

POLL\_SIZE 8 Размер пула потоков.

N 8 Размер массива элементов.

student1,c-171,1988,3,8,202 Структура «Студент»

student2,c-172,1975,3,29,136

student4,c-178,1989,3,25,47

student3,c-172,1987,3,5,46

student5,c-111,1989,3,30,48

student6,c-123,1990,3,2,246

student7,c-134,1995,3,17,88

student8,c-45,1964,3,2,87

## 2.2.3 Формат структуры «Студент»

Пример формата структуры «Студент»:

struct Student{

public:

std::string fio; // ФИО

std::string group; // группа

int year; // год рождения, может инициализироваться динамически

int month; // месяц рождения, может инициализироваться динамически

int day; // день рождения, может инициализироваться динамически

int room\_number; // номер комнаты в общежитии, может инициализироваться динамически

Атрибуты структуры ФИО(fio) и Группа(group) не имеют динамической генерации и обязательны для ввода, например: student8,c-45,,,,

## 2.3 Выходные данные

Результаты работы выводятся на консоль.

Формат выходных данных:

* TP Время, требующееся на параллельную обработку массива данных выбранным способом мс.
* TL Время, требующееся на линейную обработку массива данных выбранным способом мс.
* Идентификатор потока: атрибуты структуры «Студент»

## 2 4. Компиляция

Для компилирования выполнить команду:

*$ g++ -pthread -o task2 task2.cpp*

Компиляция с явным указанием стандарта C++11:

*$ g++ -pthread -std=c++11 -o task2 task2.cpp*

## 2.5 Запуск и аргументы командной строки

Исполнение программы выполняется запуском исполняемого файла с одним аргументом – файлом конфигурации:

*$ ./task2 config2.txt*

## 2.6 Описание способов реализации

## 2.6.1 Способ №1

При помощи массива из M потоков (M <= N), используя для синхронизации объект ядра – семафор.

Объект «семафор» реализуется с помощью механизмов ядра Linux с использованием протокола межпроцессорной коммуникации стандарта SystemV IPC.

Реализуемый объект «семафор» является массивом семафоров. Размер массива определяется размером обрабатываемого массива студентов, т.е. для доступа к каждой структуре «студент», в массиве «семафор» определяется свой семафор.

Все потоки, выполняющие обработку структуры «студент», пытаются получить конкурентный доступ к каждому элементу массива студентов через занятие семафора, определённого каждому элементу массива студентов. Для попытки занятия семафора используется *неблокирующий* доступ.

## 2.6.2 Способ №2

При помощи массива из M потоков (M <=N), используя для синхронизации сеть Петри, моделирующую семафор.

Семафор реализуется классом, где в качестве средства синхронизации используется мьютекс.

Для реализации конкурентного доступа к массиву элементов «Студен» создается массив семафоров.

Все потоки, выполняющие обработку структуры «студент», пытаются получить конкурентный доступ к каждому элементу массива студентов через занятие семафора, определённого каждому элементу массива студентов. Для попытки занятия семафора используется *неблокирующий* доступ.

## 2.6.3 Способ №3

При помощи пула из M потоков (M <=N), используя системный пул потоков или асинхронные потоки ввода/вывода.

Для имитации работы пула потоков, массив элементов (N) преобразуется в потокобезопасную очередь.

Управление пулом потоков реализуется использованием объекта ядра – семафора с использованием протокола межпроцессорной коммуникации стандарта SystemV IPC.

Количество одновременно работающих потоков в пуле выставляется в файле конфигурации атрибутом «POLL\_SIZE». Данное значение инициализирует счётчик семафора.

## 2.6.4 Способ №4

При помощи пула из M потоков (M <=N), моделируя его при помощи сети Петри.

Для имитации работы пула потоков, массив элементов (N) преобразуется в потокобезопасную очередь.

Размер пула потоков определяется параметром M. Каждым потоком используется неблокируемый доступ к очереди.

## 2.7 Тестирование и результаты работы

Для тестирования использовались два входных файла:

1. config2.txt – статический способ задания атрибутов структуры «Студент».
2. config2-1.txt – динамический способ задания атрибутов структуры «Студент».

Результаты прогона со статическим способом задания атрибутов сведены в таблицу.

