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. 0381 _____ Ионина К.С.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2022

Цель работы.

Изучить работу с процессами и потоками на языке программирования C++.

Задание.

Выполнить поэлементное сложение 2x 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

Основные теоретические положения.

Процесс – это некоторая часть работы ОС, обладающая уникальным идентификационным номером – id, и адресное пространство. Адресное пространство – некоторый список адресов в памяти, с которыми происходит работа этого процесса. С другими адресами процессу приходится работать через системный вызов. Одна программа может включать как несколько процессов, так и один, причем последнее используется наиболее часто. Разбиение на процессы позволяет распараллелить задачи.

Для порождения процессов в ОС Linux существует два способа. В данной работе новый процесс создавался с помощью системного вызова fork(). При вызове fork() возникают два полностью идентичных процесса. Весь код после fork() выполняется дважды, как в процессе-потомке, так и в процессе-родителе.

Процесс-потомок и процесс-родитель получают разные коды возврата после вызова fork(). Процесс-родитель получает идентификатор (PID) потомка. Если это значение будет отрицательным, следовательно при порождении процесса произошла ошибка. Процесс-потомок получает в качестве кода возврата значение 0, если вызов fork() оказался успешным. Функция wait() ждет завершения первого из всех возможных потомков родительского процесса.

Поток – это часть самого процесса, выполняющая определенный список действий. У каждого процесса есть как минимум один поток, и их увеличение обеспечивает распараллеливание процесса. Сам поток представляет собой стек команд со счетчиком, обладающий несколькими важными свойствами, такими как состояние и приоритет. Состояний потока всего три: состояние активности, то есть поток выполняется на данный момент, состояние неактивности, когда поток ожидает выделения процессора для перехода в состояние активности, и третье – состояние блокировки, когда потоку не выделяется время (соответственно он не занимает место в очереди, освобождая ресурсы) вне зависимости от его приоритета.

Выполнение работы.

Был создан класс Matrix с методами, необходимыми для решения поставленной задачи.

Метод writeByPtr() – записывает матрицу в память.

Метод readByPtr() – из памяти записывает матрицу в класс.

Метод check_dimensions() – проверяет, возможно ли сложить матрицы.

operator+ - производит сложение матриц, если их сложить не удалось, выводит об этом сообщение.

1. Сложение матриц происходит по шагам. На первом шаге из файла input.txt. происходит считывание матриц. На следующем шаге матрицы

складываются поэлементно, далее результат сложения матриц записывается в файл output.txt.

С помощью функции fork() происходит создание процессов. По возвращаемому значению можно определить, в каком процессе выполняется код. Для передачи данных между процессами используется shared memory.

2. Для работы с потоками был подключен файл <thread>. Поток создается с помощью конструктора thread(). inputThread – это объект, представляющий поток, в котором будет выполняться функция readMatrix2(). Вызов join() блокирует вызывающий поток (main) до тех пор, пока inputThread (а точнее readMatrix2()) не выполнит свою работу. Параметры, передаваемые по ссылке обернуты в std::ref. Для получения возвращаемого значения из функции используется std::promise. Аналогично реализованы поток сложения sumThread и поток вывода результата сложения printThread.

3. Создание потоков выполнено аналогично п.2. В данном пункте мы замеряем время выполнения потока суммирования, чтобы исследовать зависимость времени сложения от количества потоков P.

Зависимость времени сложения от количества потоков P представлена в табл.1.

Количество потоков	**Время (мс)**	**Размер матрицы**
1	271	10*10
2	227	10*10
3	653	10*10
4	236	10*10
5	357	10*10
6	2206	10*10
7	3561	10*10
8	1469	10*10

Таблица 1 – Зависимость времени сложения от количества потоков P.

Зависимость времени сложения от размера матриц при сложении с помощью 8 потоков представлена в табл.2.

Количество потоков	Время (мс)	Размер матрицы
8	405	5*5
8	547	10*10
8	1576	100*100
8	11038	1000*1000
8	779119	10000*10000

Таблица 2 – Зависимость времени сложения размера матриц.

Выводы.

В ходе выполнения лабораторной работы была изучена работа с процессами и потоками на языке программирования C++.

Был изучен дополнительный теоретический материал, а именно, std::promise, std::future.

Было исследовано, что размер матриц влияет на время выполнения операции сложения. Чем больше размер матрицы, тем дольше сложение. Также увеличение числа потоков не всегда ведет к сокращению времени работы программы.

Были решены три подзадачи, для каждый из которых написан код.

1) Разбиение чтения, сложения и вывода матриц на 3 процесса;
2) Разбиение чтения, сложения и вывода матриц на 3 потока;
3) Разбиение сложения на P потоков.