

МИНОБРНАУКИ РОССИИ  
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ  
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)  
Кафедра Вычислительной техники

ОТЧЁТ  
по лабораторной работе №3  
по дисциплине «Операционные системы»  
Тема: Процессы и потоки

Студент гр. 9308

Соболев М.С.

Преподаватель

Тимофеев А.В.

Санкт-Петербург,  
2021

## Оглавление

1. Введение .....	3
2. Реализация многопоточного приложения с использованием функций Win32 API.....	5
2.1. Замеры времени выполнения приложения для разного числа потоков ..	5
2.2. Вывод по реализации многопоточного приложения с использованием технологии Win32 API.....	8
2.3. Исходный код программы.....	10
2.4. Вывод .....	17
3. Реализация многопоточного приложения с использованием технологии OpenMP .....	18
3.1. Замеры времени выполнения приложения для разного числа потоков	18
3.2. Вывод по реализации многопоточного приложения с использованием технологии OpenMP .....	21
3.3. Исходный код программы.....	23
3.4. Вывод .....	27
4. Список использованных источников .....	28

# 1. Введение

Тема работы: Процессы и потоки.

Цель работы: исследовать механизмы создания и управления процессами и потоками в ОС Windows.

Указания к выполнению

Задание 3.1. Реализация многопоточного приложения с использованием функций Win32 API.

1. Создайте приложение, которое вычисляет число  $\pi$  с точностью N знаков после запятой по следующей формуле

$$\pi = \left( \frac{4}{1+x_0^2} + \frac{4}{1+x_1^2} + \dots + \frac{4}{1+x_{N-1}^2} \right) \times \frac{1}{N}, \text{ где } x_i = (i+0.5) \times \frac{1}{N}, i = \overline{0, N-1}$$

, где  $N = 100000000$ .

– Используйте распределение итераций блоками (размер блока =  $10 * N_{\text{студбилета}}$ ) по потокам. Сначала каждый поток по очереди получает свой блок итераций, затем тот поток, который заканчивает выполнение своего блока, получает следующий свободный блок итераций. Освободившиеся потоки получают новые блоки итераций до тех пор, пока все блоки не будут исчерпаны.

– Создание потоков выполняйте с помощью функции Win32 API CreateThread.

– Для реализации механизма распределения блоков итераций необходимо сразу в начале программы создать необходимое количество потоков в приостановленном состоянии, для освобождения потока из приостановленного состояния используйте функцию Win32 API ResumeThread.

– По окончании обработки текущего блока итераций поток не должен завершаться, а должен быть, например, приостановлен с помощью функции Win32 API SuspendThread. Затем потоку должен быть предоставлен следующий

свободный блок итераций, и поток должен быть освобождён, например, с помощью функции Win32 API ResumeThread.

2. Произведите замеры времени выполнения приложения для разного числа потоков (1, 2, 4, 8, 12, 16). По результатам измерений постройте график и определите число потоков, при котором достигается наибольшая скорость выполнения. Запротоколируйте результаты в отчёт.

3. Подготовьте итоговый отчёт с развёрнутыми выводами по заданию.

Задание 3.2. Реализация многопоточного приложения с использованием технологии OpenMP.

Указания к выполнению.

1. Создайте приложение, которое вычисляет число  $\pi$  с точностью N знаков после запятой по следующей формуле

$$\pi = \left( \frac{4}{1+x_0^2} + \frac{4}{1+x_1^2} + \dots + \frac{4}{1+x_{N-1}^2} \right) \times \frac{1}{N}, \text{ где } x_i = (i+0.5) \times \frac{1}{N}, i = \overline{0, N-1}$$

, где  $N = 100000000$ .

– Распределите работу по потокам с помощью OpenMP-директивы for.

– Используйте динамическое планирование блоками итераций (размер блока =  $10 * N_{\text{студбилета}}$ ).

