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

Студентка гр. 9303 _____ Москаленко Е.М.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург

2022

Цель работы.

Изучить понятия “процесс” и “поток”, ознакомиться с ними на практике на примере UNIX-процессов и класс std::thread языка программирования C++.

Задание.

Задача:

Выполнить поэлементное сложение 2х матриц M*N

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).

Опционально: в этом процессе могут быть 2 потока ввода/генерации данных

Процесс 2: выполняет сложение

Процесс 3: выводит результат

1.2.1

Аналогично 1.1, используя потоки (threads)

1.2.2

Разбить сложение на P потоков.

Исследовать зависимость между количеством потоков, размерами входных данных и параметрами целевой вычислительной системы.

Выполнение работы.

1. **Процессы**

Для создания и работы новых процессов используется системный вызов Fork. Он создает дочерний процесс, который выполняется одновременно с процессом, вызывающим fork() (родительский процесс). После создания нового дочернего процесса оба процесса выполнят следующую инструкцию. Дочерний процесс использует тот же ПК (счетчик программ), те же регистры процессора, те же открытые файлы, которые используются в родительском процессе. Чтобы определить, какой процесс должен выполнять инструкцию, используем метод getpid() (0 - дочерний процесс, -1 - произошла ошибка и процесс не создался, иначе - родительский процесс и нужно ждать с помощью wait(), пока процесс-потомок не исполнит свою часть программы.

Всего в решении используется три процесса: для генерации матриц, для их сложения, для записи результата в файл.

2. Потоки. **Сложение матриц в один поток**

   Та же самая задача была решена с помощью потоков (класс std::thread). Для генерации начальных матриц использованы два потока. Чтобы подождать выполнение потока (или чтобы он не завершился позже главного потока) используется метод join. Каждый поток привязан к определенной функции и работает с копиями переданных аргументов. Если необходимо в функцию потока передать ссылку, используется std::ref.

3. Потоки. **Сложение матриц в несколько потоков**

   Теперь сложение матриц происходит в несколько потоков, а не в один. Алгоритм прогоняется несколько раз, используя разное количество потоков (от 2 до количества строк в матрице), и записывает результаты в файлы “threadsN”, где N - количество отработавших потоков. Каждый поток отвечает за определенный отрезок матрицы и заполняет эти несколько строк.

Исследуем зависимость между размерами входных данных (размер матрицы) и производительностью программы по времени, используя 3 потока для сложения матриц:

Размер матрицы N*M	**Время (микросекунды)**
5*5	228
10*10	280
50*50	309
100*100	414
250*250	1603
500*500	7520

Закономерность такова: чем больше размерность матрицы, тем больше времени нужно на выполнение программы.

Теперь исследуем зависимость между количеством исполняемых потоков и производительностью программы по времени, на примере размерности матриц 500*500:

Количество потоков	**Время (микросекунды)**
2	4418
4	4445
6	4196
79	12285
160	14887
213	10365
250	19741

500	24504

Из таблицы видно, что количество исполняемых потоков не является гарантом производительности по времени.

Выводы.

В данной лабораторной работе были изучены принципы работы с процессами и потоками на примере языка C++. Был проведен сравнительный анализ времени выполнения программы в зависимости от размера матриц и количества исполняемых потоков. Можно сделать вывод, что:

1) Чем больше размер матриц - тем дольше исполняется программа.
2) Большое количество потоков не всегда влияет на увеличение скорости выполнения программы.