Московский Авиационный Институт

(Национальный Исследовательский Университет)

Институт №8 “Компьютерные науки и прикладная математика”

Кафедра №806 “Вычислительная математика и программирование”

Лабораторная работа №2 по курсу

«Операционные системы»

Группа: М8О-210Б-23

Студент: Шведов Александр Иванович

Преподаватель: Бахарев В.Д. (ФИИТ)

Оценка: _____

Дата: 02.12.24

Москва, 2024

Постановка задачи

Вариант 15.

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработки использовать стандартные средства создания потоков операционной системы (Windows/Unix). Ограничение максимального количества потоков, работающих в один момент времени, должно быть задано ключом запуска вашей программы.

Есть набор 128 битных чисел, записанных в шестнадцатеричном представлении, хранящихся в файле. Необходимо посчитать их среднее арифметическое. Округлить результат до целых. Количество используемой оперативной памяти должно задаваться "ключом"

Общий метод и алгоритм решения

Использованные системные вызовы:

- int pthread_create(pthread_t *_thread_, const pthread_attr_t *_attr_, void *(*_start_routine_)(void*), void *_arg_);- создание потока.
- int pthread_mutex_lock(pthread_mutex_t *_mutex_); - блокировка мьютекса
- int pthread_mutex_unlock(pthread_mutex_t *_mutex_); - разблокировка мьютекса
- int pthread_cond_signal(pthread_cond_t *cond) - отправляет сигнал с помощью условной переменной
- int pthread_cond_wait(pthread_cond_t *restrict cond, pthread_mutex_t *restrict mutex) – функция, которая переводит поток в ожидающее состояние, пока не будет получен сигнал с помощью условной переменной.
- void exit(int code) – завершение программы.
- int pthread_detach(pthread_t thread) - "Отсоединяет" поток от родительского потока.

Программа предназначена для вычисления среднего арифметического набора 128-битных чисел, представленных в шестнадцатеричной форме, которые считываются из файла. Для этого используются многозадачность и обработка данных в потоках с ограничением по памяти. Основные этапы работы программы:

1. Чтение данных: Программа открывает файл и читает числа блоками, размеры которых определяются с учетом размера заданной оперативной памяти.
2. Параллельная обработка данных: Данные из файла считываются блоками. Размер блока определяется на основе ограничений по памяти. Для каждого блока создается поток, который вычисляет сумму чисел в этом блоке.
3. Использование многозадачности: Для управления количеством одновременно работающих потоков используется мьютекс и условная переменная. Потоки синхронизируются с главной программой, чтобы не возникало гонок за ресурсами, и все вычисления завершались корректно.
4. Вычисление среднего арифметического: После завершения работы всех потоков главная программа собирает результаты (сумму всех чисел) и делит на количество чисел, чтобы вычислить среднее арифметическое. Результат выводится на экран.