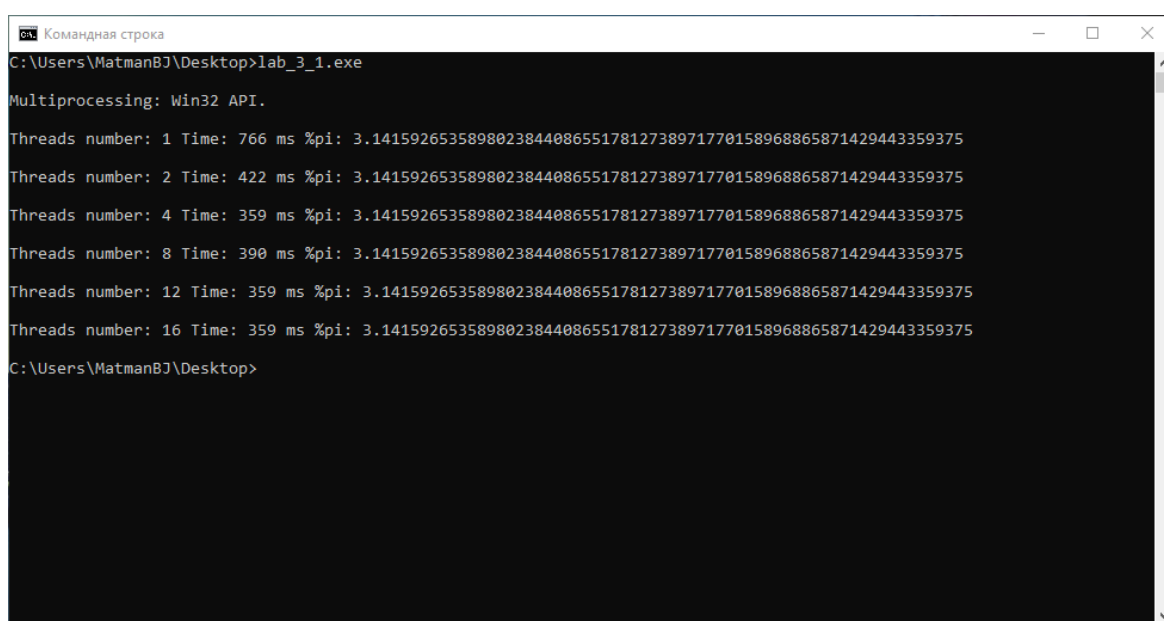
2. Произведите замеры времени выполнения приложения для разного числа потоков (1, 2, 4, 8, 12, 16). По результатам измерений постройте график и определите число потоков, при котором достигается наибольшая скорость выполнения. Запротоколируйте результаты в отчёт, сравните с результатами прошлой работы.

3. Подготовьте итоговый отчёт с развёрнутыми выводами по заданию.

## 2. Реализация многопоточного приложения с использованием функций Win32 API

### 2.1. Замеры времени выполнения приложения для разного числа потоков

Многопоточное приложение выводит результаты выполнения работы – время вычисления числа  $\pi$  и само число  $\pi$  – в терминал. В работе было сделано 5 замеров.



```
Командная строка
C:\Users\MatmanBJ\Desktop>lab_3_1.exe
Multiprocessing: Win32 API.
Threads number: 1 Time: 766 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 2 Time: 422 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 4 Time: 359 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 8 Time: 390 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 12 Time: 359 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 16 Time: 359 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
C:\Users\MatmanBJ\Desktop>
```

Рисунок 1: Замер времени выполнения 1 с использованием технологии Win 32 API

```
cmd Командная строка
C:\Users\MatmanBJ\Desktop>lab_3_1.exe

Multiprocessing: Win32 API.

Threads number: 1 Time: 719 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 2 Time: 453 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 4 Time: 359 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 8 Time: 375 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 12 Time: 360 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 16 Time: 360 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
C:\Users\MatmanBJ\Desktop>
```

Рисунок 2: Замер времени выполнения 2 с использованием технологии Win 32 API

```
cmd Командная строка
C:\Users\MatmanBJ\Desktop>lab_3_1.exe

Multiprocessing: Win32 API.

Threads number: 1 Time: 718 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 2 Time: 422 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 4 Time: 359 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 8 Time: 359 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 12 Time: 375 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 16 Time: 344 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
C:\Users\MatmanBJ\Desktop>
```

Рисунок 3: Замер времени выполнения 3 с использованием технологии Win 32 API

```
Командная строка
C:\Users\MatmanBJ\Desktop>lab_3_1.exe

Multiprocessing: Win32 API.

Threads number: 1 Time: 704 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 2 Time: 437 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 4 Time: 375 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 8 Time: 360 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 12 Time: 343 ms %pi: 3.141592653589802980503009610657727534999139606952667236328125
Threads number: 16 Time: 343 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
C:\Users\MatmanBJ\Desktop>
```

Рисунок 4: Замер времени выполнения 4 с использованием технологии Win 32 API

```
Командная строка
C:\Users\MatmanBJ\Desktop>lab_3_1.exe

Multiprocessing: Win32 API.

Threads number: 1 Time: 703 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 2 Time: 406 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 4 Time: 375 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 8 Time: 375 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 12 Time: 359 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
Threads number: 16 Time: 359 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
C:\Users\MatmanBJ\Desktop>
```

