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

Кафедра ВТ

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

Студент гр. 8307 _____ Репин С.А.

Преподаватель _____ Тимофеев А.В.

Санкт-Петербург
2020

СОДЕРЖАНИЕ

ЦЕЛЬ РАБОТЫ

Исследовать механизмы управления виртуальной памятью Win32.

ВВЕДЕНИЕ

При выполнении лабораторной работы на языке программирования C стандарта C11 было разработано консольной приложение, управление которым происходит через меню, пункты которого соответствуют пунктам заданий. Исходный код приложения доступен на GitHub [1].

Файл	Описание
menu.c	Определение типов и функций для работы с меню
main.c	Точка входа в программу; объявления конкретных меню и переходов между ними
actions.c	Реализация функций непосредственно выполняющих требования заданий (другими словами, callback'и конечных пунктов меню)
error.c	Описание номеров ошибок, а также функции отображения сообщений об ошибках

Таблица 1 Описание файлов в проекте

Сборка проекта производится с помощью Powershell-скрипта *build.ps1* (следует создать папку *build* и запускать скрипт из нее). Также потребуется пакет Build Tools for Visual Studio 2019.

Стоит обратить внимание на некоторые флаги компилятора, с которыми производилась сборка:

- */O2*. Указывает использовать набор оптимизаций по скорости
- */GL*. Включает оптимизацию всей программы (оптимизация производится на основе владения информацией обо всех модулях программы, обычно она производится только на основе одного compilation unit)

[1] https://github.com/stnrepin/os_labs/tree/master/lab3

- */Fa*. Создает дополнительно файлы сгенерированного ассемблерного кода. Удобно для анализа работы оптимизаций
- */arch:AVX2*. Включает использование расширения x86_64 AVX2
- */openmp*. Подключает библиотеку OpenMP

1. РЕАЛИЗАЦИЯ МНОГОПОТОЧНОГО ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ ФУНКЦИЙ WIN32 API

Используя функции Win32 API (*CreateThread()*, *SuspendThread()*, *ResumeThread()*, *CreateEvent()*, *SetEvent()*, *WaitForMultipleObjects()*) была написана многопоточная программа, которая распределяя итерации по потокам блоками в $830716 \cdot 10$ итераций, вычисляет число π по следующей формуле:

$$\pi = \frac{1}{N} \sum_{i=0}^{N-1} \frac{4}{1 + x_i^2}, \quad x_i = \frac{i + 0.5}{N}, \quad N = 100\ 000\ 000 \tag{1.1}$$

В итоге были получены следующие результаты замеров времени выполнения программы в зависимости от числа потоков (время измеряется в миллисекундах):

№	Число потоков					
	1	2	4	8	12	16
1	235.265	113.653	56.288	55.988	56.618	56.234
2	233.383	114.756	56.179	56.296	56.131	56.451
3	234.940	114.601	56.405	55.999	56.416	56.693
	234.529	114.337	56.291	56.094	56.388	56.459

Построим график зависимости времени выполнения программы от количества потоков (приведем два графика, чтобы разница по времени при большом количестве потоков была различима).

