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: Болдинова В.В.

Преподаватель: Бахарев В.Д.

Оценка: _____

Дата: 01.01.25

Москва, 2024

Постановка задачи

Вариант 17.

Найти в большом целочисленном массиве минимальный и максимальный элементы

Общий метод и алгоритм решения

Использованные системные вызовы:

- write(int fd, const void *buf, size_t count)* - Записывает данные из буфера buf в файл или поток, идентифицируемый файловым дескриптором fd
- _exit(int status) - Завершает выполнение процесса, немедленно освобождая ресурсы.
- sem_init(sem_t *sem, int pshared, unsigned int value)* - Инициализирует семафор с начальным значением value. Если pshared равен 0, семафор используется только внутри текущего процесса
- sem_wait(sem_t *sem)* - Уменьшает значение семафора на единицу. Если значение семафора равно 0, поток блокируется до тех пор, пока значение не увеличится.
- sem_post(sem_t *sem)* - Увеличивает значение семафора на единицу, разблокируя поток, ожидающий освобождения ресурса
- pthread_create(pthread_t *thread, const pthread_attr_t *attr, void *(*start_routine)(void *), void *arg)* - Создаёт новый поток, который начинает выполнение с функции start_routine. Аргумент arg передаётся в функцию как параметр
- pthread_join(pthread_t thread, void retval) - Блокирует текущий поток до завершения потока thread
- srand(unsigned int seed) - Инициализирует генератор случайных чисел начальным значением seed
- rand(void) - Возвращает случайное число в пределах от 0 до RAND_MAX.
- clock(void) - Возвращает количество тактов процессора, прошедших с начала выполнения программы.