Рисунок 5: Замер времени выполнения 5 с использованием технологии Win 32 API

## 2.2. Вывод по реализации многопоточного приложения с использованием технологии Win32 API

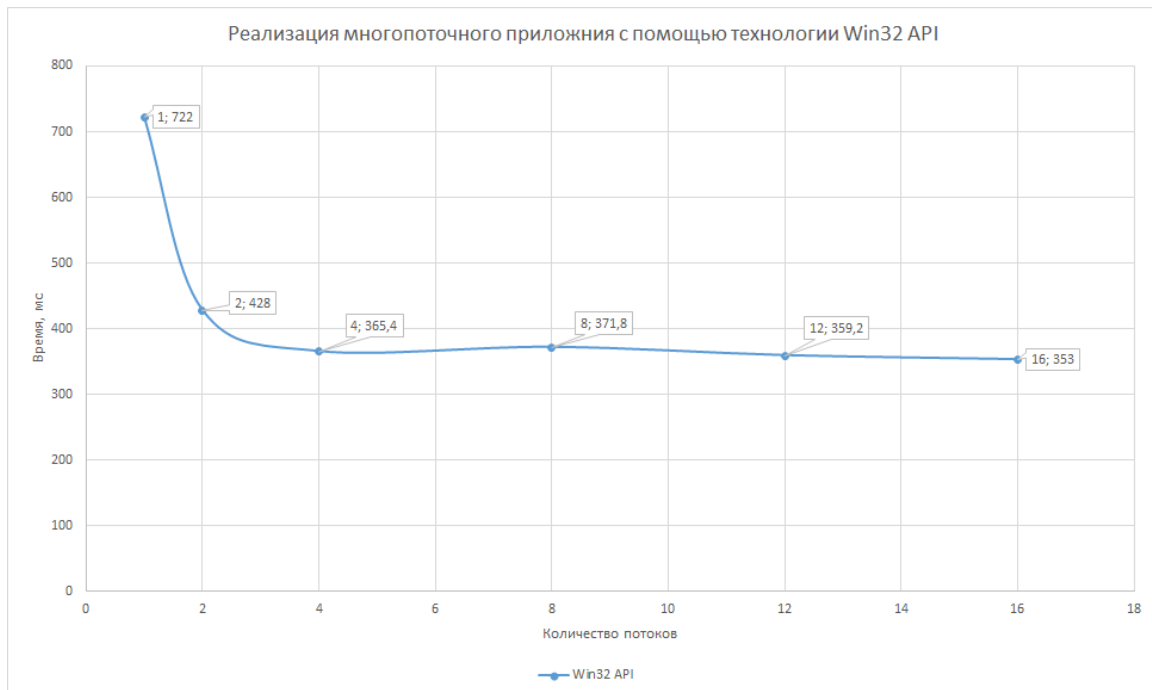


Рисунок 6: График зависимости времени выполнения задачи от количества задействованных потоков

В многопоточном приложении с использованием технологии Win32 API по графику среднего значения по 5 замерам видно, что спад времени выполнения задачи (вычисления числа пи) прекращается с 4 потоков. Это связано с тем, что разделение задач между двумя, а затем и четырьмя потоками существенно оптимизирует работу, равномерно распределяя её по нескольким потокам, где она будет выполняться быстрее и примерно одинаковое время, вместо одного, где она будет выполняться долго. Наибольшая скорость выполнения была достигнута при 16 потоках (среднее – 353 мс), но стабилизация скорости выполнения была достигнута на 4 потоках (среднее – 365,4 мс).



Общая производительность (в данном случае – величина времени на графике) с увеличением числа задействованных потоков (в частности, на графике – с четырёх потоков) растёт намного медленнее, чем в начале, а также может упасть (в частности, на графике – на восьми потоках). Это объясняется тем, что наибольшая производительность достигается при количестве потоков, равным количеству процессов. В данном случае, замеры проводились на компьютере с 2 физическими ядрами, но 4 логическими ядрами, что позволило эффективно распределить 4 потока. При большом количестве потоков необходимо вытеснять одни потоки другими, чтобы у них была возможность выполнять задачи. Поэтому время тратится на ожидание других потоков, на операции планирования и на другое.

## 2.3. Исходный код программы

/\*

Program:

Saint-Petersburg ETU OS laboratory work 3 part 1

Author:

Matvey Sobolev, 2021

Compiler:

g++ (x86\_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0

Copyright (C) 2018 Free Software Foundation, Inc.