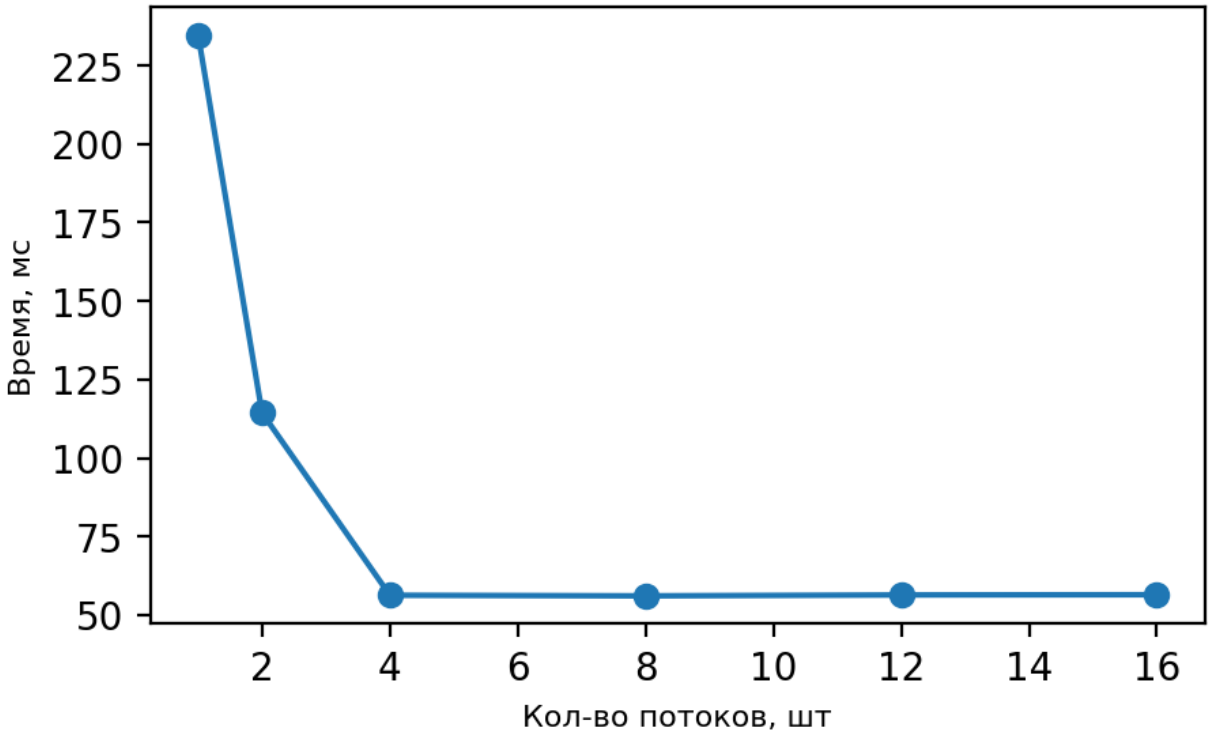

Рис. 1.1 Зависимость времени от числа потоков (Win32 API)

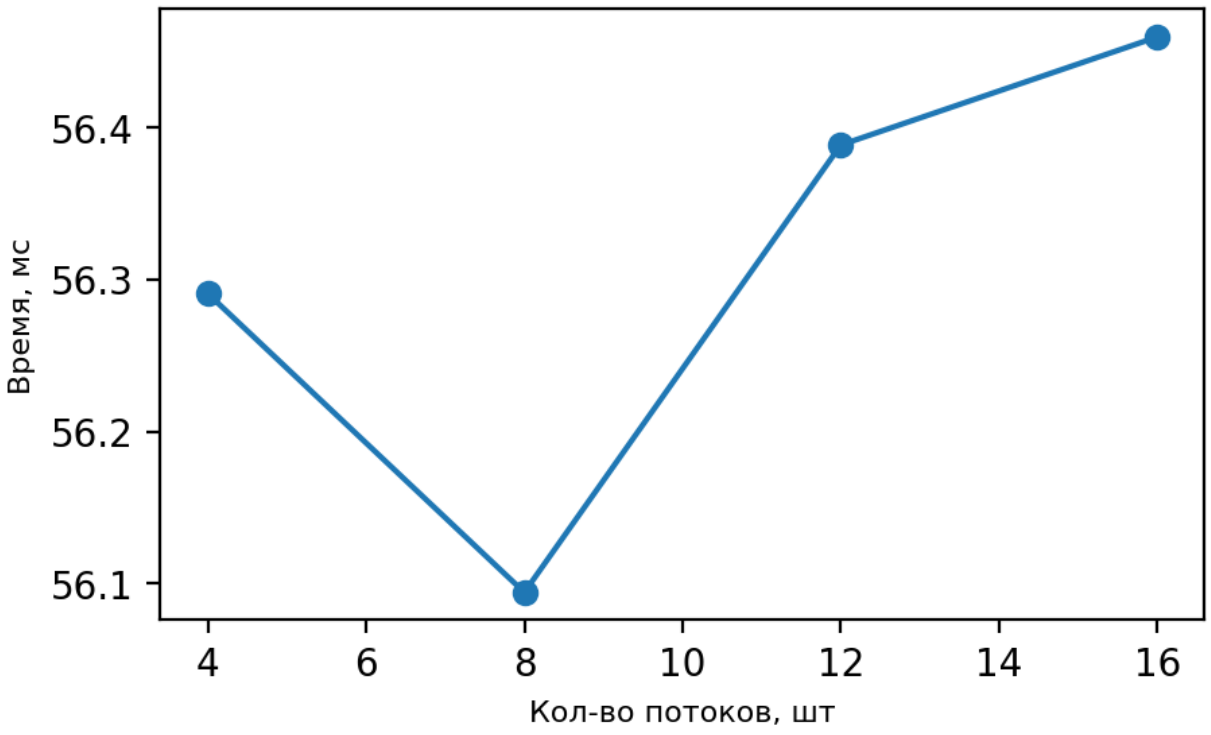

Рис. 1.2 Зависимость времени от числа потоков (Win32 API)

Как видим, максимальное быстродействие достигается при 4 или 8 потоках (в зависимости от того, считать ли 300мкс погрешностью или нет).