Таблица 1.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| PA | M | MONTH | PT мс. | POLL\_SIZE | N | TP мс. | TL мс. |
| 1 | 8 | 3 | 1000 | 8 | 8 | 1001 | 8001 |
| 2 | 8 | 3 | 1000 | 8 | 8 | 1003 | 8001 |
| 3 | 8 | 3 | 1000 | 8 | 8 | 1003 | 8001 |
| 4 | 8 | 3 | 1000 | 8 | 8 | 1001 | 8001 |

Пример вывода для случая № 1 со статическим заданием всех атрибутов студента:

*$ ./task2 config2.txt*

*Set to hardware concurrency: 8*

*Run parallel processing...*

*Run case 1...*

*140055550433024: student1, c-171, 1988, 3, 8, 202*

*140055542040320: student2, c-172, 1975, 3, 29, 136*

*140055525254912: student4, c-178, 1989, 3, 25, 47*

*140055516862208: student3, c-172, 1987, 3, 5, 46*

*140055508469504: student5, c-111, 1989, 3, 30, 48*

*140055500076800: student6, c-123, 1990, 3, 2, 246*

*140055491684096: student7, c-134, 1995, 3, 17, 88*

*140055533647616: student8, c-45, 1964, 3, 2, 87*

*TP: 1003 msec.*

*Run serial processing...*

*140055568557888: student1, c-171, 1988, 3, 8, 202*

*140055568557888: student2, c-172, 1975, 3, 29, 136*

*140055568557888: student4, c-178, 1989, 3, 25, 47*

*140055568557888: student3, c-172, 1987, 3, 5, 46*

*140055568557888: student5, c-111, 1989, 3, 30, 48*

*140055568557888: student6, c-123, 1990, 3, 2, 246*

*140055568557888: student7, c-134, 1995, 3, 17, 88*

*140055568557888: student8, c-45, 1964, 3, 2, 87*

*TL: 8001 msec.*

Пример вывода для случая № 1 с динамическим заданием всех атрибутов студента во входном файле config2-1.txt:

*$ ./task2 config2-1.txt*

*Set to hardware concurrency: 8*

*Run parallel processing...*

*Run case 1...*

*140392799696640: student2, c-172, 1960, 3, 22, 113*

*140392791303936: student4, c-178, 1967, 3, 3, 168*

*TP: 1000 msec.*

*Run serial processing...*

*140392834606912: student2, c-172, 1960, 3, 22, 113*

*140392834606912: student4, c-178, 1967, 3, 3, 168*

*TL: 8001 msec.*

## 2.8 Выводы

В результате выполнения задания №2, я получил навыки по обработке данных с использованием многопоточной модели. Использование объекта «Семафор» позволило изучить применение конкурентного доступа потоков к объектам с использованием элементов синхронизации и работы с общей памятью.

Полученные результаты, позволяют сделать выводы о преимуществах распараллеливания обработки данных для эффективного использования ресурсов нескольких процессов и экономии общего времени выполнения.

## 2.9 Список используемых источников

1. Калайда В. Т. Параллельные вычислительные процессы : учеб. пособие. – Томск : ФДО, ТУСУР, 2018.

2. С.Б. Арыков, М.А. Городничев, Г.А. Щукин Параллельное программирование над общей памятью POSIX THREADS: учеб. Пособие. – Новосибирск: Изд-во НГТУ, 2018 – 87 с.

3. C++. Практика многопоточного программирования. — СПб.: Питер, 2020. — 640 с.:

ил. — (Серия «Для профессионалов»).

## 2.10 Листинг программы

*#include <errno.h>*

*#include <fcntl.h>*

*#include <sys/stat.h>*

*#include <sys/sem.h>*

*#include <sys/types.h>*

*#include <sys/ipc.h>*

*#include <pthread.h>*

*#include <semaphore.h>*

*#include <time.h>*

*#include <string>*

*#include <sstream>*

*#include <iostream>*

*#include <vector>*

*#include <deque>*

*#include <thread>*

*#include <chrono>*

*#include <mutex>*

*#include <atomic>*

*#include <fstream>*

*#define err\_exit\_sysV(str){ perror(str); std::cerr << std::endl; exit(EXIT\_FAILURE);}*

*const std::string F\_KEY = "sem.key"; // Файловый ключ для коммуникации процессов ядра в стандарте SysV IPC Unix*

*int PA; // способ обработки массива*

*int MONTH; // заданный месяц*

*int N; // размер массива элементов*

*int M; // Количество параллельных потоков (если 0, то принимается равным числу процессорных ядер в системе)*