Код программы

Lab.c

```c
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <fcntl.h>
#include <unistd.h>
#include <pthread.h>
```

```c
#include <time.h>
#include <stdio.h> // Для snprintf

#define BUFFER_SIZE 128

uint64_t sum = 0;
size_t totalCount = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
int activeThreads = 0;
pthread_cond_t cond = PTHREAD_COND_INITIALIZER;
size_t maxThreads;
size_t memoryLimit;

typedef struct {
    uint64_t *numbers;
    size_t count;
} ThreadData;

void *processChunk(void *arg) {
    ThreadData *data = (ThreadData *)arg;
    uint64_t localSum = 0;

    for (size_t i = 0; i < data->count; i++) {
        localSum += data->numbers[i];
    }

    pthread_mutex_lock(&mutex);
    sum += localSum;
    totalCount += data->count;
    activeThreads--;
    pthread_cond_signal(&cond);
    pthread_mutex_unlock(&mutex);

    free(data->numbers);
    free(data);
    return NULL;
}

void createThreads(uint64_t *numbers, size_t count) {
    ThreadData *data = malloc(sizeof(ThreadData));
    data->numbers = numbers;
    data->count = count;

    pthread_t thread;
    pthread_mutex_lock(&mutex);
    while (activeThreads >= maxThreads) {
        pthread_cond_wait(&cond, &mutex);
    }
    activeThreads++;
    pthread_mutex_unlock(&mutex);

    if (pthread_create(&thread, NULL, processChunk, data) != 0) {
        exit(1);
    }
```

```
    pthread_detach(thread);
}


int main(int argc, char *argv[]) {
    if (argc != 4) {
        const char *msg = "Использование: <имя файла> <макс. потоки> <память>\n";
        write(STDERR_FILENO, msg, strlen(msg));
        exit(1);
    }
    clock_t start = clock();

    const char *filename = argv[1];
    maxThreads = strtoul(argv[2], NULL, 10);
    memoryLimit = strtoul(argv[3], NULL, 10);

    int file = open(filename, O_RDONLY);
    if (file == -1) {
        const char *msg = "Ошибка открытия файла\n";
        write(STDERR_FILENO, msg, strlen(msg));
        exit(1);
    }

    size_t numbersPerChunk = memoryLimit / sizeof(uint64_t);
    char buffer[BUFFER_SIZE];
    size_t bufferIndex = 0;
    uint64_t *numbers = malloc(numbersPerChunk * sizeof(uint64_t));
    size_t count = 0;

    while (1) {
        ssize_t bytesRead = read(file, buffer + bufferIndex, BUFFER_SIZE - bufferIndex);
        if (bytesRead <= 0) break;

        bytesRead += bufferIndex;
        size_t start = 0;

        for (size_t i = 0; i < bytesRead; i++) {
            if (buffer[i] == '\n') {
                buffer[i] = '\0';
                numbers[count++] = strtoull(buffer + start, NULL, 16);
                start = i + 1;

                if (count == numbersPerChunk) {
                    createThreads(numbers, count);
                    numbers = malloc(numbersPerChunk * sizeof(uint64_t));
                    count = 0;
                }
            }
        }

        bufferIndex = bytesRead - start;
        memmove(buffer, buffer + start, bufferIndex);
    }
```

```
    if (count > 0) {
        createThreads(numbers, count);
    } else {
        free(numbers);
    }

    pthread_mutex_lock(&mutex);
    while (activeThreads > 0) {
        pthread_cond_wait(&cond, &mutex);
    }
    pthread_mutex_unlock(&mutex);

    close(file);

    uint64_t average = (totalCount > 0) ? (sum / totalCount) : 0;

    char result[64];
    snprintf(result, sizeof(result), "%lu\n", average);
    write(STDOUT_FILENO, "Среднее арифметическое: ", 45);
    write(STDOUT_FILENO, result, strlen(result));

    clock_t end = clock();
    char time[64];
    float seconds = (float)(end - start) / CLOCKS_PER_SEC;
    snprintf(time, sizeof(time), "%lf\n", seconds);
    write(STDOUT_FILENO, time, strlen(time));

    return 0;
}
```

Протокол работы программы

Тестирование:

```
alshved@HPal:/mnt/c/Users/mrshv/OneDrive/Desktop/Os/2$ ./a.out test.txt 1 2048
Среднее арифметическое: 1373343066833646
0.064835
```

```
alshved@HPal:/mnt/c/Users/mrshv/OneDrive/Desktop/Os/2$ ./a.out test.txt 2 2048
Среднее арифметическое: 1373343066833646
0.060780
```

```
alshved@HPal:/mnt/c/Users/mrshv/OneDrive/Desktop/Os/2$ ./a.out test.txt 3 2048
Среднее арифметическое: 1373343066833646
0.057580
```

```
alshved@HPal:/mnt/c/Users/mrshv/OneDrive/Desktop/Os/2$ ./a.out test.txt 5 2048
Среднее арифметическое: 1373343066833646
0.055573
```

```
alshved@HPal:/mnt/c/Users/mrshv/OneDrive/Desktop/Os/2$ ./a.out test.txt 6 2048
Среднее арифметическое: 1373343066833646
0.046916
```