This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

\*/

#include <iostream>

#include <windows.h>

#include <string>

#include <iomanip>

#include <list>

#include <numeric>

using namespace std;

// ----- VARIABLES INITIALIZATION -----

DWORD startTime = 0; // starting counting pi-number point

```

DWORD finishTime = 0; // ending counting pi-number point
DWORD allTime = -1; // milliseconds, which will take the pi counting
size_t blocksIterator; // current block
size_t blocksNumber; // number of all blocks
HANDLE synchIteration; // synchronizing iteration mutex
HANDLE synchSummary; // synchronizing summary mutex
const size_t BLOCKSIZE = 10 * 930824; // iteration distribution for threads
const size_t N = 100000000; // N iterations (not signs after comma)

list<long double> list1; // list of the all parts to summary
long double summaryResult = 0.0; // final pi result for each number of threads

// ----- FUNCTION DECLARATION -----

DWORD WINAPI countingPI (LPVOID localInThreads);
void preparingPI (int localNumberOfThreads);

// ----- MAIN -----

int main (int argc, char **argv)
{
    int numberOfThreads[] = { 1, 2, 4, 8, 12, 16 }; // number of threads
    int arraySize = sizeof(numberOfThreads)/sizeof(numberOfThreads[0]); // counting an array size
    long double piNumber; // the %pi number
    cout << "\nMultiprocessing: Win32 API.\n";

    for (int i = 0; i < arraySize; i++)
    {
        list1.clear();
        summaryResult = 0.0;
        preparingPI(numberOfThreads[i]);
    }
}

```

```

        cout << "\nThreads number: " << numberOfThreads[i] << " Time: " << allTime << " ms" <<
setprecision(N) << " %pi: " << summaryResult << "\n";
    }
    return 0;
}

```

// ----- FUNCTION'S BODY -----

```

DWORD WINAPI countingPI(LPVOID localInThreads)

```

```

{
    int i; // iterator
    int localIndicator = 1; // end indicator
    size_t startPoint = 2;
    size_t endPoint = 1;
    long double localResult = 0.0; // local result summary

```

```

    while (localIndicator != 0)

```

```

    {
        // SYNCHRONIZING ITERATIONS -- START

```

```

        DWORD waitError = WaitForSingleObject (synchIteration, INFINITE); // while isn't released,
i can't quit

```

```

        if (waitError != WAIT_OBJECT_0)
        {
            cout << "Sorry, you have error w/ sunchIteration (" << waitError << "). Last error number: "
<< GetLastError() << "\n";
        }

```

```

        if (blocksIterator < blocksNumber)
        {

```

```

        startPoint = blocksIterator * BLOCKSIZE; // blocksize number start
(iteration*number_of_items_in_block)

        endPoint = (blocksIterator + 1) * BLOCKSIZE - 1; // blocksize number end
(iteration*number) // HERE CHANGED FORMULA

        if (endPoint > N - 1) // checking for out of range error
        {
            endPoint = N - 1;
        }
        blocksIterator = blocksIterator + 1; // increasing iteration number
    }
else
{
    startPoint = 2;
        endPoint = 1;
}

ReleaseMutex (synchIteration);

// SYNCHRONIZING ITERATIONS -- END

if (startPoint <= endPoint)
{
    localResult = 0.0;

    for (i = startPoint; i++ <= endPoint; ) // formula counting
    {
        localResult = localResult + (4 / (1 + (((long double)i + 0.5) / (long double)N)*(((long
double)i + 0.5) / (long double)N)));
    }

    // SYNCRONIZING SUMMARY -- BEGIN

```

```

waitError = WaitForSingleObject (synchSummary, INFINITE);

if (waitError != WAIT_OBJECT_0)
{
    cout << "Sorry, you have problem w/ synchSummary (" << waitError << "). Last error
number: " << GetLastError() << endl;
}

list1.push_back(localResult); // adding the result to the list

ReleaseMutex (synchSummary);

// SYNCHRONIZING SUMMARY -- END
}
else
{
    localIndicator = 0;
}
}

return 0;
}

void preparingPI (int localNumberOfThreads)
{
    // 1 -- PREPARING AND INITIALIZING

    // initilaizing objects and variables

    blocksIterator = 0; // setting to null block iterator
    blocksNumber = N % BLOCKSIZE == 0 ? (N / BLOCKSIZE) : (N / BLOCKSIZE + 1); // if div
is full or not

