Московский Авиационный Институт

(Национальный Исследовательский Университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

**Лабораторная работа №3 по курсу**

**«Операционные системы»**

**Тема работы**

**“Потоки”**

Студент: Меджидли Махмуд Ибрагим оглы

Группа: М8О-208Б-20

Вариант: 13

Преподаватель: Миронов Евгений Сергеевич

Оценка: \_\_\_\_\_\_\_\_\_\_\_

Дата: \_\_\_\_\_\_\_\_\_\_\_

Подпись: \_\_\_\_\_\_\_\_\_\_\_

Москва, 2022

**Содержание**

1. Репозиторий
2. Постановка задачи
3. Общие сведения о программе
4. Общий метод и алгоритм решения
5. Исходный код
6. Демонстрация работы программы
7. Выводы

**Репозиторий**

https://github.com/loshadkaigogo/OS

**Постановка задачи**

Задача: реализовать задачу следующего рода. 128-битные числа в шестнадцатеричном представлении хранятся в файле. Необходимо вычислить их среднее арифметическое. Количество оперативной памяти подаётся.

**Метод решения**

**Используются следующие системные вызовы:**

1. **thread() –** создает новый поток выполнения в программе.
2. **thread.join()** – для блокировки потока.

**Общие сведения о программе**

Для реализации поставленной задачи нам нужны следующие библиотеки: <unistd.h> - для работы с системными вызовами в Linux.

<limits.h> - для определения характеристик общих типов переменных.

<stdlib.h> - для того, чтобы можно было пользоваться функциями, отвечающими за работу с памятью.

<time.h> - для функций, работающих со временем (нужно для строчки srand(time(NULL)) - для генерации рандомных чисел с использованием текущего времени).  
<pthread.h> - для работы с потоками.

<ctype.h> - для классификации и преобразования отдельных символов.

<sys/stat.h> - для доступа к файлам.

<fcntl.h> - для работы с файловым дескриптором.

<inttypes.h> - макросы для использования с функциями printf и scanf.

<string.h> - для использования функций над строками.

Для работы с потоками согласно заданию помимо библиотеки <pthreads.h> я использую такие системные вызовы, как pthread\_create, отвечающий за создание потока, имеющий тип возвращаемого значения int и принимающий 4 аргумента: указатель на поток, атрибуты по умолчанию, указатель на нужную нам функцию и аргументы), а также pthread\_join, отвечающий  
за ожидание завершения потока, имеющий тип возвращаемого значения int и принимающий 2 аргумента: указатель на поток и указатель на указатель в качестве аргумента для хранения возвращаемого значения).

Помимо системных вызовов, связанных с потоками, в моей программе имеются следующие системные вызовы:  
off\_t lseek(...) - устанавливает смещение для файлового дескриптора в значение аргумента offset.

int open(...) - открытие файлового дескриптора.

void exit(...) - выход из процесса с заданным статусом.

int close(...) - закрытие файлового дескриптора.

int write(...) - записывает количество байтов в 3 аргументе из буфера в файл с дескриптором fd.  
Программа собирается и запускается при помощи следующих команд:

gcc lab3.c -pthread -o main

./main thread\_number memory\_amount (пример: ./main 1 72).

**Общий метод и алгоритм решения**

Я создаю две структуры. Первая структура struct Command отвечает за пользовательские данные: количество подающейся оперативной памяти и количество потоков. Вторая стуктура struct Params хранит в себе параметры для одного потока. С запуском программы в файле генерируются числа, и после генерации каждый поток инициализируется начальными данными в функции void initialization(...), принимающей три аргумента: указатель на пользовательскую структуру, указатель на структуру одного потока и количество чисел. Эта же функция распределяет числа по потокам. Далее каждый поток считает свою локальную сумму, в конце работы программы локальные суммы складываются и подсчитывается среднее арифметическое.