Strace:

execve("./a.out", ["./a.out", "test1.txt", "2", "256"], 0x7ffd6e420f28 /* 28 vars */) = 0

```
brk(NULL)                               = 0x55c8a2a78000

arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc0845e120) = -1 EINVAL (Invalid argument)

mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fe2f44a9000

access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (No such file or directory)

openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3

newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=37379, ...}, AT_EMPTY_PATH) = 0

mmap(NULL, 37379, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7fe2f449f000

close(3)                                = 0

openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3

read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0P\237\2\0\0\0\0"..., 832) = 832

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

pread64(3, "\4\0\0\0 \0\0\0\5\0\0\0GNU\0\2\0\0\300\4\0\0\0\3\0\0\0\0\0\0\0"..., 48, 848) = 48

pread64(3, "\4\0\0\0\24\0\0\0\3\0\0\0GNU\0I\17\357\204\3$\f\221\2039x\324\224\323\236S"..., 68, 896) =
68

newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2220400, ...}, AT_EMPTY_PATH) = 0

pread64(3, "\6\0\0\0\4\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0@\0\0\0\0\0\0\0"..., 784, 64) = 784

mmap(NULL, 2264656, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7fe2f4276000

mprotect(0x7fe2f429e000, 2023424, PROT_NONE) = 0

mmap(0x7fe2f429e000, 1658880, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7fe2f429e000

mmap(0x7fe2f4433000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7fe2f4433000

mmap(0x7fe2f448c000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x215000) = 0x7fe2f448c000

mmap(0x7fe2f4492000, 52816, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7fe2f4492000

close(3)                                = 0

mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7fe2f4273000

arch_prctl(ARCH_SET_FS, 0x7fe2f4273740) = 0

set_tid_address(0x7fe2f4273a10)         = 15003

set_robust_list(0x7fe2f4273a20, 24)     = 0

rseq(0x7fe2f42740e0, 0x20, 0, 0x53053053) = 0
```

mprotect(0x7fe2f448c000, 16384, PROT_READ) = 0

mprotect(0x55c883426000, 4096, PROT_READ) = 0

mprotect(0x7fe2f44e3000, 8192, PROT_READ) = 0

prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0

munmap(0x7fe2f449f000, 37379)   = 0

clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=2887400}) = 0

openat(AT_FDCWD, "test1.txt", O_RDONLY) = 3

getrandom("\xa3\x04\xe5\x6b\x4d\xa2\xd9\xea", 8, GRND_NONBLOCK) = 8

brk(NULL)   = 0x55c8a2a78000

brk(0x55c8a2a99000)   = 0x55c8a2a99000

read(3, "b1efeddb19282324499a60d67727be25"..., 128) = 128

read(3, "823b7b\r\nfe546a37ce4b234f955dce54"..., 102) = 102

read(3, "cae6be\r\na9e34cfdc206c20283954d64"..., 102) = 102

read(3, "0984f2\r\na5def3557ee69d738a09d4b7"..., 102) = 102

read(3, "e14f77\r\nafdf3c1f4df5e4d070d4edaa"..., 102) = 102

read(3, "b8c9ba\r\n0da47829fc5b8de4a20981f7"..., 102) = 102

read(3, "c9e568\r\n2d55b1a47bf80decd0a5ca38"..., 102) = 102

read(3, "74bee3\r\n8ce2ca4ac828c74548f220d4"..., 102) = 102

read(3, "05c071\r\n8e6435966a37bb73ef50495e"..., 102) = 102

read(3, "4d9c82\r\n8ac17bacc12e299356a1696a"..., 102) = 102

read(3, "00b300\r\n6b7b44349f2117a0f6be1f7d"..., 102) = 102

rt_sigaction(SIGRT_1, {sa_handler=0x7fe2f4307870, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7fe2f42b8520},
NULL, 8) = 0

