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

# ОТЧЕТ

**по лабораторной работе №1**

**по дисциплине «Параллельные алгоритмы»**

**Тема: Основы работы с процессами и потоками**

Студент гр. 0303 _____ Давыдов М. Д.

Преподаватель _____ Сергеева Е. И.

Санкт-Петербург

2023

**Цель работы.**

Изучить механизм работы процессов и потоков на примере языка C++.

**Задание.**

Лабораторная состоит из 3-х подзадач, которые выполняют одинаковую задачу с использованием процессов или потоков.

Выполнить умножение 2-х матриц.

Входные матрицы вводятся из файла (или генерируются).

Результат записывается в файл.

1.1.

Выполнить задачу, разбив её на 3 процесса. Выбрать механизм обмена данными между процессами.

Процесс 1: заполняет данными входные матрицы (читает из файла или генерирует их некоторым образом).

Процесс 2: выполняет умножение

Процесс 3: выводит результат

1.2.1

Аналогично 1.1, используя потоки (std::threads)

1.2.2

Разбить умножение на P потоков (“наивным” способом по по строкам-столбцам).

**Основные теоретические положения.**

Процесс — это идентифицируемая абстракция совокупности взаимосвязанных системных ресурсов на основе отдельного и независимого виртуального адресного пространства в контексте которой организуется выполнение потоков. Стандарт ISO 9000:2000 Definitions определяет процесс как совокупность взаимосвязанных и взаимодействующих действий, преобразующих входящие данные в исходящие.

Поток — наименьшая единица обработки, исполнение которой может быть назначено ядром операционной системы. Реализация потоков выполнения

и процессов в разных операционных системах отличается друг от друга, но в большинстве случаев поток выполнения находится внутри процесса.

**Выполнение работы.**

Выполнение пункта 1.1:

Для более удобного представления процессов в коде была реализована иерархия классов с родительским виртуальным классом Process и дочерними классами CalculateProcess, ReadProcess, WriteProcess. Во всех классах есть приватное поле m_sharedMemory для обращения к общему блоку памяти и обмена данными.

Также был написан класс SharedMemory, указатели на которые как раз хранятся в описанных выше классах.

Для более удобной отладки и логирования были написаны функции в файлах ErrorWriter.h ErrorWriter.cpp.

Для реализации процессов использовался функция fork(). Для корректного завершения всех процессов, логика возвращаемого fork() значения была обработана.

Процесс, отвечающий за чтение матриц, получает указатель на разделяемую память и считывает обе матрицы.

Процесс, отвечающий за умножение матриц, также получает указатель на разделяемую память, а также ожидает окончания работы процесса чтения. После чего выполняет умножение.

Процесс, занимающийся вывод, также ожидает выполнения предыдущих процессов, получает указатель на разделяемую память и выводит результирующую матрицу в файл.

Выполнение пункта 1.2:

Для выполнения данного пункты был реализован класс Threads с набором статических функций для работы потоков: чтения, вычисления, записи. Методы этого класс выполняют соответвующий названиям функционал, по аналогии с процессами.

Выполнение пункта 1.3:

Для выполнения пункта с разделением вычислений на P потоков методы класса Threads были модифицированы и теперь в нем в цикле создается P потоков и распределяются по строкам.

Ниже в таблице 1 представлена зависимость времени выполнения программы от размеров входных данных при фиксированном *P* количестве потоков:

| Время, мс | Размер матриц | Количество потоков |
|---|---|---|
| 0,011 | 10 x 10 | 4 |
| 0,079 | 100 x 100 | 4 |
| 0,440 | 100 x 1000 | 4 |

Таблица 1. Зависимость времени выполнения от размеров матриц

Из полученных результатов можно сделать вывод, что при увеличении размера матриц также увеличивает и время выполнения программы (что очевидно).

Далее, зависимость времени выполнения от количества потоков при фиксированных размерах матриц (см. табл. 2):

| Время, мс | Количество потоков | Размер матриц |
|---|---|---|
| 0,287 | 1 | 100 x 100 |
| 0,246 | 2 | 100 x 100 |
| 0,256 | 4 | 100 x 100 |
| 0,300 | 8 | 100 x 100 |
| 0,295 | 16 | 100 x 100 |
| 0,293 | 32 | 100 x 100 |

Таблица 2. Зависимость времени выполнения от количества потоков

Из таблицы видно, что оптимальное количество потоков в нашем случае при размере матриц 100 x 100 равно 2. При дальнейшем увеличении количества потоков время выполнения программы увеличивается так как переключение

между потоками занимает время, в то время как система не может предоставить столько физических потоков.

**Выводы.**

При выполнении данной лабораторной работы, мы познакомились с механизмами работы процессов и потоков на языке C++. Было исследовано, что при увеличении числа потоков не всегда уменьшается время выполнения программы, так как после определенного числа потоков, время выполнения программы постепенно увеличивается из-за того, переключение между большим количеством потоков не всегда дает выигрыш по времени.