**Анализ скорости и эффективности**

Рассмотрим сначала 10000 чисел.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество потоков **p** | Время сортировки с одним потоком **T1** (мкс) | Сортировка с **p** потоками **Tp** (мкс) | Ускорение  (**Sp**=**T1/Tp**) | Эффективность  (**Xp**=Sp/p) |
| 1 | 115504 | 115504 | 1 | 1 |
| 2 | 115504 | 87468 | 1.32 | 0.66 |
| 3 | 115504 | 79805 | 1.45 | 0.48 |
| 4 | 115504 | 75855 | 1.52 | 0.38 |
| 5 | 115504 | 81048 | 1.43 | 0.29 |
| 6 | 115504 | 69663 | 1.66 | 0.28 |
| 7 | 115504 | 76898 | 1.5 | 0.21 |
| 8 | 115504 | 83455 | 1.38 | 0.17 |
| 9 | 115504 | 74030 | 1.56 | 0.17 |
| 10 | 115504 | 65547 | 1.76 | 0.18 |

Здесь видно, что эффективность понижается с увеличением числа потоков, но плохо просматривается закономерность в ускорении.

Возьмём массив из 50000 тысяч чисел.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество потоков **p** | Время сортировки с одним потоком **T1** (мкс) | Время сортировки с одним потоком **T1** (мкс) | Ускорение  (**Sp**=**T1/Tp**) | Эффективность  (**Xp**=Sp/p) |
| 1 | 590869 | 590869 | 1 | 1 |
| 2 | 590869 | 299094 | 1,98 | 0.99 |
| 3 | 590869 | 364958 | 1,62 | 0.54 |
| 4 | 590869 | 366851 | 1,61 | 0.40 |
| 5 | 590869 | 353109 | 1,67 | 0.33 |
| 6 | 590869 | 310636 | 1.90 | 0.32 |
| 7 | 590869 | 395236 | 1.49 | 0.21 |
| 8 | 590869 | 302383 | 1.95 | 0.24 |
| 9 | 590869 | 337434 | 1.75 | 0.19 |
| 10 | 590869 | 375124 | 1.58 | 0.16 |
| 100 | 590869 | 280406 | 2.11 | 0.02 |

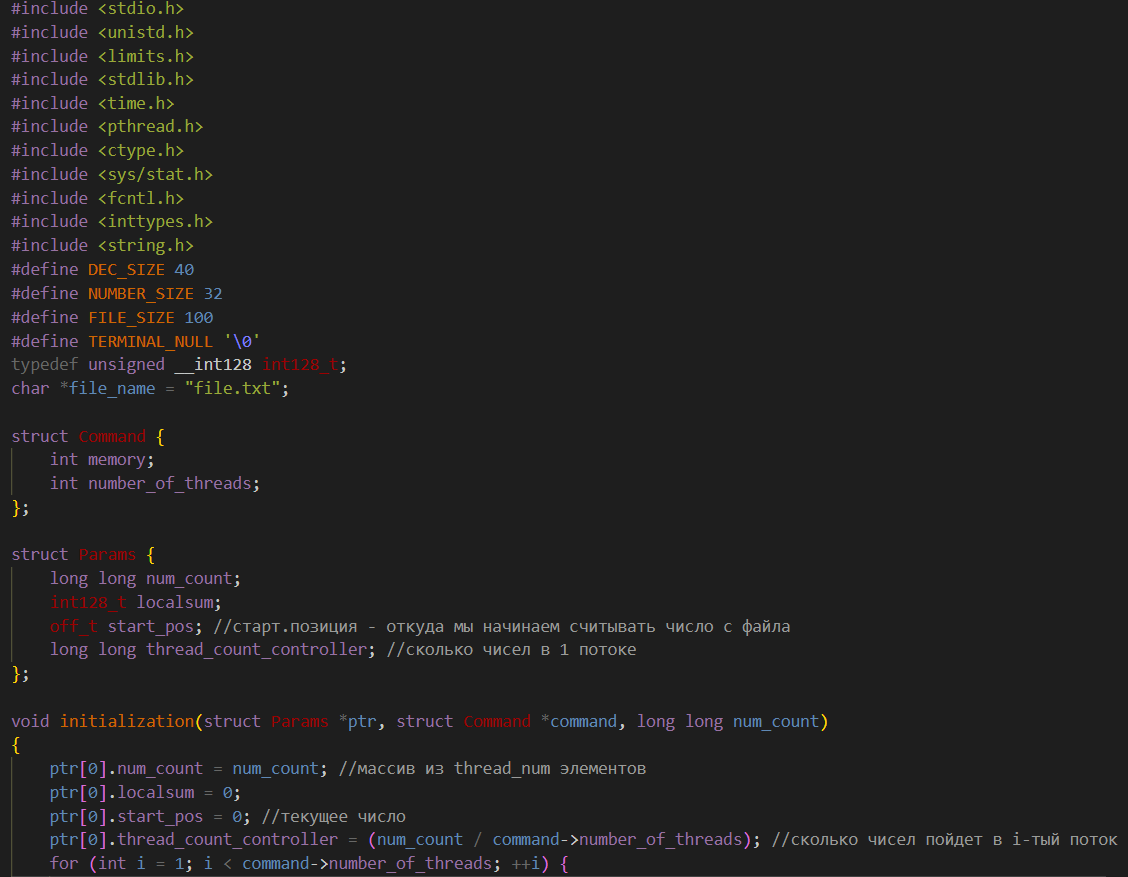
Здесь видно, что при большом количестве потоков ускорение падает, и всё же погрешность ещё велика.