```

```

int i = 0; // iterator
HANDLE *threadsArray = new HANDLE[localNumberOfThreads];
synchIteration = CreateMutex (NULL, FALSE, NULL); // synchronizing object for selected
iterations
synchSummary = CreateMutex (NULL, FALSE, NULL); // synchronizing object for summary
counting

// 2 -- CHECKING THREADS AND CREATING THREADS

// creating threads for counting pi-number (just creating and setting threads here)

for (i = 0; i < localNumberOfThreads; i++)
{
    threadsArray[i] = CreateThread (NULL, 0, countingPI, NULL, CREATE_SUSPENDED,
NULL);
}

// 3 -- COUNTING PI-NUMBER

// starting the timer

startTime = GetTickCount();

// starting threads for counting pi-number (just starting here)

for (unsigned i = 0; i < localNumberOfThreads; i++)
{
    ResumeThread (threadsArray[i]);
}

// waiting until all threads will be released

```

```
DWORD waitError = WaitForMultipleObjects(localNumberOfThreads, threadsArray, true, INFINITE);
```

```
// making the final result
```

```
summaryResult = std::accumulate(std::begin(list1), std::end(list1), 0.0);  
summaryResult = summaryResult / N;
```

```
// ending the timer
```

```
finishTime = GetTickCount();
```

```
// counting final time
```

```
allTime = finishTime - startTime;
```

```
// 4 -- ENDING AND CLEANING
```

```
// "cleaning": closing handles and cleaning memory
```

```
for (i = 0; i < localNumberOfThreads; ++i)  
{  
    CloseHandle(threadsArray[i]);  
}  
CloseHandle(synchIteration);  
CloseHandle(synchSummary);  
delete threadsArray;  
}
```



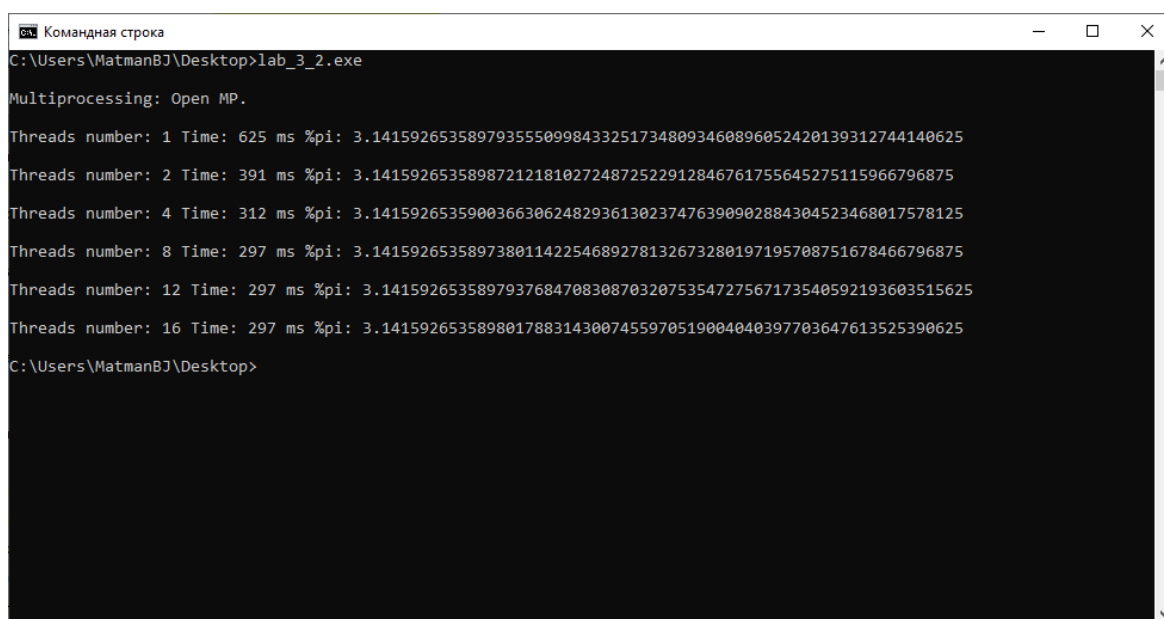
## 2.4. Вывод

В ходе выполнения первой части («Реализация многопоточного приложения с использованием технологии Win32 API») лабораторной работы 3 «Процессы и потоки» была освоена технология распараллеливания на основе Win32API. В частности, с помощью технологии распараллеливания Win32 API в программе было вычислено число пи с размером блока итерации для каждого потока  $10 \cdot 930824$  и общим количеством итераций 100000000. В работе были проведены замеры для 1, 2, 4, 8, 12 и 16 потоков соответственно, а также составлены графики зависимости времени от количества потоков на данную задачу. Была выявлена зависимость времени числа задействованных потоков от числа логических процессов. Таким образом и было реализовано многопоточное приложение с использованием технологии Win32 API.