*int PT; //Пауза после обработки каждого элемента массива, мс*

*int POLL\_SIZE; //размер пула потоков*

*std::mutex mut\_cout;*

*// Структура Студент*

*struct Student{*

*public:*

*std::string fio; // ФИО*

*std::string group; // группа*

*int year; // год рождения*

*int month; // месяц рождения*

*int day; // день рождения*

*int room\_number; // номер комнаты в общежитии*

*Student(){};*

*Student(std::string f, std::string g, int y, int m, int d, int r):fio(f), group(g), year(y), month(m), day(d), room\_number(r){};*

*Student(std::string str){*

*std::istringstream iss(str);*

*std::string w;*

*std::vector<std::string> vs;*

*while(!iss.eof())*

*{*

*if(getline(iss,w,','))*

*vs.push\_back(w);*

*}*

*fio = vs[0];*

*group = vs[1];*

*year = std::atoi(vs[2].c\_str());*

*if(!year)*

*year = Random(1960, 2000);*

*month = std::atoi(vs[3].c\_str());*

*if(!month)*

*month = Random(1, 12);*

*day = std::atoi(vs[4].c\_str());*

*if(!day)*

*day = Random(1, 30);*

*if(vs.size() != 6)*

*room\_number = Random(1, 200);*

*else*

*room\_number = std::atoi(vs[5].c\_str());*

*if(!room\_number)*

*room\_number = Random(1, 200);*

*}*

*int Random(int min, int max) {*

*return min + rand() % (max - min);*

*}*

*void display(){*

*std::cout << fio << ", " << group << ", " << year << ", " << month << ", " << day << ", " << room\_number << "\n";*

*}*

*};*

*// парсинг файла с входными данными*

*std::vector<Student> load\_and\_set(std::string filename)*

*{*

*std::vector<Student> arr\_st;*

*std::string str;*

*std::ifstream infile (filename);*

*if(!infile)*

*{*

*std::cout << "File cannot be opened: " << filename << "\n";*

*std::cout << "Error code: " << infile.rdstate() << "\n";*

*exit(-1);*

*}*

*while (!infile.eof())*

*{*

*srand(time(0));*

*infile >> str;*

*if(str == "PA")*

*infile >> PA;*

*else if(str == "M")*

*{*

*infile >> M;*

*if(M == 0)*

*M = std::thread::hardware\_concurrency();*

*std::cout << "Set to hardware concurrency: " << M << "\n";*

*}*

*else if(str == "MONTH")*

*infile >> MONTH;*

*else if(str == "PT")*

*infile >> PT;*

*else if(str == "POLL\_SIZE")*

*infile >> POLL\_SIZE;*

*else if(str == "N")*

*{*

*infile >> N;*

*for(int j=0; j<=N;j++)*

*{*

*getline(infile, str);*

*if(str.size() == 0)*

*continue;*

*arr\_st.push\_back({str});*

*}*

*}*

*}*

*infile.close();*

*return arr\_st;*

*}*

*// Работа с записью студента*

*void work\_with\_student(Student& st, const int& month)*

*{*

*if(st.month == month)*

*{*

*std::lock\_guard<std::mutex> lk(mut\_cout); // авто-блокировка потока вывода*

*std::cout << std::this\_thread::get\_id() << ": "; // вывод информации на консоль*

*st.display();*

*}*

*std::this\_thread::sleep\_for(std::chrono::milliseconds(PT)); // Пауза после обработки каждого элемента массива*