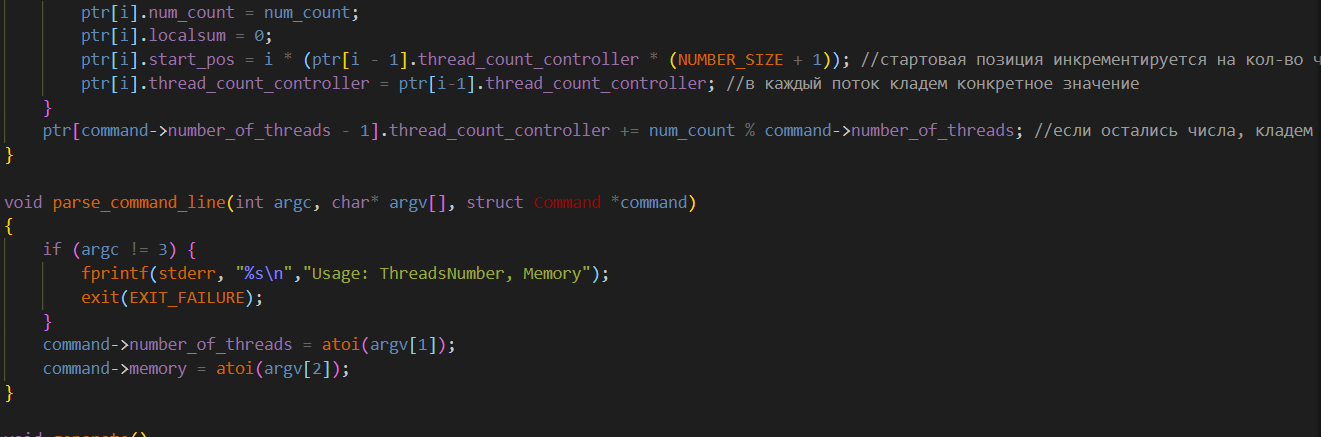
Последней возьмём последовательность из 100000 чисел.

