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:

#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  
  
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

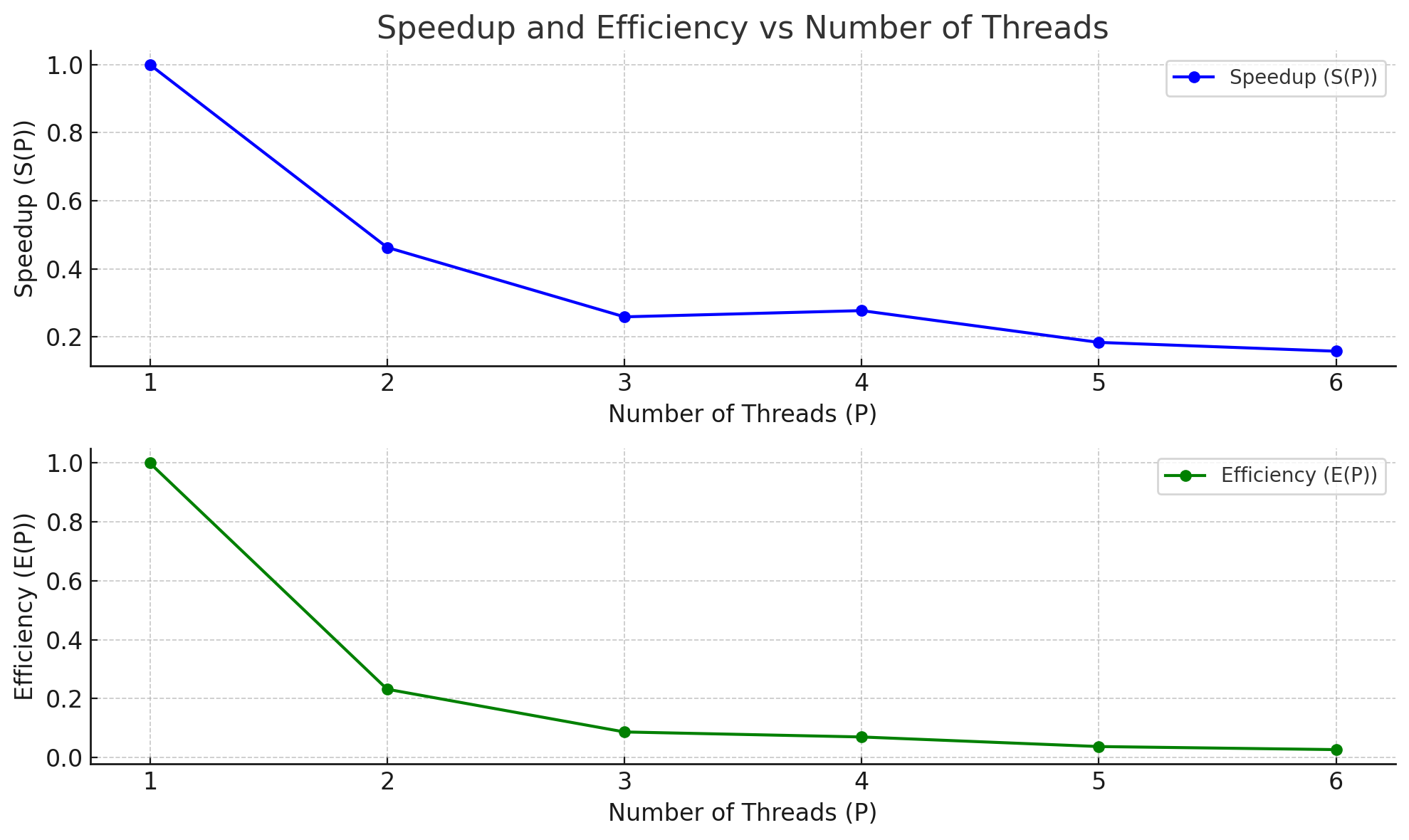
Maximum value: 999 ticks

**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\_THREAD|CLONE\_SYSVSEM|CLONE\_SETTLS|CLONE\_PARENT\_SETTID|CLONE\_CHILD\_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.