2. РЕАЛИЗАЦИЯ МНОГОПОТОЧНОГО ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ OPENMP

OpenMP – это открытый стандарт API для разработки параллельных программ для C, C++, Fortran. В данной работе он используется как альтернативный способ выполнения вычислений, описанных в предыдущем разделе. С помощью директив *parallel*, *for*, *schedule*, *reduction* реализуется программа по своей работе аналогичная программе, созданной в предыдущем разделе. Соответственно, ожидается, что время выполнения в обоих случаях будет идентичным.

Были получены следующие результаты замеров времени выполнения программы в зависимости от числа потоков (время измеряется в миллисекундах):

№	Число потоков					
	1	2	4	8	12	16
1	232.053	114.239	56.267	55.576	56.341	56.646
2	231.792	115.124	56.534	56.456	56.401	56.112
3	232.797	114.441	56.319	56.096	56.130	56.416
	232.214	114.601	56.373	56.043	56.290	56.391

Заметим, что, как и предполагалось выше, эти значения практически не отличаются от значений, полученных в предыдущем разделе.

Как и ранее, построим графики.

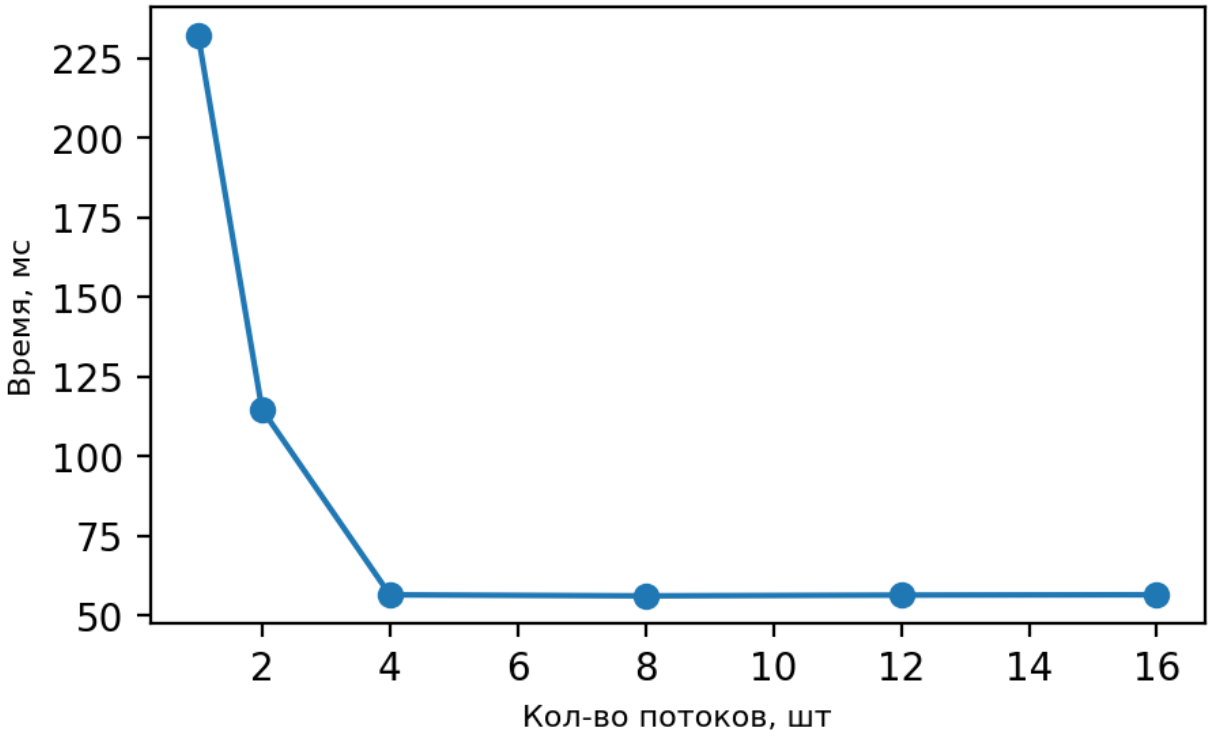

Рис. 2.1 Зависимость времени от числа потоков (OpenMP)

