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. 0304 _____ Говорющенко А.В.

Преподаватель _____ Сергеева Е.И.

Санкт-Петербург
2023

Цель работы.

Изучить основы работы с процессами и потоками

Задание.

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

Выполнить умножение 2х 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).

Выполнение работы.

1. Умножение матриц с помощью 3 процессов.

Для работы с матрицами был создан класс Matrix, хранящий количество рядов и столбцов, а также саму матрицу, и реализующий весь необходимый функционал.

Для межпроцессного взаимодействия были использованы сокеты, позволяющие запускать процессы независимо друг от друга, и написан класс Socket.

```
class Socket {
    public:
        Socket(int fd = -1);
        Socket(const std::string& fileName);
        Socket(Socket&& other);
        virtual ~Socket();

        bool valid() const;
        bool write(const int *data, int count) const;
        bool read(int *data, int count) const;

    protected:
        int _fd;
};
```

Для применения UNIX domain-сокетов был написан класс ServerSocket, который позволяет устанавливать соединение с помощью файла сокета.

```
class ServerSocket: public Socket  {
    public:
        ServerSocket(const std::string fileName);
        ~ServerSocket();

        Socket accept();

    private:
        struct sockaddr_un _addr;
};
```

Для выполнения умножения матриц было создано 3 программы: reader, executor и writer, которые обмениваются данными по сокетам /tmp/readSocket и /tmp/writeSocket.

reader является сервером на сокете /tmp/readSocket, в нем генерируются случайные матрицы и отправляются по сокету. executor является одновременно и клиентом на /tmp/readSocket, откуда получает сгенерированные матрицы, и сервером на сокете /tmp/writeSocket, по которому передает результат умножения матриц. writer выводит полученные перемноженные матрицы и является клиентом на сокете /tmp/writeSocket.

2. Умножение матриц с помощью 3 потоков.

Для умножения матриц в 3 потока была написана программа threadProg, в которой создаются потоки reader, executor и writer. Для синхронизации потоков используется join(), блокирующий вызывающий поток.

3. Умножение матриц с помощью P потоков.

Для умножения матриц с использованием P потоков была создана программа pThreadProg, в которой поток execute запускал P потоков, которые считали определенные элементы результирующий матрицы.

4. Измерение производительности.

Измерение производительности осуществлялось с помощью команды time в bash. Было измерено реальное, системное и пользовательское время при 100 запусках программы.

Таблица 1. Умножение матриц с помощью 3 процессов.

Размер матриц	User time, c	System time, c	Real time, c
10x10	0.618	0.363	0.528
50x50	1.309	0.493	1.755
100x100	3.502	0.762	4.631
150x150	8.962	0.977	10.564

Таблица 2. Умножение матриц с помощью 3 потоков.

Размер матриц	User time, с	System time, с	Real time, с
10x10	0.206	0.115	0.311
50x50	0.796	0.195	1.137
100x100	3.195	0.210	4.085
150x150	7.972	0.336	8.372

По результатам двух таблиц видно, что умножение матриц разбиением на потоки требует меньше времени, чем разбиением на процессы. Это связано с тем, что создание процессов более тяжелый процесс для системы, чем создание потоков.

Таблица 3. Умножение матриц с помощью P потоков.

Размер матриц	Кол-во потоков	Real time, с
50x50	2	0.971
	4	1.876
	8	1.279
	16	1.037
	32	1.198
	48	1.315
	64	1.418
150x150	2	6.119
	4	5.358
	8	5.291
	16	7.857
	32	13.204
	48	19.015
	64	24.490

По результатам таблицы видно, что наименьшее время выполнения достигается при количестве потоков равном количеству процессоров системы —

8. Если это количество меньше, то часть процессоров не заняты, а в случае превышения количества процессоров, последние вынуждены переключаться между потоками, тратя на это дополнительное время.

Выводы.

В ходе работы были изучены процессы и потоки, а также исследовано время выполнения умножения матриц в зависимости от количества потоков и проведено сравнение потоков и процессов.