*}*

*// получить семафор ядра SysV IPC*

*int get\_sysv\_sem(int num\_sem, int sem\_op, int flag)*

*{*

*key\_t key = ftok(F\_KEY.c\_str(), 1);*

*if(key == -1)*

*err\_exit\_sysV(("Cannot create key for SysV IPC: " + F\_KEY).c\_str());*

*// создать массив семафоров в ядре*

*int semid = semget(key, num\_sem, IPC\_CREAT|0666);*

*if(semid == -1)*

*err\_exit\_sysV("Cannot create semaphores");*

*struct sembuf mysops[num\_sem];*

*// инициализация семафоров*

*for(short j=0; j<num\_sem;j++)*

*{*

*mysops[j].sem\_num = j;*

*mysops[j].sem\_op = sem\_op;*

*mysops[j].sem\_flg = flag;*

*}*

*int stat\_sem = semop(semid, mysops, num\_sem);*

*if(stat\_sem == -1)*

*{*

*// удаление семафоров*

*semctl(semid, IPC\_RMID, 0); // удаление сегмента общей памяти*

*err\_exit\_sysV("Cannot initialize semaphore");*

*}*

*return semid;*

*}*

*// Семафор ПЕТРИ*

*class MySem{*

*private:*

*int count;*

*mutable std::mutex mut;*

*public:*

*MySem(){count = 1;}*

*MySem(MySem const&)=delete; // конструктор копирования*

*MySem& operator=(MySem const&)=delete; // присваивание объекта*

*int pos() // увеличивает счётчик*

*{*

*std::lock\_guard<std::mutex> lk(mut);*

*count++;*

*return 0;*

*}*

*int trywait() // неблокированное уменьшение счётчика*

*{*

*std::lock\_guard<std::mutex> lk(mut);*

*if(count == 0)*

*return -1;*

*else*

*{*

*count--;*

*return 0;*

*}*

*}*

*int getvalue() // получить значение счётчика*

*{*

*std::lock\_guard<std::mutex> lk(mut);*

*return count;*

*}*

*};*

*// безопасная очередь*

*class MySafeQueue*

*{*

*private:*

*std::deque <Student> arr\_st;*

*mutable std::mutex mut;*

*public:*

*std::atomic\_bool is\_done;*

*MySafeQueue(){}*

*MySafeQueue(const std::vector<Student>& st){*

*for(const auto& it: st)*

*arr\_st.push\_back(it);*

*}*

*bool try\_pop(Student& st) // неблокирующее получение элемента очереди*

*{*

*std::lock\_guard<std::mutex> lk(mut);*

*if(arr\_st.empty())*

*return false;*

*st = arr\_st.front();*

*arr\_st.pop\_front();*

*return true;*

*}*

*void push(Student st)*

*{*

*std::lock\_guard<std::mutex> lk(mut);*

*arr\_st.push\_back(st);*

*}*

*bool empty()*

*{*

*std::lock\_guard<std::mutex> lk(mut);*

*return arr\_st.empty();*

*}*

*};*

*void thread\_job\_var1(std::vector<Student>& arr\_st, int& semid, int month)*

*{*

*// структура управления семафором*

*struct sembuf mybuf;*

*mybuf.sem\_op = -2;*

*mybuf.sem\_flg = IPC\_NOWAIT;*

*for(int j=0; j<N; j++) // Для каждого элемента массива*

*{*

*mybuf.sem\_num = j; // выбрать семафор элемента*

*int stat\_sem = semop(semid, &mybuf, 1); // занять семафор элемента неблокируемым способом*

*if(stat\_sem == -1)*

*continue; // семафор занят, идём дальше*

*else*

*{*

*work\_with\_student(arr\_st[j], month);*

*}*

*}*

*}*

*void thread\_job\_var2(std::vector<Student>& arr\_st, std::vector<MySem>& vsem, int month)*

*{*

*for(int j=0; j<N; j++) // Для каждого элемента массива*

*{*

*int stat\_sem = vsem[j].trywait(); // занять семафор элемента неблокируемым способом*

*if(stat\_sem == -1)*

*continue; // семафор занят, идём дальше*

*else*

*{*

*work\_with\_student(arr\_st[j], month);*

*}*

*}*

*}*

*void thread\_job\_var3(MySafeQueue& myqueue, int& semid, int month)*

*{*

*// структура управления семафором*

*struct sembuf mybuf;*

*mybuf.sem\_num = 0;*

*mybuf.sem\_flg = 0;*

*while(true)*

*{*

*// Проверка на завершение работы потока*

*if(myqueue.empty())*

*return;*

*mybuf.sem\_op = -1;*

*// занять семафор*

*semop(semid, &mybuf, 1);*

*Student st;*

*if(myqueue.try\_pop(st)) // получить объект из очереди*

*{*

*work\_with\_student(st, month);*

*}*

*// освободить семафор*

*mybuf.sem\_op = 1;*

*semop(semid, &mybuf, 1);*

*}*

*}*

*void thread\_job\_var4(MySafeQueue& myqueue, int month)*

*{*

*while(true)*

*{*

*// Проверка на завершение работы потока*

*if(myqueue.empty())*

*return;*

*Student st;*

*if(myqueue.try\_pop(st)) // неблокируемый доступ к очереди*

*{*

*work\_with\_student(st, month);*

*}*

*else*

*{*

*std::this\_thread::yield(); // заснуть*

*}*

*}*

*}*

*// 1. При помощи массива из M потоков (M ≤ N), используя для синхронизации объект ядра – семафор.*

