Министерство науки и высшего образования Российской Федерации
Новосибирский Государственный технический университет
Кафедра автоматизированных систем управления

Отчет по лабораторной работе №3

по дисциплине «Параллельное программирование»

«Реализовать модель с равноправными узлами, представляющие пары потоков, читающие входной файл и вычисляющие результат в соответствие с заданным вариантом на языке C++ с использованием стандарта POSIX.»

Вариант – рпк1 пп

Выполнили
студенты группы АВТ-813:
Кинчаров Данил
Пайхаев Алексей
Чернаков Кирилл
Преподаватель:
Ландовский Владимир Владимирович,
к.т.н., доцент кафедры АСУ

г. Новосибирск
2020 г.

Содержание

1. Постановка задачи.

Программа решает множество независимых однотипных задач. В варианте задается модель распределения работы между потоками и тип решаемых задач. В каждом варианте для простоты предполагается, что условия задач (входные данные) расположены в файле, причем сложность задач может сильно отличаться друг от друга и заранее разделить их на равные по объему вычислений части невозможно. Результаты должны записываться в выходные файлы. Использовать POSIX Threads.

рпк1 - модель с равноправными узлами, представляющими собой пары потоков, первый читает входной файл и вычисляет результаты, второй записывает выходной файл. Файл результатов общий, каждый результат записывается отдельно (каждое обращение к файлу записывает один результат).рм - разложение на простые множители, элемент

пп - проверка на простоту, исходные данные - число, результат - да/нет.

2. Описание алгоритма, структуры данных.

Для каждой пары reader-writer существует общий массив результатов, соответствующий мьютекс для осуществления корректной параллельной работы потоков при чтении элементов массива и записи результатов в общий выходной файл, а также условие для пассивного ожидания результатов в массиве.

