

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 строки `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 <threads.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 структура `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 Tr (мкс)	Ускорение (Sp=T1/Tr)	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/Tr)	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 Tr (мкс)	Ускорение ($S_p = T1/Tr$)	Эффективность ($X_p = Sp/p$)
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

```
#include <stdio.h>
#include <unistd.h>
#include <limits.h>
#include <stdlib.h>
#include <time.h>
#include <pthread.h>
#include <ctype.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <inttypes.h>
#include <string.h>
#define DEC_SIZE 40
#define NUMBER_SIZE 32
#define FILE_SIZE 100
#define TERMINAL_NULL '\0'
typedef unsigned __int128 int128_t;
char *file_name = "file.txt";

struct Command {
    int memory;
    int number_of_threads;
};

struct Params {
    long long num_count;
    int128_t localsum;
    off_t start_pos; //старт.позиция - откуда мы начинаем считывать число с файла
    long long thread_count_controller; //сколько чисел в 1 потоке
};

void initialization(struct Params *ptr, struct Command *command, long long num_count)
{
    ptr[0].num_count = num_count; //массив из thread_num элементов
    ptr[0].localsum = 0;
    ptr[0].start_pos = 0; //текущее число
    ptr[0].thread_count_controller = (num_count / command->number_of_threads); //сколько чисел пойдет в i-тый поток
    for (int i = 1; i < command->number_of_threads; ++i) {
        ptr[i].num_count = num_count;
        ptr[i].localsum = 0;
        ptr[i].start_pos = i * (ptr[i - 1].thread_count_controller * (NUMBER_SIZE + 1)); //стартовая позиция инкрементируется на кол-во чисел
        ptr[i].thread_count_controller = ptr[i - 1].thread_count_controller; //в каждый поток кладем конкретное значение
    }
    ptr[command->number_of_threads - 1].thread_count_controller += num_count % command->number_of_threads; //если остались числа, кладем их в последний поток
}

void parse_command_line(int argc, char* argv[], struct Command *command)
{
    if (argc != 3) {
        fprintf(stderr, "%s\n", "Usage: ThreadsNumber, Memory");
        exit(EXIT_FAILURE);
    }
    command->number_of_threads = atoi(argv[1]);
    command->memory = atoi(argv[2]);
}
```

```

void generate()
{
    int fd = open(file_name, O_RDWR | O_CREAT, 0666);
    char array[NUMBER_SIZE];
    srand(time(NULL));
    for (int i = 0; i < FILE_SIZE; ++i) {
        for (int i = 0; i < NUMBER_SIZE; ++i) {
            if (((int) rand()) % 2 == 0) {
                array[i] = '0' + (((int) rand()) % 10);
            } else {
                array[i] = 'A' + (((int) rand()) % 6);
            }
        }
        write(fd, &array, NUMBER_SIZE);
        write(fd, "\n", 1); //пишем в файле перевод строки
    }
    close(fd);
}

void print(int128_t num)
{
    printf("DEC: \n");
    char array[DEC_SIZE + 1]; //создаем массив размером десятиричного числа
    int i;
    for (i = 0; i < DEC_SIZE; ++i) {
        array[i] = '0'; //заполняем нулями массив
    }
    array[DEC_SIZE] = TERMINAL_NULL;
    for (i = DEC_SIZE - 1; num > 0; --i) {
        array[i] = (int) (num % 10) + '0'; //представляем в коде аски и заполняем массив с конца
        num /= 10;
    }
    printf("%s\n", &array[i + 1]);
}

```

```

int number_checker(char *s)
{
    return (*s >= '0' && *s <= '9'); //1, если цифра, 0, если нет
}

int hexdexconvert(char *s) //вспомогательная функция
{
    if(*s == 'A')
        return 10;
    if(*s == 'B')
        return 11;
    if(*s == 'C')
        return 12;
    if(*s == 'D')
        return 13;
    if(*s == 'E')
        return 14;
    if(*s == 'F')
        return 15;
    return 0;
}

```



```

int128_t to128bit(char *s)
{
    int128_t num = 0;
    while (*s) {
        if (number_checker(s))
            num = num * 16 + (*s - '0');
        else {
            int b = hexdexconvert(s);
            num = num * 16 + b;
        }
        ++s;
    }
    return num;
}

void *thread_function(void *a)
{
    struct Params *ptr = (struct Params*)a;
    char array[NUMBER_SIZE + 1];
    char c;
    int fd = open(file_name, O_RDWR | O_CREAT, 0666);

    lseek(fd, ptr->start_pos, SEEK_SET); //несклько потоков, каждый поток со своего места в файле будет считывать
    for (int i = 0; i < ptr->thread_count_controller; ++i)
    {
        read(fd, array, NUMBER_SIZE);
        array[NUMBER_SIZE] = TERMINAL_NULL;
        int128_t s;
        s = to128bit(array); //переводим буфер в 128битное десят. число
        s /= ptr->num_count;
        ptr->localsum += s;
        read(fd, &c, 1); //считываем один символ из строки
    }
    close(fd);
    return 0;
}

```

```

int main(int argc, char* argv[])
{
    struct Command command;
    parse_command_line(argc, argv, &command); //получаем количество аргументов в argv, argv - массив из строк
    if (command.number_of_threads * sizeof(struct Params) + command.number_of_threads * sizeof(pthread_t) > command.memory) {
        fprintf(stderr, "%s\n", "Too much threads for this amount of memory");
        exit(EXIT_FAILURE);
    }
    pthread_t *thread_id = (pthread_t*)malloc(command.number_of_threads * sizeof(pthread_t)); //выделяем память под массив потоков
    struct Params *params = (struct Params*)malloc(command.number_of_threads * sizeof(struct Params)); //выделяем память под массив параметров
    generate();
    int fd = open(file_name, O_RDWR | O_CREAT, 0666);
    long long size = lseek(fd, 0, SEEK_END); //переместились в конец файла
    close(fd);
    long long num_count = size / (NUMBER_SIZE + 1); //подсчитываем количество цифр
    initialization(params, &command, num_count); //проинициализировали массив параметров
    for (int i = 0; i < command.number_of_threads; ++i) {
        pthread_create(&thread_id[i], NULL, thread_function, (void *) &params[i]);
    }
    for (int j = 0; j < command.number_of_threads; ++j) {
        pthread_join(thread_id[j], NULL);
    }
    int128_t res = 0;
    for (int i = 0; i < command.number_of_threads; ++i) {
        res += params[i].localsum;
    }
    print(res);
    free(thread_id);
    free(params);
    return 0;
}

```

Демонстрация работы программы

Тест 1.

```
gcc lab3.c -pthread -o main
```

```
./main 1 55
```

```
Too much threads for this amount of memory
```

Тест 2.

```
./main 1 56
```

```
DEC:
```

```
207416305755511745509774597726159956578
```

Тест 3.

```
./main 10 5600
```

```
DEC:
```

```
215179883814826999064480751907889506549
```

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.