### 3. Реализация многопоточного приложения с использованием технологии OpenMP

#### 3.1. Замеры времени выполнения приложения для разного числа потоков

Многопоточное приложение выводит результаты выполнения работы – время вычисления числа  $\pi$  и само число  $\pi$  – в терминал. В работе было сделано 5 замеров.



```
Командная строка
C:\Users\MatmanBJ\Desktop>lab_3_2.exe
Multiprocessing: Open MP.
Threads number: 1 Time: 625 ms %pi: 3.14159265358979355509984332517348093460896052420139312744140625
Threads number: 2 Time: 391 ms %pi: 3.1415926535898721218102724872522912846761755645275115966796875
Threads number: 4 Time: 312 ms %pi: 3.14159265359003663062482936130237476390902884304523468017578125
Threads number: 8 Time: 297 ms %pi: 3.14159265358973801142254689278132673280197195708751678466796875
Threads number: 12 Time: 297 ms %pi: 3.14159265358979376847083087032075354727567173540592193603515625
Threads number: 16 Time: 297 ms %pi: 3.1415926535898017883143007455970519004040397703647613525390625
C:\Users\MatmanBJ\Desktop>
```

Рисунок 7: Замер времени выполнения 1 с использованием технологии OpenMP

```
Командная строка
C:\Users\MatmanBJ\Desktop>lab_3_2.exe

Multiprocessing: Open MP.

Threads number: 1 Time: 641 ms %pi: 3.14159265358979355509984332517348093460896052420139312744140625
Threads number: 2 Time: 485 ms %pi: 3.1415926535897374153281924602509889155044220387935638427734375
Threads number: 4 Time: 328 ms %pi: 3.1415926535898095368903870650001408648677170276641845703125
Threads number: 8 Time: 344 ms %pi: 3.14159265358974456780992434712374006267054937779903411865234375
Threads number: 12 Time: 312 ms %pi: 3.14159265358982032188307764730694771060370840132236480712890625
Threads number: 16 Time: 328 ms %pi: 3.1415926535898017883143007455970519004040397703647613525390625
C:\Users\MatmanBJ\Desktop>
```

Рисунок 8: Замер времени выполнения 2 с использованием технологии OpenMP

```
Командная строка
C:\Users\MatmanBJ\Desktop>lab_3_2.exe

Multiprocessing: Open MP.

Threads number: 1 Time: 609 ms %pi: 3.14159265358979355509984332517348093460896052420139312744140625
Threads number: 2 Time: 407 ms %pi: 3.14159265359006297543657815207751582420314662158489227294921875
Threads number: 4 Time: 313 ms %pi: 3.141592653590023517633234018120447217370383441448211669921875
Threads number: 8 Time: 312 ms %pi: 3.141592653589711189345001773887133822427131235599517822265625
Threads number: 12 Time: 297 ms %pi: 3.14159265358982443816504570577308186329901218414306640625
Threads number: 16 Time: 297 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
C:\Users\MatmanBJ\Desktop>
```

Рисунок 9: Замер времени выполнения 3 с использованием технологии OpenMP

```
Командная строка
C:\Users\MatmanBJ\Desktop>lab_3_2.exe

Multiprocessing: Open MP.

Threads number: 1 Time: 625 ms %pi: 3.14159265358979355509984332517348093460896052420139312744140625
Threads number: 2 Time: 437 ms %pi: 3.14159265359006297543657815207751582420314662158489227294921875
Threads number: 4 Time: 312 ms %pi: 3.141592653590060472447442752041979474597610533237457275390625
Threads number: 8 Time: 296 ms %pi: 3.14159265358973801142254689278132673280197195708751678466796875
Threads number: 12 Time: 297 ms %pi: 3.14159265358982384207069127324274404600146226584911346435546875
Threads number: 16 Time: 312 ms %pi: 3.1415926535898017883143007455970519004040397703647613525390625
C:\Users\MatmanBJ\Desktop>
```