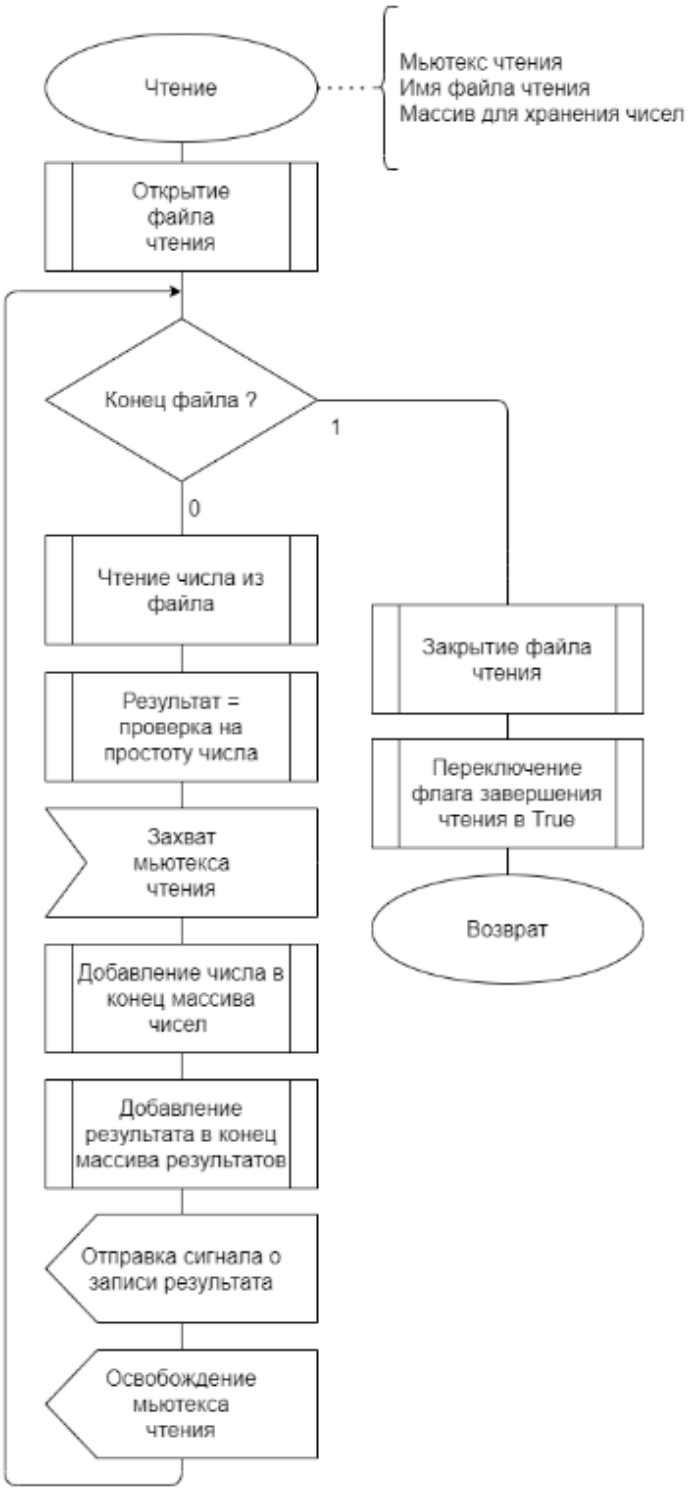
