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

Кафедра вычислительной техники

Отчет по лабораторной работе №3
по дисциплине «Операционные системы»

Тема: Процессы и потоки

Студент гр. 9308 Дубенков С.А

Преподаватель Тимофеев А.В.

Санкт-Петербург

2021

Цель работы

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

Код обоих заданий представлен на Github:
https://github.com/sedub01/eltech/tree/main/OS/3

Задание 3.1

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

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

где N=100000000.

В данной лабораторной работе был использован такой метод синхронизации как критические секции. Критическая секция – участок кода, который в определенный момент времени может выполняться только одним потоком. Если код поместить в критическую секцию, то другие потоки не смогут войти в нее до тех пор, пока предыдущий поток не завершит его выполнение. Это единственный метод синхронизации, не требующий привлечения ядра Windows, т.к. КС не является объектом ядра.

Эксперимент был произведен на ПК с AMD Ryzen 5 4600H с 12 логическими ядрами.

Был произведен замер времени по кол-ву потоков (1, 2, 4, 8, 12, 16), меняя значение MAX_THREADS. При проведении эксперимента могут возникать случайные погрешности (одни значения будут больше, другие меньше), поэтому, для того чтобы «сгладить углы», было решено производить не один замер, а 10, суммировать время и выводить среднее арифметическое времени вычисления числа пи.
График представлен ниже.