Рисунок 10: Замер времени выполнения 4 с использованием технологии OpenMP

```
Командная строка
C:\Users\MatmanBJ\Desktop>lab_3_2.exe

Multiprocessing: Open MP.

Threads number: 1 Time: 625 ms %pi: 3.14159265358979355509984332517348093460896052420139312744140625
Threads number: 2 Time: 453 ms %pi: 3.14159265358981788156082781693356764662894420325756072998046875
Threads number: 4 Time: 328 ms %pi: 3.14159265358997702595987566720481254378682933747768402099609375
Threads number: 8 Time: 312 ms %pi: 3.14159265358975827689587412283600542650674469769001007080078125
Threads number: 12 Time: 297 ms %pi: 3.141592653589800623013805758176886229193769395351409912109375
Threads number: 16 Time: 297 ms %pi: 3.14159265358980238440865517812738971770158968865871429443359375
C:\Users\MatmanBJ\Desktop>
```

Рисунок 11: Замер времени выполнения 5 с использованием технологии OpenMP

### 3.2. Вывод по реализации многопоточного приложения с использованием технологии OpenMP

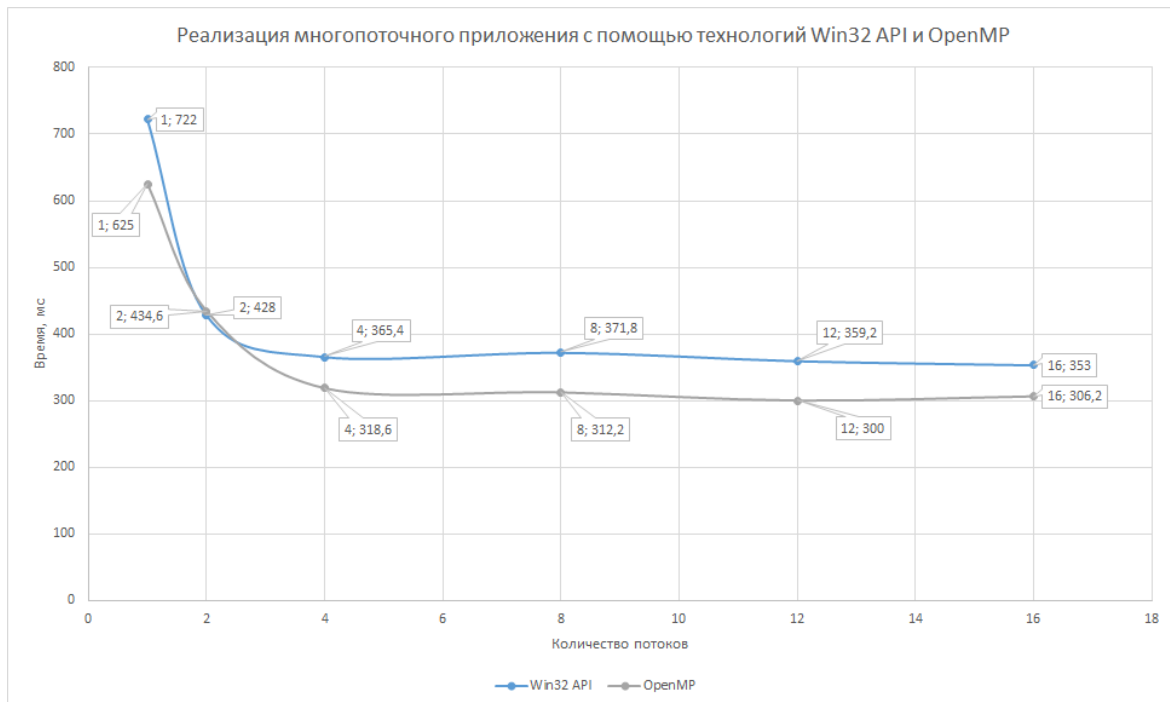


Рисунок 12: График зависимости времени выполнения задачи от количества задействованных потоков

В многопоточном приложении с использованием технологии OpenMP по графику среднего значения по 5 замерам видно, что спад времени выполнения задачи (вычисления числа пи) прекращается с 4 потоков. Данная тенденция аналогична графику многопоточного приложения с использованием технологии Win32 API. Наибольшая скорость выполнения была достигнута при 12 потоках (среднее – 300 мс), но стабилизация скорости выполнения была достигнута на 4 потоках (среднее – 318,6 мс).

На графике также видно, что использование технологии OpenMP при выполнении задачи занимает немного меньше времени, чем использование технологии Win32 API. Скорее всего, это связано с тем, что распределение крупной вычислительной задачи (в данном случае – вычисление числа пи)

согласно данным Microsoft хорошо подходит к выполнению на стандарте OpenMP.

### 3.3. Исходный код программы

/\*

Program:

Saint-Petersburg ETU OS laboratory work 3 part 2

Author:

Matvey Sobolev, 2021

Compiler:

g++ (x86\_64-posix-seh-rev0, Built by MinGW-W64 project) 8.1.0

Copyright (C) 2018 Free Software Foundation, Inc.