Код программы

L2.c:

```c
#include <pthread.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <stdatomic.h>
#include <semaphore.h>

#define INT_SIZE 32

typedef struct thread_data {
    int* array;
    int start;
    int end;
} thread_data;

atomic_int global_min;
atomic_int global_max;
sem_t thread_limit;  // Семафор для ограничения количества потоков
```

```c
void* find_min_max(void* arg);
void write_int(int fd, int value);

int main(int argc, char** argv) {
    if (argc != 4) {
        const char error_msg[] = "Usage: ./program <array_size> <max_threads>
<seed>\n";
        write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
        _exit(EXIT_FAILURE);
    }

    long arr_size = atol(argv[1]);
    int max_threads = atoi(argv[2]);
    unsigned int seed = atoi(argv[3]);

    if (arr_size <= 0 || max_threads <= 0) {
        const char error_msg[] = "Error: Array size and max threads must be
positive integers.\n";
        write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
        _exit(EXIT_FAILURE);
    }

    // Инициализация массива
    int* array = malloc(arr_size * sizeof(int));
    if (!array) {
        const char error_msg[] = "Failed to allocate memory for array\n";
        write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
        _exit(EXIT_FAILURE);
    }

    srand(seed);
    for (long i = 0; i < arr_size; i++) {
        array[i] = rand() % 1000;
    }

    pthread_t* threads = malloc(max_threads * sizeof(pthread_t));
    thread_data* thread_data_arr = malloc(max_threads * sizeof(thread_data));

    // Инициализация семафора
    sem_init(&thread_limit, 0, max_threads);

    // Устанавливаем начальные значения для глобальных переменных
    atomic_store(&global_min, array[0]);
    atomic_store(&global_max, array[0]);

    // Засекаем общее время выполнения
    clock_t start_full = clock();

    int chunk_size = arr_size / max_threads + (arr_size % max_threads != 0);
    int thread_count = 0;

    for (int i = 0; i < max_threads; i++) {
        thread_data_arr[i].array = array;
        thread_data_arr[i].start = i * chunk_size;
        thread_data_arr[i].end = (i == max_threads - 1) ? arr_size : (i + 1)
* chunk_size;

        // Ограничение количества активных потоков с использованием семафора
        sem_wait(&thread_limit);

        if (pthread_create(&threads[i], NULL, find_min_max,
&thread_data_arr[i]) != 0) {
            const char error_msg[] = "Failed to create thread\n";
            write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
```

```
            free(array);
            _exit(EXIT_FAILURE);
        }
        thread_count++;
    }

    for (int i = 0; i < thread_count; i++) {
        if (pthread_join(threads[i], NULL) != 0) {
            const char error_msg[] = "Failed to join thread\n";
            write(STDERR_FILENO, error_msg, sizeof(error_msg) - 1);
            free(array);
            _exit(EXIT_FAILURE);
        }
    }

    clock_t end_full = clock();

    // Вывод результата
    const char time_msg[] = "Full time: ";
    write(STDOUT_FILENO, time_msg, sizeof(time_msg) - 1);
    write_int(STDOUT_FILENO, (int)(end_full - start_full));
    const char newline[] = " ticks\n";
    write(STDOUT_FILENO, newline, sizeof(newline) - 1);

    const char min_msg[] = "Global Min: ";
    write(STDOUT_FILENO, min_msg, sizeof(min_msg) - 1);
    write_int(STDOUT_FILENO, atomic_load(&global_min));
    write(STDOUT_FILENO, newline, sizeof(newline) - 1);

    const char max_msg[] = "Global Max: ";
    write(STDOUT_FILENO, max_msg, sizeof(max_msg) - 1);
    write_int(STDOUT_FILENO, atomic_load(&global_max));
    write(STDOUT_FILENO, newline, sizeof(newline) - 1);

    // Очистка ресурсов
    sem_destroy(&thread_limit);
    free(array);
    free(threads);
    free(thread_data_arr);

    return 0;
}

void* find_min_max(void* arg) {
    thread_data* data = (thread_data*)arg;

    int local_min = data->array[data->start];
    int local_max = data->array[data->start];

    for (int i = data->start; i < data->end; i++) {
        if (data->array[i] < local_min)
            local_min = data->array[i];
        if (data->array[i] > local_max)
            local_max = data->array[i];
    }

    // Атомарное обновление глобальных значений
    int current_min = atomic_load(&global_min);
    while (local_min < current_min &&
!atomic_compare_exchange_weak(&global_min, &current_min, local_min));

    int current_max = atomic_load(&global_max);
    while (local_max > current_max &&
!atomic_compare_exchange_weak(&global_max, &current_max, local_max));
```

```
    // Сигнализируем освобождение потока
    sem_post(&thread_limit);

    return NULL;
}

void write_int(int fd, int value) {
    char buffer[INT_SIZE];
    int len = snprintf(buffer, sizeof(buffer), "%d", value);
    write(fd, buffer, len);
}
```

Протокол работы программы

Тестирование:

root@LAPTOP-6G05B5VT:~# strace -o /mnt/d/si/OSI/Lab2/L2/out.txt /mnt/d/si/OSI/Lab2/L2/parent 10000 5 77777

Execution time: 1350 ticks

Minimum value: 0 ticks

Maximum value: 999 ticks

root@LAPTOP-6G05B5VT:~# strace -o /mnt/d/si/OSI/Lab2/L2/out.txt /mnt/d/si/OSI/Lab2/L2/parent 10000 5 77777

Execution time: 1309 ticks

Minimum value: 0 ticks

Maximum value: 999 ticks

root@LAPTOP-6G05B5VT:~# gcc -o /mnt/d/si/OSI/Lab2/L2/parent /mnt/d/si/OSI/Lab2/L2/L2.c

root@LAPTOP-6G05B5VT:~# strace -o /mnt/d/si/OSI/Lab2/L2/out.txt /mnt/d/si/OSI/Lab2/L2/parent 10000 5 77777

Execution time: 1713 ticks

Minimum value: 0 ticks

Maximum value: 999 ticks

Strace:

execve("/mnt/d/si/OSI/Lab2/L2/parent", ["/mnt/d/si/OSI/Lab2/L2/parent", "10000", "5", "77777"], 0x7ffc13365278 /* 26 vars */) = 0
brk(NULL) = 0x555cf5dcf000
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7f5937965000
access("/etc/ld.so.preload", R_OK) = -1 ENOENT (No such file or directory)