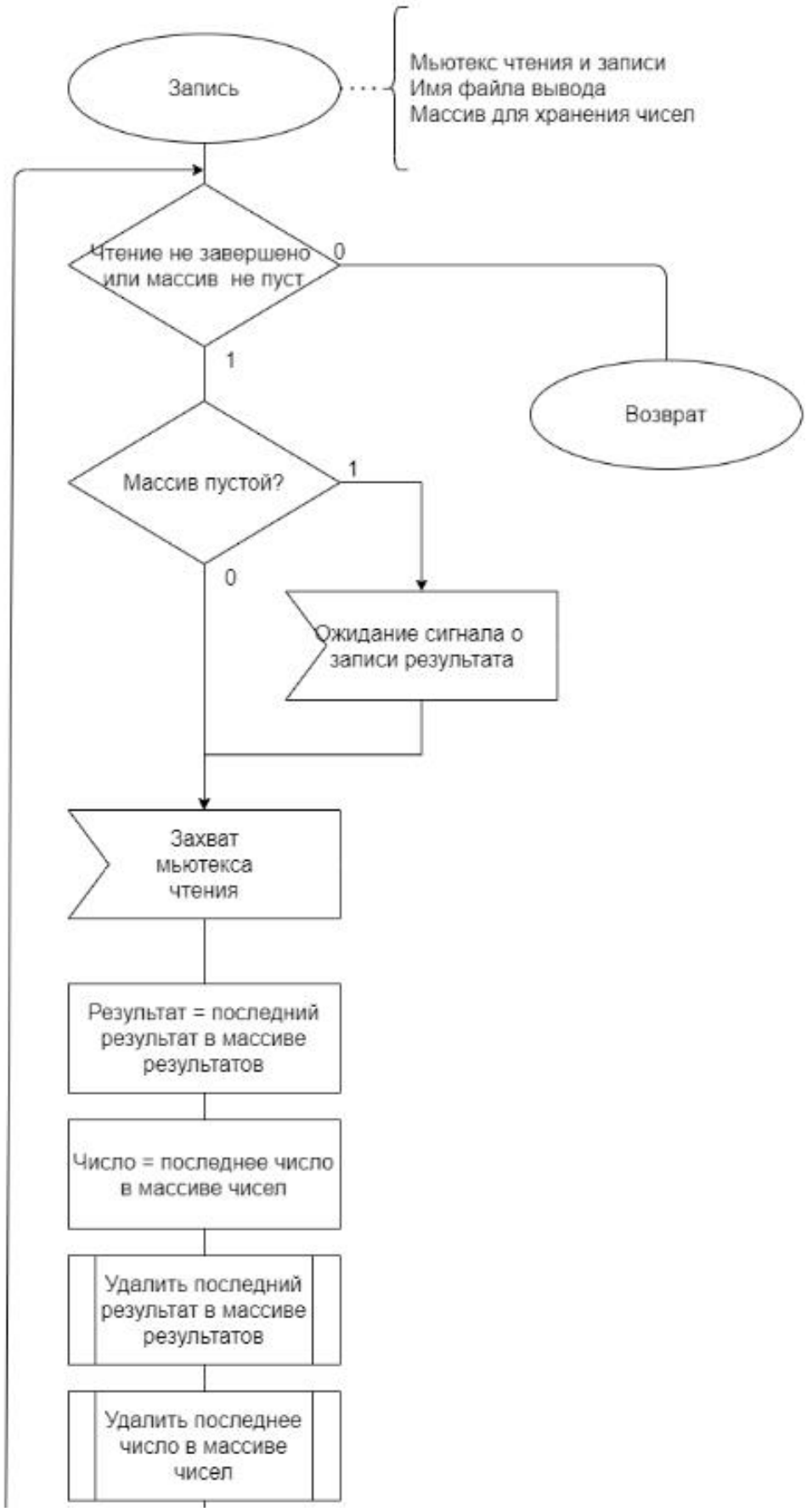