*void run\_var1(std::vector<Student>& arr\_st, int month)*

*{*

*std::cout << "Run case 1...\n";*

*int semid = get\_sysv\_sem(N, 2, IPC\_NOWAIT);*

*std::vector<std::thread> v\_th;*

*for(int j=0; j<M; j++)*

*{*

*v\_th.push\_back(std::thread(thread\_job\_var1, std::ref(arr\_st), std::ref(semid), month));*

*}*

*for(auto& th: v\_th)*

*th.join();*

*// удаление семафоров*

*semctl(semid, IPC\_RMID, 0); // удаление сегмента общей памяти*

*}*

*// 2. При помощи массива из M потоков (M ≤ N), используя для синхронизации сеть Петри, моделирующую семафор.*

*void run\_var2(std::vector<Student>& arr\_st, int month)*

*{*

*std::cout << "Run case 2...\n";*

*std::vector<MySem> vsem(N);*

*std::vector<std::thread> v\_th;*

*for(int j=0; j<M; j++)*

*{*

*v\_th.push\_back(std::thread(thread\_job\_var2, std::ref(arr\_st), std::ref(vsem), month));*

*}*

*for(auto& th: v\_th)*

*th.join();*

*}*

*// 3. При помощи пула из M потоков (M ≤ N), используя системный пул потоков или асинхронные потоки ввода/вывода.*

*void run\_var3(std::vector<Student>& arr\_st, int month)*

*{*

*std::cout << "Run case 3...\n";*

*MySafeQueue myqueue;*

*// заполнение очереди*

*for(const auto stud:arr\_st)*

*myqueue.push(stud);*

*int semid = get\_sysv\_sem(1, POLL\_SIZE, 0); // семафор пула потоков*

*std::vector<std::thread> v\_th;*

*for(int j=0; j<M; j++)*

*{*

*v\_th.push\_back(std::thread(thread\_job\_var3, std::ref(myqueue), std::ref(semid), month));*

*}*

*for(auto& th: v\_th)*

*th.join();*

*// удаление семафоров*

*semctl(semid, IPC\_RMID, 0); // удаление сегмента общей памяти*

*}*

*// 4. При помощи пула из M потоков (M ≤ N), моделируя его при помощи сети Петри.*

*void run\_var4(std::vector<Student>& arr\_st, int month)*

*{*

*std::cout << "Run case 4...\n";*

*MySafeQueue myqueue;*

*// заполнение очереди*

*for(const auto stud:arr\_st)*

*myqueue.push(stud);*

*std::vector<std::thread> v\_th;*

*for(int j=0; j<M; j++)*

*{*

*v\_th.push\_back(std::thread(thread\_job\_var4, std::ref(myqueue), month));*

*}*

*myqueue.is\_done = true;*

*for(auto& th: v\_th)*

*th.join();*

*}*

*// Последовательная обработка массива данных*

*void serial\_processing(std::vector<Student>& arr\_st, int month)*

*{*

*std::cout << "Run serial processing...\n";*

*auto start\_time = std::chrono::steady\_clock::now();*

*for(auto& st: arr\_st)*

*work\_with\_student(st, month);*

*auto stop\_time = std::chrono::steady\_clock::now();*

*std::cout << "TL: " << std::chrono::duration\_cast<std::chrono::milliseconds>(stop\_time - start\_time).count() << " msec." << std::endl;*

*}*

*int main(int argc, char\* argv[])*

*{*

*std::vector<Student> arr\_st;*

*if(argc == 2)*

*arr\_st = load\_and\_set(argv[1]);*

*else*

*{*

*std::cout << "Unknown command-line options...\n";*

*exit(-1);*

*}*

*if(PA > 4 or PA < 1)*

*{*

*std::cout << "Selected the unknown variant PA = " << PA << "\n";*

*std::cout << "Exit\n";*

*exit(-1);*

*}*

*std::cout << "Run parallel processing...\n";*

*auto start\_time = std::chrono::steady\_clock::now();*

*switch (PA)*

*{*

*case 1:{*

*run\_var1(arr\_st, MONTH);*

*break;*

*}*

*case 2:{*

*run\_var2(arr\_st, MONTH);*

*break;*

*}*

*case 3:{*

*run\_var3(arr\_st, MONTH);*

*break;*

*}*

*case 4:{*

*run\_var4(arr\_st, MONTH);*

*break;*

*}*

*}*

*auto stop\_time = std::chrono::steady\_clock::now();*

*std::cout << "TP: " << std::chrono::duration\_cast<std::chrono::milliseconds>(stop\_time - start\_time).count() << " msec." << std::endl;*

*serial\_processing(arr\_st, MONTH);*

*return 0;*

*}*