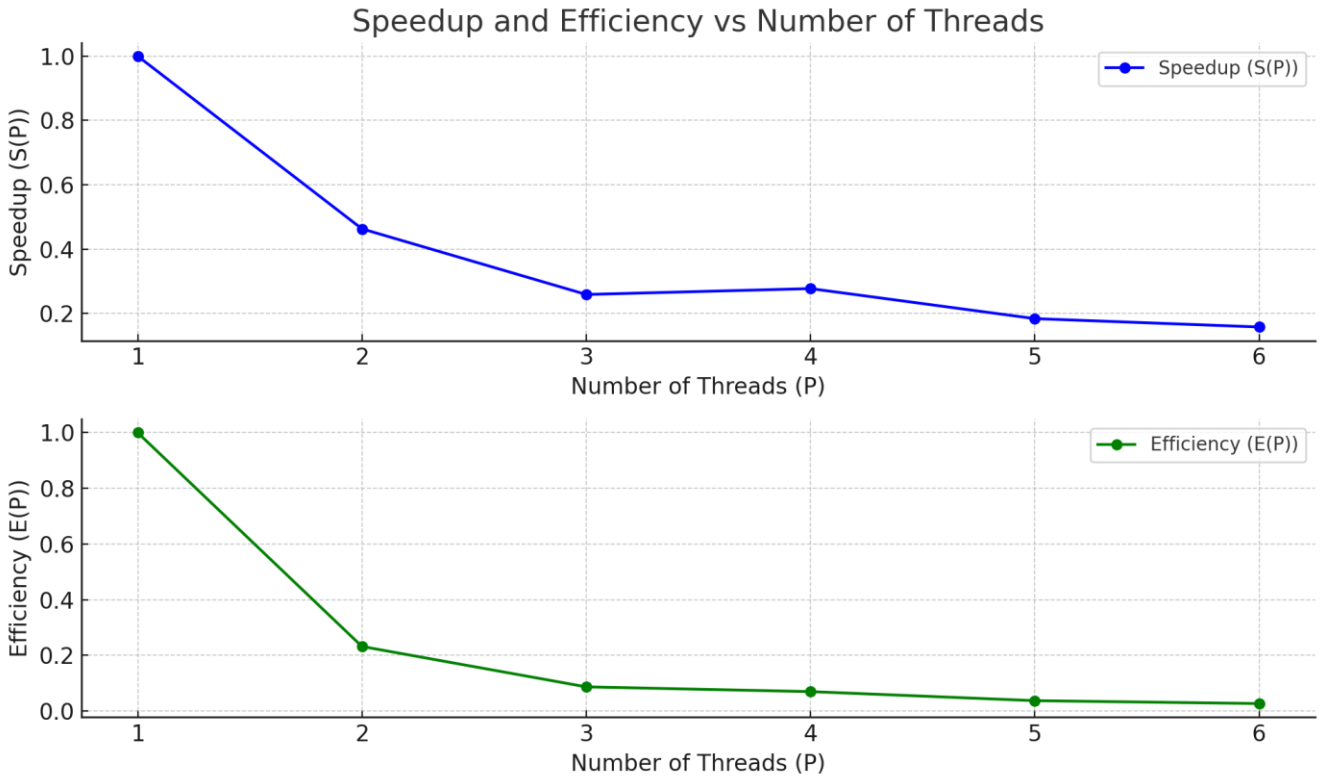
```
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
fstat(3, {st_mode=S_IFREG|0644, st_size=20115, ...}) = 0
mmap(NULL, 20115, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f5937960000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libc.so.6", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\3\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\220\243\2\0\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
fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 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, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) =
0x7f593774e000
mmap(0x7f5937776000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7f5937776000
mmap(0x7f59378fe000, 323584, PROT_READ,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1b0000) = 0x7f59378fe000
mmap(0x7f593794d000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7f593794d000
mmap(0x7f5937953000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7f5937953000
close(3)                                = 0
mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -
1, 0) = 0x7f593774b000
arch_prctl(ARCH_SET_FS, 0x7f593774b740) = 0
set_tid_address(0x7f593774ba10)         = 118152
set_robust_list(0x7f593774ba20, 24)     = 0
rseq(0x7f593774c060, 0x20, 0, 0x53053053) = 0
mprotect(0x7f593794d000, 16384, PROT_READ) = 0
mprotect(0x555ce143a000, 4096, PROT_READ) = 0
mprotect(0x7f593799d000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY})
= 0
munmap(0x7f5937960000, 20115)           = 0
getrandom("\x9f\x77\x08\xac\x91\x3b\xd0\xb1", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x555cf5dcf000
brk(0x555cf5df0000)                     = 0x555cf5df0000
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=3125500}) = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7f59377e7520, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO,
sa_restorer=0x7f5937793320}, 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) = 0x7f5936f4a000
mprotect(0x7f5936f4b000, 8388608, PROT_READ|PROT_WRITE) = 0
rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_T
HREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CH
ILD_CLEARTID, child_tid=0x7f593774a990, parent_tid=0x7f593774a990, exit_signal=0,
stack=0x7f5936f4a000, stack_size=0x7fff80, tls=0x7f593774a6c0} => {parent_tid=[118153]},
```