Рисунок 1 – Блок-схема работы потока, который читает входной файл


Запись
Мьютекс чтения и записи
Имя файла вывода
Массив для хранения чисел
Чтение не завершено или массив не пуст
0
1
Возврат
Массив пустой?
1
0
Ожидание сигнала о записи результата
Захват мьютекса чтения
Результат = последний результат в массиве результатов
Число = последнее число в массиве чисел
Удалить последний результат в массиве результатов
Удалить последнее число в массиве чисел

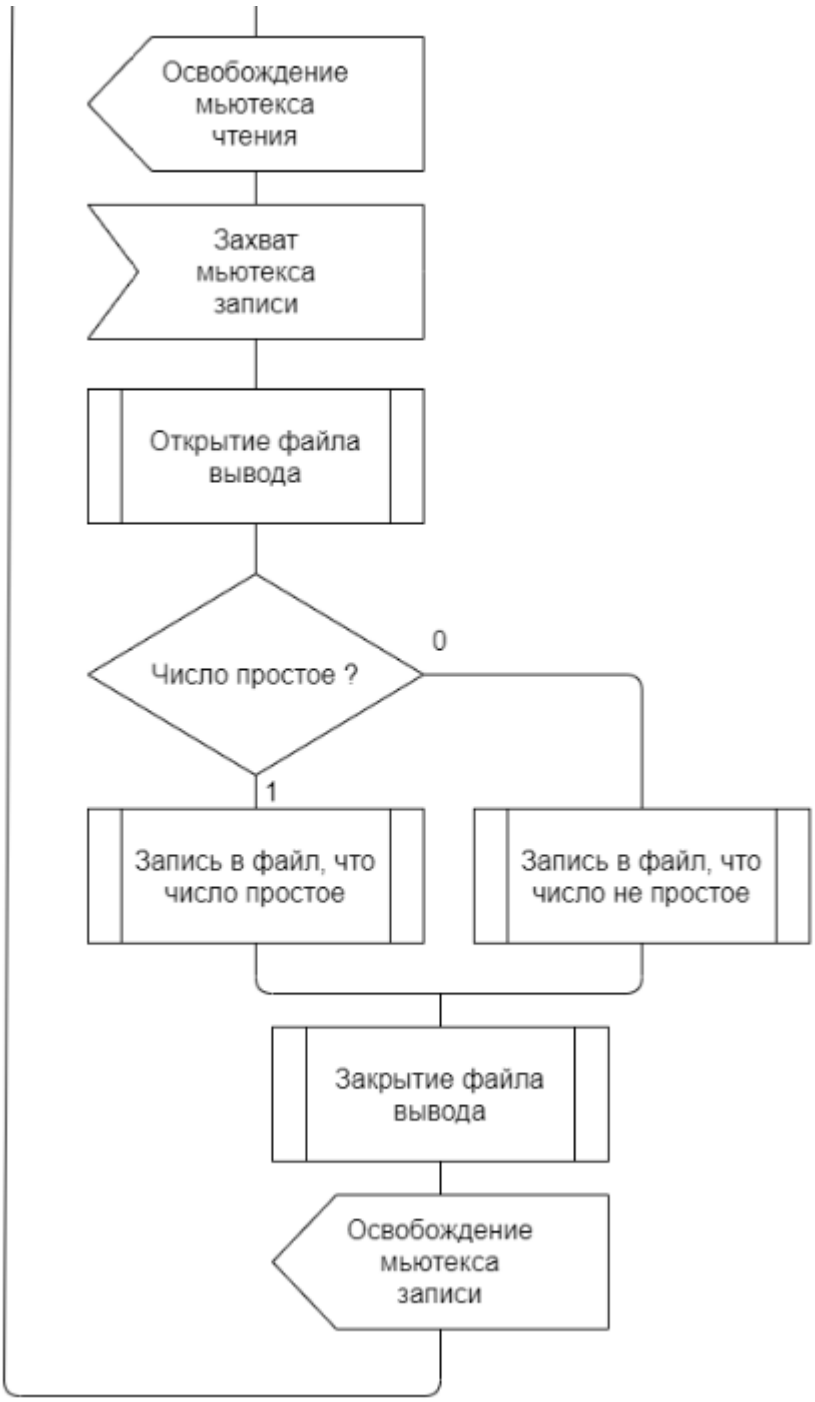


Рисунок 2 – Блок-схема работы потока, который записывает файл вывода

3. Текст программы.

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include "pthread.h"
#include <conio.h>
#include <string>
#include <chrono>
#include <vector>
#include <fstream>
#include <cstring>

using namespace std;
#pragma comment(lib, "pthreadVCE2.lib")
void driver();

bool isPrimeNumber(int);
void processNumbersInFiles(vector<string>);
vector<string> fillInputFiles();
string checkFiles(vector<string>);
ifstream getReadStream(string);
ofstream getWriteStream(string);
void* processInput(void*);
void* processOutput(void*);
void clearFile(string);

struct bufferArg {
	bool isReadingFinish = false;
	vector<bool> results = {};
	vector<int> numbers = {};
	string inputFileName = "";
	string outputFileName = "";
	pthread_mutex_t vectorMutext;
	pthread_cond_t vectorCond;
};

pthread_mutex_t fileMutext;
const int numbersAmount = 200000;
const string outputName = "output.txt";

int main()
{
	do {
		driver();
	} while (_getch() != EOF);

}

void driver() {
	vector<string> files = fillInputFiles();
	auto start = std::chrono::system_clock::now();
	processNumbersInFiles(files);
	auto end = std::chrono::system_clock::now();
	cout << "Done\nTime: " <<
std::chrono::duration_cast<std::chrono::milliseconds>(end - start).count() << endl;
}

void processNumbersInFiles(vector<string> files) {
	string checkResult = checkFiles(files);
	vector<pthread_t*> outputThreads = {};
	vector<pthread_t*> inputThreads = {};

	pthread_mutex_init(&fileMutext, NULL);
	clearFile(outputName);
```

```
	if (checkResult == "") {
		for (int i = 0; i < files.size();i++) {

			struct bufferArg* args = new bufferArg();
			args->inputFileName = files[i];
			args->outputFileName = outputName;
			pthread_mutex_init(&args->vectorMutext, NULL);
			pthread_cond_init(&args->vectorCond, NULL);

			pthread_t inputThread = pthread_t();
			pthread_t outputThread = pthread_t();
			outputThreads.push_back(&outputThread);
			inputThreads.push_back(&inputThread);

			pthread_create(&inputThread, NULL, processInput, (void*)args);
			pthread_create(&outputThread, NULL, processOutput, (void*)args);
		}