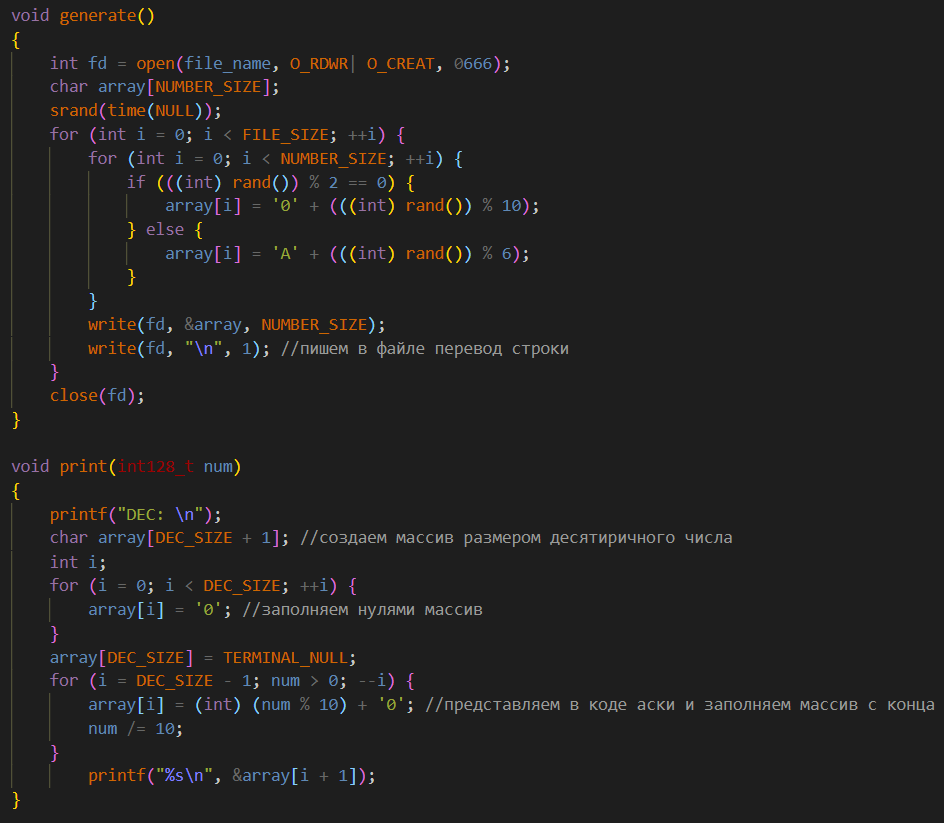
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество потоков **p** | Время сортировки с одним потоком **T1** (мкс) | Сортировка с **p** потоками **Tp** (мкс) | Ускорение  (**Sp**=**T1/Tp**) | Эффективность  (**Xp**=Sp/p) |
| 1 | 960749 | 960748 | 1 | 1 |
| 2 | 960748 | 717900 | 1,34 | 0,67 |
| 3 | 960748 | 897206 | 1,07 | 0,36 |
| 4 | 960748 | 663090 | 1,45 | 0,36 |
| 5 | 960748 | 743881 | 1,29 | 0,25 |
| 6 | 960748 | 588472 | 1,63 | 0,27 |
| 7 | 960748 | 663981 | 1,45 | 0,21 |
| 8 | 960748 | 659101 | 1,46 | 0,18 |
| 9 | 960748 | 694577 | 1,38 | 0,15 |
| 10 | 960748 | 668109 | 1,44 | 0,14 |
| 15 | 960748 | 813362 | 1,18 | 0,08 |
| 1000 | 960748 | 783225 | 1,23 | 0,001 |
| 10000 | 960748 | 779213 | 1,23 | 0,0001 |