88) = 118153
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f5936749000
mprotect(0x7f593674a000, 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=0x7f5936f49990, parent_tid=0x7f5936f49990, exit_signal=0, stack=0x7f5936749000, stack_size=0x7fff80, tls=0x7f5936f496c0} => {parent_tid=[118154]}, 88) = 118154
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f5935f48000
mprotect(0x7f5935f49000, 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=0x7f5936748990, parent_tid=0x7f5936748990, exit_signal=0, stack=0x7f5935f48000, stack_size=0x7fff80, tls=0x7f59367486c0} => {parent_tid=[0]}, 88) = 118155
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f5935747000
mprotect(0x7f5935748000, 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=0x7f5935f47990, parent_tid=0x7f5935f47990, exit_signal=0, stack=0x7f5935747000, stack_size=0x7fff80, tls=0x7f5935f476c0} => {parent_tid=[0]}, 88) = 118156
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7f5934f46000
mprotect(0x7f5934f47000, 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=0x7f5935746990, parent_tid=0x7f5935746990, exit_signal=0, stack=0x7f5934f46000, stack_size=0x7fff80, tls=0x7f59357466c0} => {parent_tid=[0]}, 88) = 118157
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
munmap(0x7f5936f4a000, 8392704) = 0
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=4838400}) = 0
write(1, "Execution time: ", 16) = 16
write(1, "1713", 4) = 4
write(1, " ticks\n", 7) = 7
write(1, "Minimum value: ", 15) = 15

```
write(1, "0", 1)                    = 1
write(1, " ticks\n", 7)             = 7
write(1, "Maximum value: ", 15)     = 15
write(1, "999", 3)                  = 3
write(1, " ticks\n", 7)             = 7
exit_group(0)                       = ?
+++ exited with 0 +++
```

Число потоков	Время выполнения	ускорение	эффективность
1	63	1.000	1.000
2	136	0.463	0.232
3	243	0.259	0.086
4	227	0.278	0.069
5	342	0.184	0.037
6	398	0.158	0.026

Можно заметить, что при увеличении числа потоков ускорение уменьшается, а эффективность резко падает. Это связано с накладными расходами на синхронизацию и управление потоками.



Тестирование на 4 потоках

Размер массива	Время выполнения
100	244
10000	277
1000000	988
100000000	130450

Вывод по графикам и таблицам:

Задача поиска максимума и минимума в массиве не подходит для многопоточности, так как в этом случае программа замедляется. Это сама по себе задача поиска максимума и минимума элементарная. Многопоточность полезна для задач с высокой вычислительной сложностью или большим объемом независимых операций, например, для матричных операций, рендеринга графики или обработки больших данных. Однако для линейных задач с низкими вычислительными затратами многопоточность становится избыточной и снижает производительность из-за накладных расходов.

Вывод

В ходе выполнения лабораторной работы я освоила процесс создания многопоточных программ на языке Си, а также научилась синхронизировать потоки с использованием мьютексов. В процессе тестирования я проанализировала влияние количества потоков на производительность и ускорение выполнения алгоритма.