		for (int i = 0; i < outputThreads.size(); i++) {
			pthread_join(*(inputThreads[i]), NULL);
			pthread_join(*(outputThreads[i]), NULL);
		}
	}
	else
	{
		cout << "Error when open file: " << checkResult << endl;
	}
}

ifstream getReadStream(string inputFileName) {
	ifstream fileInput(inputFileName);
	return fileInput;
}

ofstream getWriteStream(string outputFileName) {
	ofstream fileOutput(outputFileName, ios_base::app);
	return fileOutput;
}

void* processInput(void* arguments) {
	struct bufferArg* args = (struct bufferArg*)arguments;
	int a = 0;
	ifstream input;

	while (!input.is_open()) {
		input = getReadStream(args->inputFileName);
	}

	while (input >> a) {
		bool result = isPrimeNumber(a);
		pthread_mutex_lock(&args->vectorMutext);
		args->numbers.push_back(a);
		args->results.push_back(result);
		pthread_cond_signal(&args->vectorCond);
		pthread_mutex_unlock(&args->vectorMutext);
	}
	args->isReadingFinish = true;

	return NULL;
}

void* processOutput(void* arguments) {
	struct bufferArg* args = (struct bufferArg*)arguments;
	ofstream output;
```

```
	while (!args->isReadingFinish || !args->results.empty()) {

		pthread_mutex_lock(&args->vectorMutext);

		if (args->results.empty()) {
			pthread_cond_wait(&args->vectorCond, &args->vectorMutext);
		}

		bool result = args->results.back();
		int number = args->numbers.back();
		args->numbers.pop_back();
		args->results.pop_back();
		pthread_mutex_unlock(&args->vectorMutext);

		pthread_mutex_lock(&fileMutext);
		while (!output.is_open()) {
			output = getWriteStream(args->outputFileName);
		}

		if (result) {
			output << "Number " << number << " is prime\n";
		}
		else {
			output << "Number " << number << " is not prime\n";
		}
		output.close();
		pthread_mutex_unlock(&fileMutext);
	}

	return NULL;
}

string checkFiles(vector<string> files) {
	if (files.size() <= 0) {
		return "No files";
	}

	for (int i = 0; i < files.size(); i++) {
		ifstream finput(files[i]);
		if (!finput.is_open()) {
			return files[i];
		}
	}
	return "";
}

vector<string> fillInputFiles() {
	vector<string> files = {};
	int filesNumber = 0;
	cout << "Enter number of input files or -1 to continue ( 1 file will use 2 threads
)" << endl;
	cin >> filesNumber;
	for (int i = 0; i < filesNumber; i++) {
		string fileName = "input" + to_string(i) + ".txt";
		files.push_back(fileName);
		ofstream writeStream(fileName);
		for (int j = 0; j < numbersAmount; j++) {
			writeStream << j << '\n';
		}
	}
	return files;
}