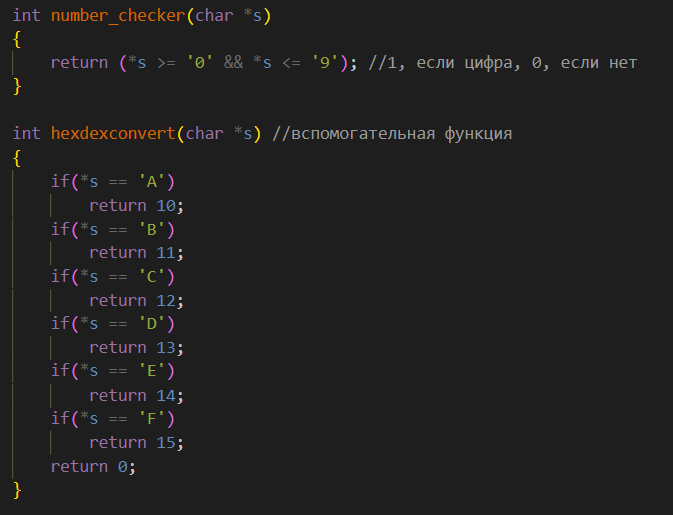
Как мы видим, при большом количестве потоков эффективность сильно снижается. На большом количестве данных не очень большое ускорение.