This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.

ATTENTION:

WHILE COMPILING DO "g++ -fopenmp -o .\a.exe main.cpp"

\*/

#include <iostream>

#include <windows.h>

#include <string>

#include <iomanip>

#include <omp.h>

using namespace std;

```
// ----- VARIABLES INITIALIZATION -----

DWORD startTime = 0; // starting counting pi-number point
DWORD finishTime = 0; // ending counting pi-number point
DWORD allTime = -1; // milliseconds, which will take the pi counting
const size_t BLOCKSIZE = 10 * 930824; // iteration distribution for threads
const size_t N = 100000000; // N iterations (not signs after comma)

// ----- FUNCTION DECLARATION -----

long double countingPI (size_t localIterations, size_t localBlocksize, int localNumberOfThreads);

// ----- MAIN -----

int main (int argc, char **argv)
{
    int numberOfThreads[] = { 1, 2, 4, 8, 12, 16 }; // number of threads
    int arraySize = sizeof(numberOfThreads)/sizeof(numberOfThreads[0]); // counting an array size
    long double piNumber; // the %pi number
    cout << "\nMultiprocessing: Open MP.\n";

    for (int i = 0; i < arraySize; i++)
    {
        startTime = 0;
        finishTime = 0;
        allTime = 0;
        piNumber = countingPI(N, BLOCKSIZE, numberOfThreads[i]);
        cout << "\nThreads number: " << numberOfThreads[i] << " Time: " << allTime << " ms" <<
setprecision(N) << " %pi: " << piNumber << "\n";
    }
    return 0;
}
```



```
// ----- FUNCTION'S BODY -----
```

```
long double countingPI (size_t localIterations, size_t localBlocksize, int localNumberOfThreads)
```

```
{
```

```
    // 1 -- COUNTING PI-NUMBER
```

```
    int i = 0; // iterator
```

```
    // starting the timer
```

```
    startTime = GetTickCount();
```

```
    long double summaryResult = 0.0;
```

```
    #pragma omp parallel shared(startTime, finishTime, allTime) reduction (+: summaryResult)
num_threads(localNumberOfThreads)
```

```
    {
```

```
        #pragma omp for schedule(dynamic, localBlocksize) nowait
```

```
        for (i = 0; i < localIterations; i++)
```

```
        {
```

```
            summaryResult = summaryResult + (4 / (1 + (((long double)i + 0.5) / (long
double)localIterations)*(((long double)i + 0.5) / (long double)localIterations)));
```

```
        }
```

```
    }
```

```
    // making the final result
```

```
    summaryResult = summaryResult / localIterations;
```

```
    // ending the timer
```

```
    finishTime = GetTickCount();
```

```
// counting final time

allTime = finishTime - startTime;

return summaryResult;
}
```

### 3.4. Вывод

В ходе выполнения второй части («Реализация многопоточного приложения с использованием технологии OpenMP») лабораторной работы 3 «Процессы и потоки» была освоена технология OpenMP, позволяющая на программном уровне осуществить распараллеливание приложения. В частности, с помощью технологии OpenMP в программе было вычислено число  $\pi$  с размером блока итерации для каждого потока  $10 \cdot 930824$  и общим количеством итераций 100000000. В работе были проведены замеры для 1, 2, 4, 8, 12 и 16 потоков соответственно, а также составлены графики зависимости времени от количества потоков на данную задачу. Было осуществлено сравнение замеров с замерами технологии Win32 API, в результате чего была замечена более быстрая работа технологии OpenMP. Таким образом и было реализовано многопоточное приложение с использованием технологии OpenMP.

## 4. Список использованных источников

1. Операционные системы: электронные методические указания к лабораторным работам / Сост.: А. В. Тимофеев. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2016.
2. Таненбаум Э. Современные операционные системы. 2-е изд. – СПб.: Питер, 2002. – 1040 с.: ил.
3. Столлингс, Вильям. Операционные системы, 4-е издание. : Пер. с англ. – М. : Издательский дом «Вильямс», 2002. – 848 с. : ил. – Парал. Тит. Англ.
4. Документация «Microsoft» [сайт]. URL: <https://docs.microsoft.com/ru-ru/cpp/parallel/concrt/comparing-the-concurrency-runtime-to-other-concurrency-models?view=msvc-170>.
5. Курс «Операционные системы» в образовательной онлайн-системе Google Класс [сайт]. URL: <https://classroom.google.com/c/Mzg3ODc4NDE5MDU4>.