void clearFile(string fileName) {
```

```
	ofstream writeStream;
	writeStream.open(fileName);
	writeStream.close();
}

bool isPrimeNumber(int x) {
	for (int i = 2; i <= sqrt(x); i++)
		if (x % i == 0)
			return false;
	return true;
}
```

3. Диаграмма последовательности.

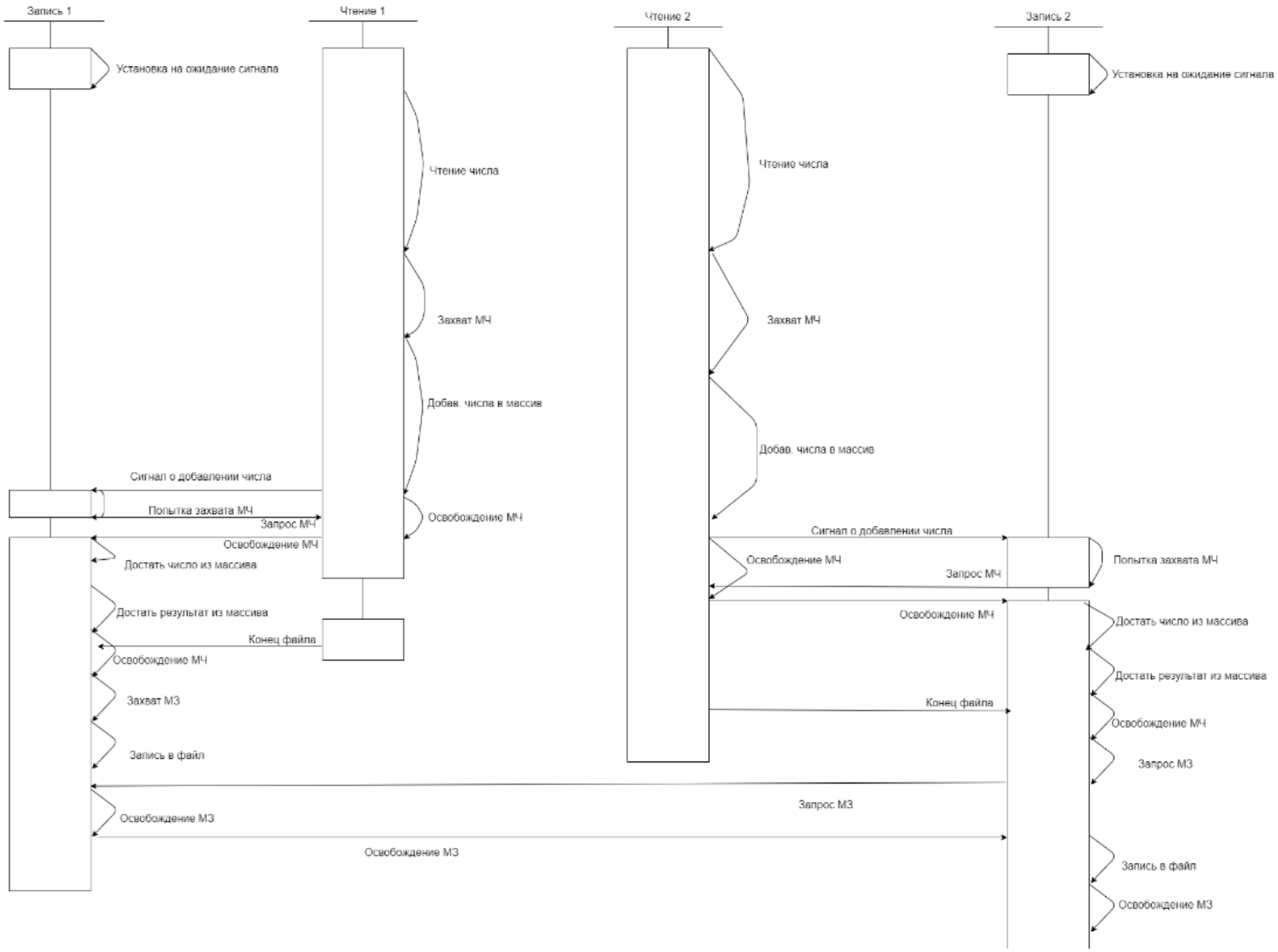


Рисунок 3 – Диаграмма последовательности типовой ситуации, в которой 2 пары потоков.

5. Результаты работы программы.
200000 элементов, 2 пары потоков:

В файлах input(0-1) записаны последовательно числа от 0 до 200000.

