Минобрнауки России  
Федеральное государственное автономное образовательное  
Учреждение высшего образования  
«Санкт-Петербургский государственный электротехнический  
Университет им. В.И. Ульянова (Ленина)»  
(СПГЭТУ «ЛЭТИ»)  
Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

отчет

**по лабораторной работе №3**

**по дисциплине «Операционные системы»**

Тема: процессы и потоки

Выполнил студент группы 9308: Дементьев Д.П.

Принял: к.т.н., доцент Тимофеев А.В.

Санкт-Петербург

2021

1. **Цель работы:**

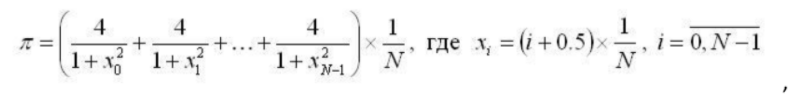
Исследовать механизмы создания и управления процессами и потоками в ОС Windows.

**Ссылка на репозиторий (GitHub):**

https://github.com/CamelNotFemale/5\_sem\_OS/tree/main/lab3

1. **Реализация многопоточного приложения с использованием функций Win32 API.**
   1. **Указания к выполнению.**

Создайте приложение, которое вычисляет число pi с точностью N знаков после запятой по следующей формуле:



где N=100000000.

* Используйте распределение итераций блоками (размер блока = 10\* Nстудбилета) 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 освобожден, например, с помощью функции Win32 API ResumeThread.
  1. **Результаты работы программы.**

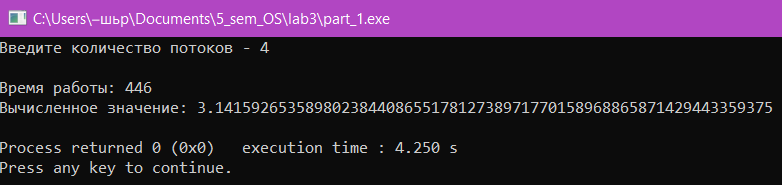


Рис. 1.1. Пример работы программы

Программа спроектирована следующим образом. Сначала пользователь вводит количество потоков. Затем эти потоки создаются в приостановленном состоянии, после чего, с помощью функции ResumeThread, потоки освобождаются из приостановленного состояния и параллельно исполняют функцию, вычисляющую требуемое число.

* 1. **Замеры времени**

Рис.1.2. График зависимости для собственной программы.

Как видно из графика с увеличением числа потоков время подсчета результата уменьшается, однако, с увеличением количества потоков прирост производительности уменьшается. Можно заметить, что при достижении количества потоков, равных количеству логических ядер процессора, время практически не изменяется.

Также наступает момент, когда дальнейшее увеличение количества потоков начинает негативно сказываться на производительности. Это можно объяснить тем, что каждый из потоков требует ресурсы процессора на поддержание своей работоспособности, из чего следует, что наступит момент, когда поддержание работы потоков будет требовать больше ресурсов, чем сама выполняемая программа.

* 1. **Выводы**

В ходе выполнения первого задания лабораторной работы были получены навыки в работе с многопоточными приложениями. Также были сделаны выводы, что при увеличении количества потоков до четырех наблюдается существенный рост производительности, после чего она остается примерно постоянной. В работе использовался двухъядерный процессор с четырьмя логическими процессорами.

Связано это с тем, что для каждого ядра процессора определяется 2 логических процессора, что означает 2 «одновременно» исполняемых потоков на каждом ядре. Так как ядер в процессоре всего 2, количество параллельно выполняющихся потоков равно 4. Таким образом при запуске более 4 потоков улучшение производительности минимальное или отсутствует. Также стоит заметить, что при значительно большом значении количества потоков производительность может падать, вследствие большого количества переключений между потоками.

1. **Реализация многопоточного приложения с использованием технологии OpenMP.**
   1. **Указания к выполнению.**

Создайте приложение, которое вычисляет число pi с точностью N знаков после запятой:

* Распределите работу по потокам с помощью OpenMP-директивы for.
* Используйте динамическое планирование блоками итераций (размер блока = 10 \* Nстудбилета).
  1. **Результаты работы программы и замеры времени**

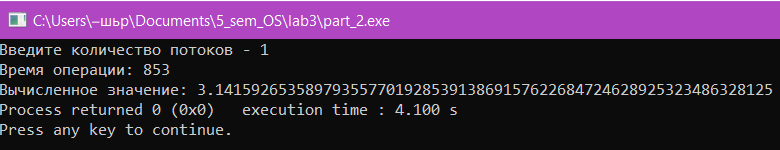


Рис.2.1. Пример работы программы.

Рис.2.2. График зависимости работы OpenMp.

* 1. **Вывод**

В ходе выполнения второго задания лабораторной работы были получены следующие результаты: время работы программы с OpenMP практически идентичные сравнительно с собственно написанной параллельной программой. При объявлении директивы в свойствах самого распараллеливания работы между потоками в условии schedule() был указан способ dynamic. Это означает, что работа распределяется блоками заданного размера между потоками. Как только какой-либо из потоков заканчивает обработку своей порции данных, он захватывает следующую. Стоит отметить, что при этом подходе несколько бОльшие накладные расходы, но можно добиться лучшей балансировки загрузки между потоками. Также наблюдается спад прироста скорости работы приложения с увеличением количества потоков. Наиболее быстрая работа достигается аналогично первому пункту лабораторной работы.