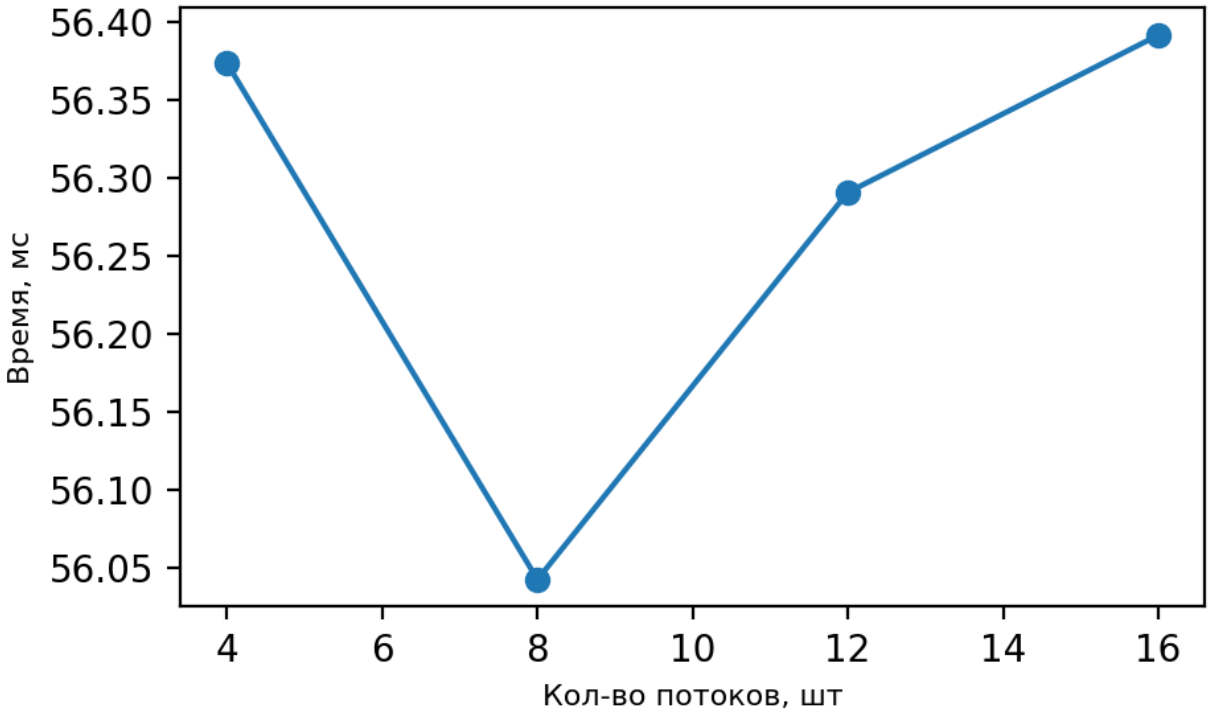

Рис. 2.2 Зависимость времени от числа потоков (OpenMP)

ВЫВОД

В ходе выполнения данной лабораторной работы были получены знания и практические навыки разработки многопоточной программы в Windows, используя как примитивы Win32 API, так и технологию OpenMP. В обоих случаях, время выполнения программы и точность вычислений не отличались. Экспериментально было получено оптимальное количество потоков для выполнения данной задачи — 4 или 8. Кроме того, было продемонстрировано, как применение потоков благоприятно сказывается на времени выполнения программы (как минимум в данной задачи).

При написании приложения с использованием WinAPI была изучена организация потоков в ОС Windows и API ОС для тесного с ними взаимодействия. Были получены знания о жизненном цикле потоков (это было необходимо для динамического распределения работы по потокам), а также был рассмотрен способ реализации ожидания освобождения потоков с помощью событий.

На этом примере была продемонстрирована простота, которую предоставляет технология OpenMP по сравнению с WinAPI, позволяя абстрагировать от низкоуровневых и платформозависимых деталей реализации параллельных программ. Такое упращение очень важно для написании серьезных и крупных приложений, так как минимизирует возможность ошибиться при написании кода, учитывая трудоемкость реализации правильного многопоточного кода и его отладку.

При проектировании и разработке программы нельзя было пройти мимо общих понятий: состояние гонки, примитивы синхронизации, приоритеты потоков, написание программ оптимальных с точки зрения доступа к кэшу процессора и предсказания переходов, знание о которых позволило написать корректное и быстрое приложение. Во время оптимизации программы пришлось познакомиться с некоторыми оптимизациями компилятора: встроенные функции (для бесплатных абстракций), разворачивание цикла (для уменьшения числа итераций и, соответственно, сравнений в цикле, в котором в основном находятся потоки при выполнении), применение векторных инструкций (для ускорения самой частой операции — выполнения i-ой итерации в формуле вычисления числа π) и другие.