rt_sigprocmask(SIG_UNBLOCK, [RTMIN RT_1], NULL, 8) = 0

mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7fe2f3a72000

mprotect(0x7fe2f3a73000, 8388608, PROT_READ|PROT_WRITE) = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE
_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID,
child_tid=0x7fe2f4272910, parent_tid=0x7fe2f4272910, exit_signal=0, stack=0x7fe2f3a72000,
stack_size=0x7fff00, tls=0x7fe2f4272640} => {parent_tid=[15004]}, 88) = 15004

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

read(3, "862558\r\n90e40a697f9d17e80bc2c31d"..., 102) = 102

read(3, "4ecbb0\r\nb40a53f1c54c83d9ae85bdc3"..., 102) = 102

read(3, "5fe24b\r\nfe67060c4453dd02e1e2f760"..., 102) = 102

read(3, "a5bbe8\r\na4199e3d2ab1336b7e40e23d"..., 102) = 102

read(3, "e980fb\r\naad1f823d858051f9950df38"..., 102) = 102

read(3, "bc0ad1\r\n4364e962e94cb91d34b7469d"..., 102) = 102

read(3, "d0e910\r\n72cf58306b9119b6457cfcd8"..., 102) = 102

read(3, "54c324\r\n1a7e59d3af8d429d6fdbc08f"..., 102) = 102

read(3, "508186\r\nc7210582dbd03961a39e656c"..., 102) = 102

read(3, "632326\r\n11a7b2719600291573f30b76"..., 102) = 102

read(3, "2ce150\r\n5cc50f260441645a07edf59c"..., 102) = 102

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7fe2f4272910, parent_tid=0x7fe2f4272910, exit_signal=0, stack=0x7fe2f3a72000, stack_size=0x7fff00, tls=0x7fe2f4272640} => {parent_tid=[0]}, 88) = 15005

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

read(3, "e24d10\r\n8dd18fc9189d48a685a1541e"..., 102) = 102

read(3, "b78987\r\n9c13c119a196a9616e85a15e"..., 102) = 102

read(3, "d772a8\r\n679be98048d24cdfb98ac919"..., 102) = 102

read(3, "4e4416\r\nab28e993e2319221e507908c"..., 102) = 102

read(3, "1996a6\r\n164f871560b72c784da8b2b5"..., 102) = 102

read(3, "6253d4\r\n7e81b2e94e77a1df8790af7b"..., 102) = 102

read(3, "548829\r\n522b60cbb68df1cbb701df2a"..., 102) = 102

read(3, "6bc56b\r\n487a6114939da006e2580771"..., 102) = 102

read(3, "419384\r\nede8a47bb934baa77f616d3a"..., 102) = 102

read(3, "7cba3d\r\n0768c9965896a9eaede16734"..., 102) = 102

rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7fe2f4272910, parent_tid=0x7fe2f4272910, exit_signal=0, stack=0x7fe2f3a72000, stack_size=0x7fff00, tls=0x7fe2f4272640} => {parent_tid=[0]}, 88) = 15006