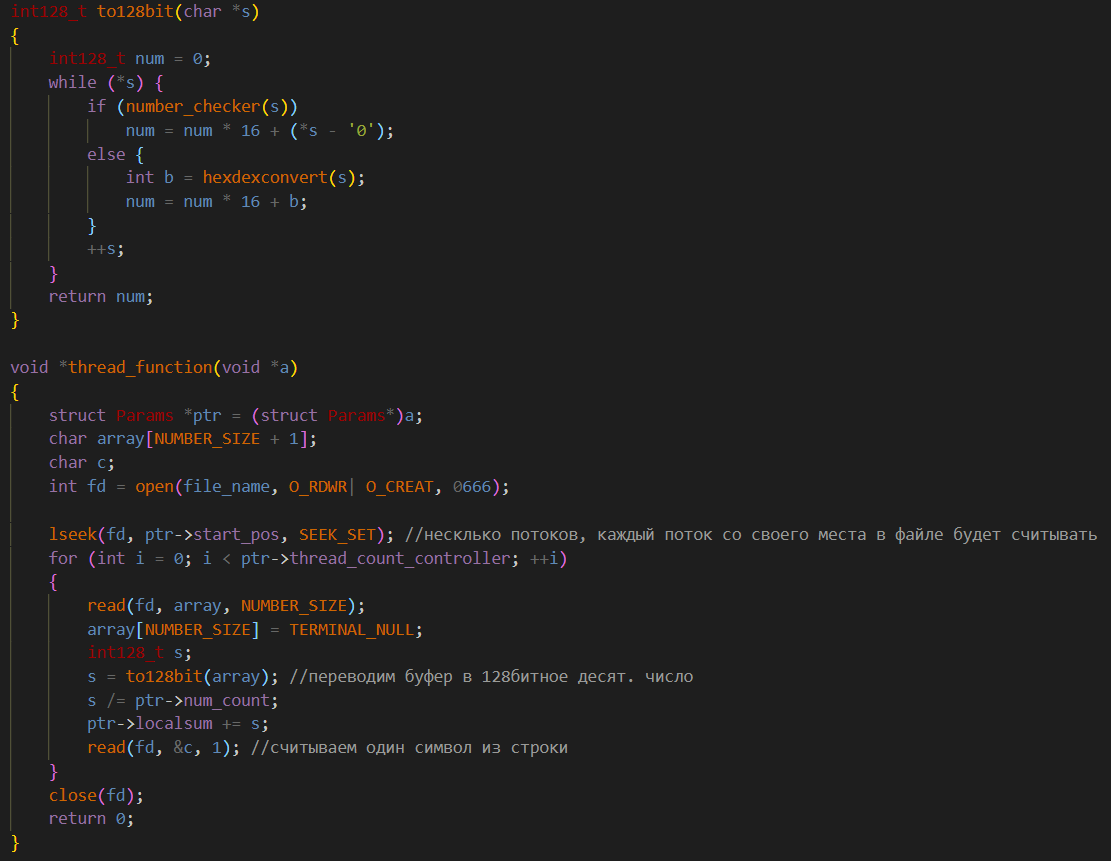
**Исходный код**

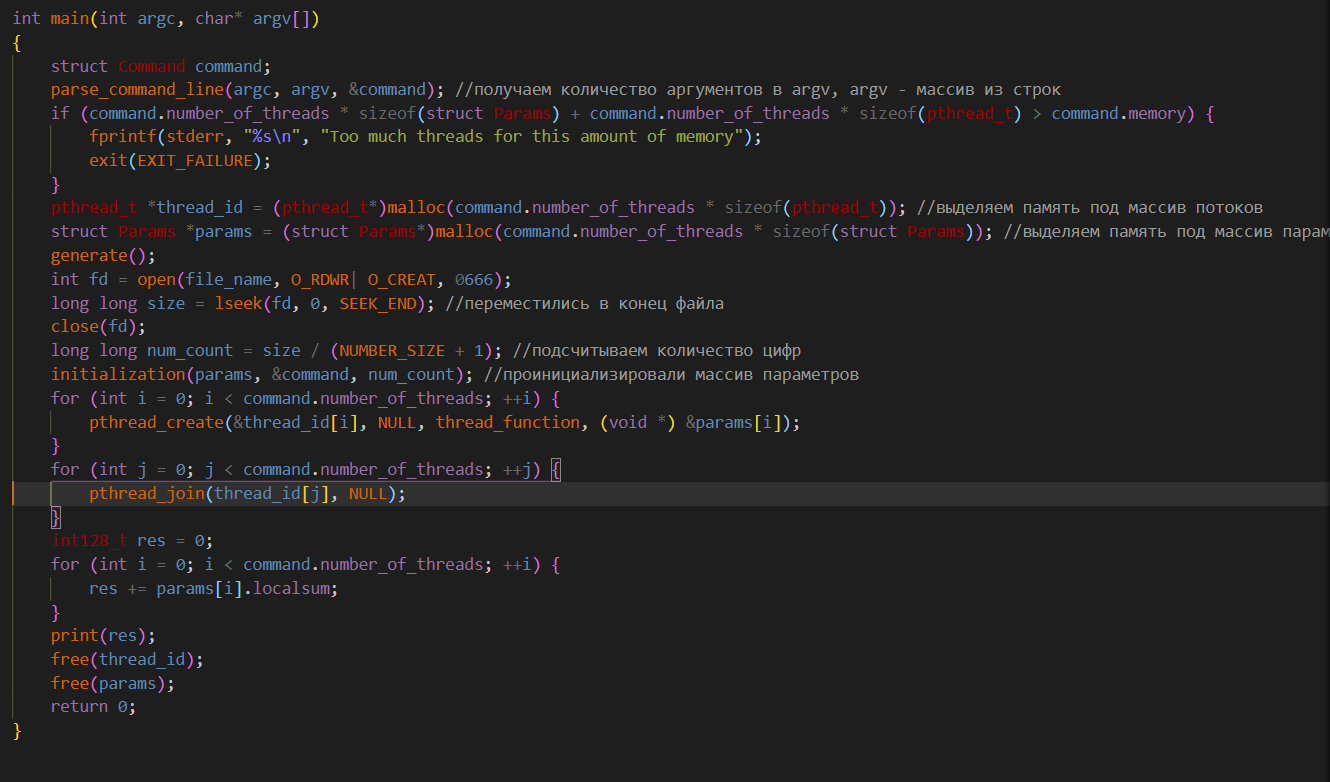
****

****

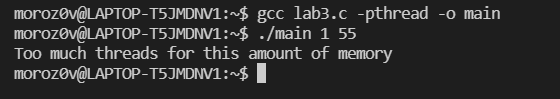
****

****

****

**  
  
  
  
Демонстрация работы программы**Тест 1.

gcc lab3.c –pthread -o main

./main 1 55 ****Тест 2.  
./main 1 56  


Тест 3.

./main 10 5600



**Выводы**Данная лабораторная работа помогла мне успешно ознакомиться с тем, как устроены потоки в Linux. Во время выполнения своего задания я изучил особенности системных вызовов и узнал многие тонкости работы с потоками.