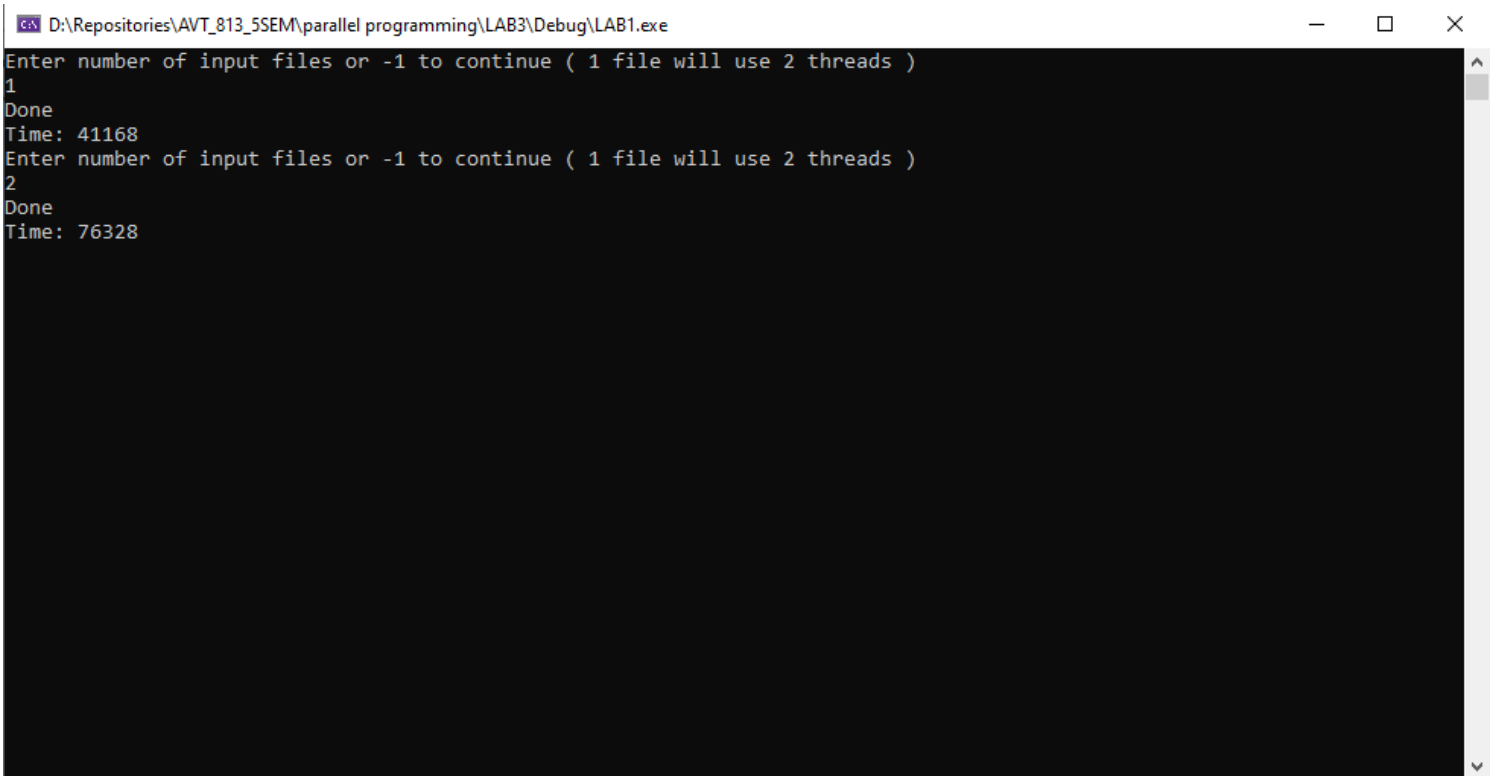


Рисунок 4 – Время работы программы с разным количеством потоков

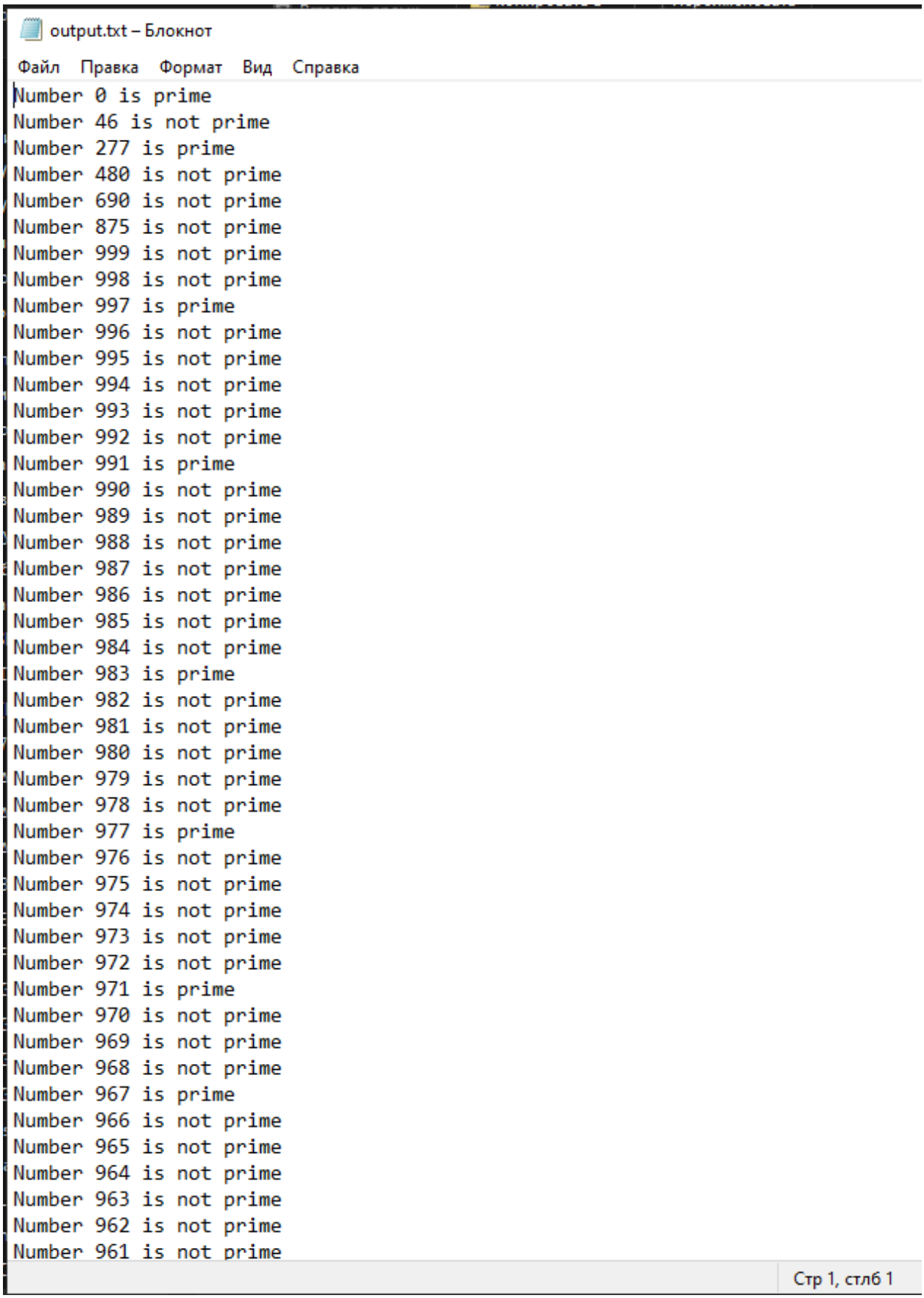


Рисунок 5 – Файл с результатами работы программы.

5. Выводы.

В ходе работы была написана программа, с помощью которой осуществляется проверка чисел на простоту, заданных в файлах чтения. Разложение осуществляется с помощью пар потоков. Для передачи данных внутри пары потоков используется массив.

Есть 2 типа потоков:

1) Reader считывает информацию из файла и вычисляет результат проверки затем передает в поток writer;

2) Writer записывает результаты в выходной файл.

Построена диаграмма последовательности, а также продемонстрированы примеры работы программы с различным количеством пар потоков. При увеличении количества файлов, то есть пар потоков, время выполнения программы не увеличивается прямо пропорционально, так как за чтение данных из входных файлов отвечают параллельно работающие потоки. В целом время выполнения программы увеличивается за счёт общего файла вывода.