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

read(3, "27be25\r\nba7ec5d416569827a2744d28"..., 102) = 102

read(3, "823b7b\r\nfe546a37ce4b234f955dce54"..., 102) = 102

read(3, "cae6be\r\na9e34cfdc206c20283954d64"..., 102) = 102

read(3, "0984f2\r\na5def3557ee69d738a09d4b7"..., 102) = 102

read(3, "e14f77\r\nafdf3c1f4df5e4d070d4edaa"..., 102) = 102

read(3, "b8c9ba\r\n0da47829fc5b8de4a20981f7"..., 102) = 102

read(3, "c9e568\r\n2d55b1a47bf80decd0a5ca38"..., 102) = 102

read(3, "74bee3\r\n8ce2ca4ac828c74548f220d4"..., 102) = 102

read(3, "05c071\r\n8e6435966a37bb73ef50495e"..., 102) = 102

read(3, "4d9c82\r\n8ac17bacc12e299356a1696a"..., 102) = 102

read(3, "00b300\r\n6b7b44349f2117a0f6be1f7d"..., 102) = 102

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7fe2f4272910, parent_tid=0x7fe2f4272910, exit_signal=0, stack=0x7fe2f3a72000, stack_size=0x7fff00, tls=0x7fe2f4272640} => {parent_tid=[0]}, 88) = 15007

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

read(3, "862558\r\n90e40a697f9d17e80bc2c31d"..., 102) = 102

read(3, "4ecbb0\r\nb40a53f1c54c83d9ae85bdc3"..., 102) = 102

read(3, "5fe24b\r\nfe67060c4453dd02e1e2f760"..., 102) = 102

read(3, "a5bbe8\r\na4199e3d2ab1336b7e40e23d"..., 102) = 102

read(3, "e980fb\r\naad1f823d858051f9950df38"..., 102) = 102

read(3, "bc0ad1\r\n4364e962e94cb91d34b7469d"..., 102) = 102

read(3, "d0e910\r\n72cf58306b9119b6457cfcd8"..., 102) = 102

read(3, "54c324\r\n1a7e59d3af8d429d6fdbc08f"..., 102) = 102

read(3, "508186\r\nc7210582dbd03961a39e656c"..., 102) = 102

read(3, "632326\r\n11a7b2719600291573f30b76"..., 102) = 102

read(3, "2ce150\r\n5cc50f260441645a07edf59c"..., 102) = 102

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7fe2f4272910, parent_tid=0x7fe2f4272910, exit_signal=0, stack=0x7fe2f3a72000, stack_size=0x7fff00, tls=0x7fe2f4272640} => {parent_tid=[0]}, 88) = 15008

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

read(3, "e24d10", 102)   = 6

read(3, "", 96)   = 0

rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0

clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7fe2f4272910, parent_tid=0x7fe2f4272910, exit_signal=0, stack=0x7fe2f3a72000, stack_size=0x7fff00, tls=0x7fe2f4272640} => {parent_tid=[0]}, 88) = 15009

rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0

close(3)   = 0

write(1, "\320\241\321\200\320\265\320\264\320\275\320\265\320\265\320\260\321\200\320\270\321\204\320\274\320\265\321\202\320\270\321"..., 45Среднее арифметическое:) = 45

write(1, "113868790578454021\n", 19113868790578454021

)   = 19

clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=9583500}) = 0

write(1, "0.006696\n", 90.006696

)   = 9

exit_group(0)   = ?

+++ exited with 0 +++

Число потоков	Время выполнения, мс	Ускорение	Эффективность
1	648	1	1
2	607	1, 06	0.53
3	575	1,12	0.37
5	555	1,16	0.23
6	469	1,38	0.23

Объяснение.

Увеличение количества потоков уменьшает время выполнения программы за счёт вычислений, выполняемых параллельно. С каждым новым добавленный потоком, эффективность снижается. Это связано с тем, что потоки уменьшают время выполнения при помощи дополнительной нагрузки на ЦП, который может производить конечное количество операций в секунду. После превышения числа потоков над количеством ядер прирост эффективности практически отсутствует, так как параллельные потоки фактически выполняются последовательно.

Вывод

В ходе лабораторной работе я приобрел базовые навыки по работе с потоками в си. Помимо этого, я изучил основные принципы параллельного программирования, а также применил эту концепцию на практике. В ходе выполнения лабораторной работы я столкнулся с трудностями, так как тяжело было в первый раз написать параллельно выполняемый код.