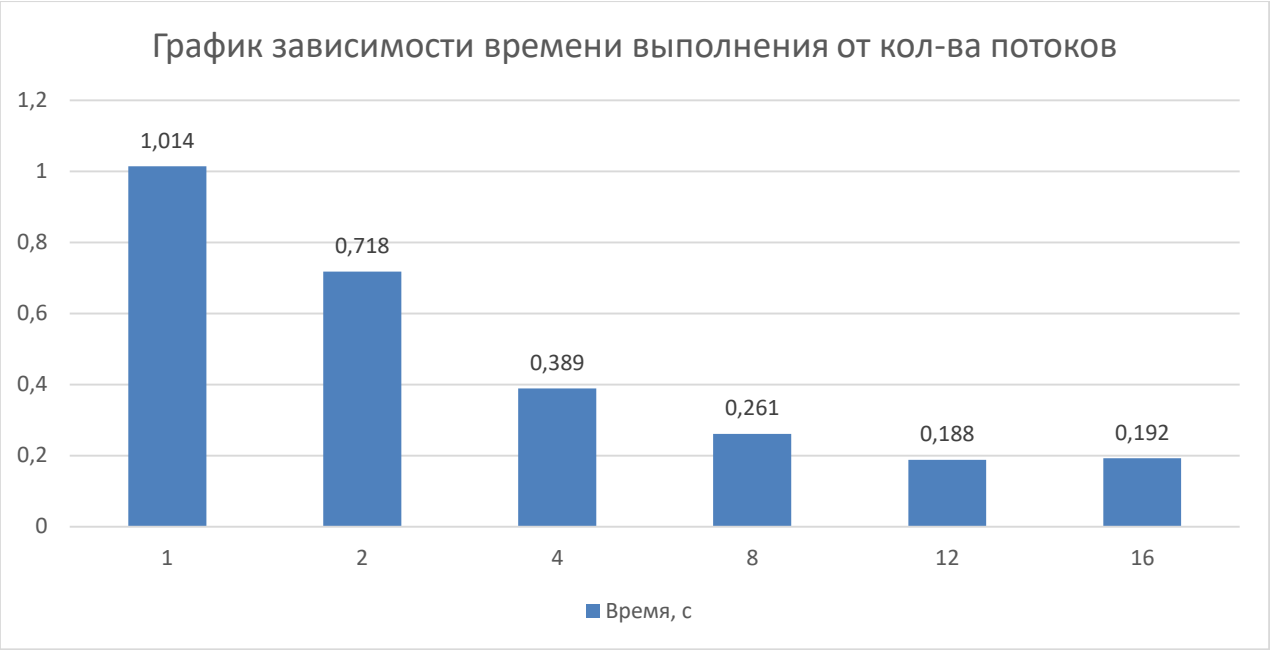

Рисунок 1. График зависимости времени выполнения от кол-ва потоков

Вывод: с увеличением кол-ва потоков уменьшается время выполнения программы, как следствие, повышается ее эффективность. Это происходит потому, что в однопоточной среде вся нагрузка приходилась на одно ядро процессора – теперь же, благодаря распараллеливанию, части программы выполняются одновременно на нескольких ядрах, однако в таком случае необходимо следить за выполнением потоков: за созданием, течением и их удалением.
Сначала время выполнения уменьшалось примерно в два раза, затем темпы уменьшения времени начали снижаться, так как функция ResumeThread вызывается столько же раз, сколько потоков, из-за чего первый поток запускается и начинает работу, в то время второй поток тоже запускается и начинает работу, однако за время запуска предыдущий поток уже выполнил часть его работы, поэтому, по индукции, следующий поток будет выполнять меньше работы, из-за чего ему достается меньше ресурсов, из-за чего следующие потоки практически не влияют на производительность

Оптимальное количество потоков для моей программы – 12 (столько же, сколько логических ядер процессора)

В современном процессоре одновременно запущены несколько тысяч процессов и потоков, так что написанная программа не входила с ними в конфликт

Задание 3.2

Создайте приложение, которое вычисляет число pi с точностью

N знаков после запятой по следующей формуле

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

где N=100000000 с помощью OpenMP-стандарта.

Был произведен замер времени по кол-ву потоков (1, 2, 4, 8, 12, 16), меняя значение MAX_THREADS.

График приведен ниже

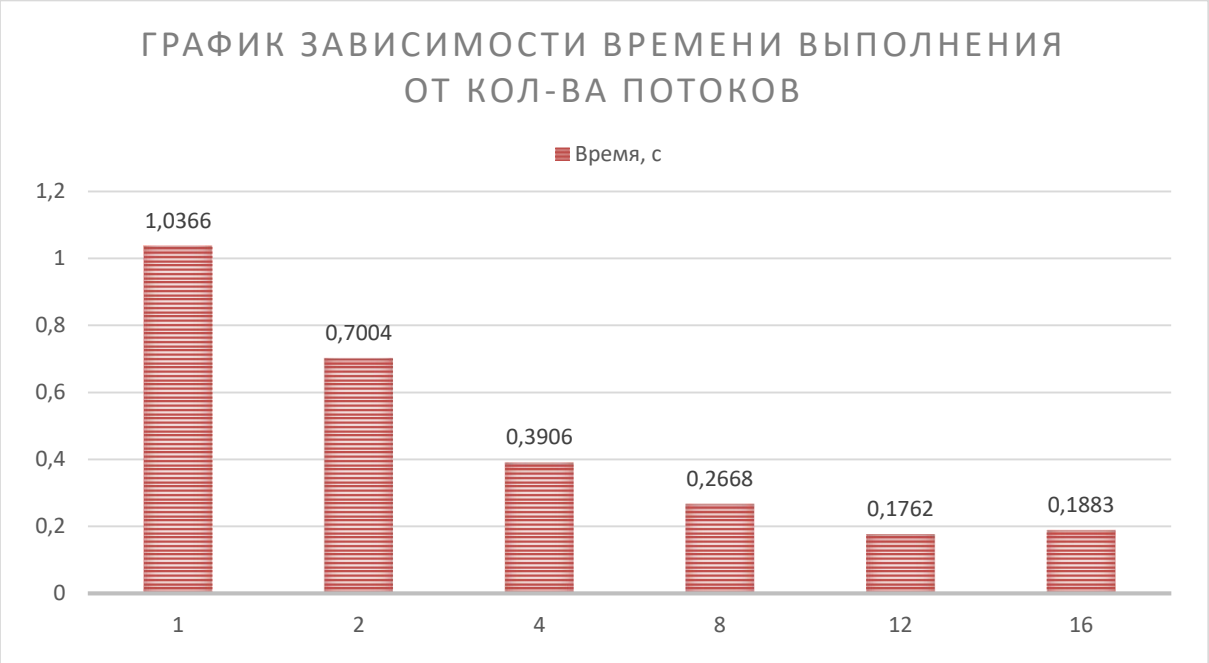

Рисунок 2. График зависимости времени выполнения от кол-ва потоков

Вывод: с использованием стандарта OpenMP прослеживается та же зависимость времени выполнения от кол-ва потоков: чем больше потоков, тем меньше время исполнения. Отклонения от результатов предыдущей работы минимальны.
Благодаря использованию этого стандарта программист может не заботиться о создании, протекании и удалении потоков: стандарт OpenMP сделает все за него, однако необходимо хорошо знать этот стандарт, так как отсутствующее или неправильное ключевое слово могут привести к краху программы при ее компиляции.