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-214БВ-24

Студент: Кириенко А. И.

Преподаватель Бахарев В. Д.

Оценка: _____

Дата: 11.11.25

Москва, 2024

Постановка задачи

Вариант 3.

Отсортировать массив целых чисел при помощи параллельной сортировки слиянием.

Общий метод и алгоритм решения

Задача - сортировка массива чисел методом сортировки слияния. Исходный массив делиться на две части. Каждая часть сортируется рекурсивно. Отсортированные массивы объединяются в упорядоченный массив.

Для ускорения вычислений используется многопоточность: каждый подмассив может обрабатываться в отдельном потоке. Создание слишком большого число поток приводит к потере производительности, поэтому чтобы этого избежать используется семафор (sem_t), который ограничивает максимальное количество одновременно выполняющихся потоков.

Алгоритм решения:

1. Генерируются случайные числа для массива. Создается копия массива для последовательной сортировки.
2. Вызывается функция merge_sort_recurs() - рекурсивная реализация сортировки слиянием без потоков.
3. Засекается время начала и конца выполнения с помощью gettimeofday(). И вычисляется время seq_time в миллисекундах.
4. Создается семафор и структура, содержащая параметры сортировки: указатель на массив, границы подмассива и указатель на семафор.
5. Основном поток запускает функцию merge_sort_thread() в новом потоке.
6. Главный поток ожидает завершения главного потока сортировки pthread_join. Засекается время выполнения par_time.
7. На экран выводятся время выполнения последовательной и параллельной сортировок. Печатается отсортированный массив.

Код программы

laba2.c

```c
#include <pthread.h> //для PROXI Threads

#include <stdlib.h> //для malloc

#include <semaphore.h> //для sem_t

#include <unistd.h> //для write()

#include <sys/time.h> //для gettimeofday

#include <time.h> //для time()/srand()

void write_int(long v){
```

```
    char buf[32];
    int i = 0;
    if (v == 0) {write(1, "0", 1); return;}
    if (v < 0) {write(1, "-", 1); v = -v;}
    while ( v > 0){
        buf[i++] = '0' + (v % 10);
        v/=10;
    }
    for (int j = i - 1; j >= 0; --j) write(1, &buf[j], 1);
}


void merge(int *arr, int left, int mid, int right){//слияние двух отсортированных подмассивов
    int i = left, j = mid + 1, k = 0;
    int len = right - left + 1; //количество элементов в подмассиве
    int *tmp = (int*) malloc(len * sizeof(int)); //просим ядро выделить память
    if (!tmp) return; //ошибка выделения

    while (i <= mid && j <= right){
        if (arr[i] <= arr[j])
            tmp[k++] = arr[i++];
        else
            tmp[k++] = arr[j++];
    }

    while (i <= mid) tmp[k++] = arr[i++]; //остатки левой части
    while (j <= right) tmp[k++] = arr[j++]; //остатки правой части

    for (i = 0; i < k; i++) arr[left + i] = tmp[i]; //записываем отсортированное в исходный массив
```

```
    free(tmp);


}
void merge_sort_recurs(int *arr, int left, int right){ //обычная рекурсивная сортировка без потоков (последовательная)
    if (left >= right) return;
    int mid = (left + right) / 2;
    merge_sort_recurs(arr, left, mid);
    merge_sort_recurs(arr, mid + 1, right);
    merge(arr, left, mid, right);
}
//структура аргументов потоков
typedef struct{
    int *arr;
    int left;
    int right;
    sem_t *sem; //указатель на семафор (ограничивает число одновременно работающих потоков)
}thread_arg_t;



void* merge_sort_thread(void *arg){
    thread_arg_t *data = (thread_arg_t*) arg; //преобразуем указатель void к типу thread_arg_t

    int left = data->left;
    int right = data->right;
    int *arr = data->arr;

    if (left >= right){
```

```
        sem_post(data->sem);//освобождаем место (один поток завершился)
        free(data);
        return NULL;
    }

    int mid = (left + right) / 2;

        //идентификаторы потоков
    pthread_t left_thread, right_thread;
    int created_left = 0, created_right = 0;

    if (sem_trywait(data->sem) == 0){
        thread_arg_t *left_arg = malloc(sizeof(thread_arg_t));
        if (left_arg){
            left_arg->arr = arr;
            left_arg->left = left;
            left_arg->right = mid;
            left_arg->sem = data->sem;

            if (pthread_create(&left_thread, NULL, merge_sort_thread, left_arg) ==
0)
    //**куда сохранить идентификатор, атибуты потока, функция которую будет
выполнять, аргумент который передаем в строку
                created_left = 1;
            else{
                sem_post(data->sem);
                free(left_arg);
                merge_sort_recurs(arr, left, mid);
            }
        }else{
```

```
            sem_post(data->sem);
            merge_sort_recurs(arr, left, mid); //если нет место, сортируем
        }
    } else{
        merge_sort_recurs(arr, left, mid);
    }

    //для правой:
    if (sem_trywait(data->sem) == 0){
        thread_arg_t *right_arg = malloc(sizeof(thread_arg_t));
        if (right_arg){
            right_arg->arr = arr;
            right_arg->left = mid + 1;
            right_arg->right = right;
            right_arg->sem = data->sem;

            if (pthread_create(&right_thread, NULL, merge_sort_thread, right_arg)
== 0)
                created_right = 1;
            else{
                sem_post(data->sem);
                free(right_arg);
                merge_sort_recurs(arr, mid + 1, right);
            }
        }else{
            sem_post(data->sem);
            merge_sort_recurs(arr, mid + 1, right); //если нет место, сортируем
        }
    } else{
```

```
        merge_sort_recurs(arr, mid + 1, right);
    }
    //ждем завершения
    if (created_left) pthread_join(left_thread, NULL);
    if (created_right) pthread_join(right_thread, NULL);

    merge(arr, left, mid, right); //слияение отсортированных половин

    sem_post(data->sem); //освобождаем разрешение
    free(data);
    return NULL;
}


int main(int argc, char* argv[]){
    if (argc != 2){
        write(2, "Usage: Enter 2 arguments\n", sizeof("Usage: Enter 2 arguments\n") - 1);
        return 1;
    }

    int MAX_THREADS = atoi(argv[1]); //максимум потоков одновременно (преобразуем строку в число)
    int N = 10000; //размер массива

    int *arr = (int*) malloc(N * sizeof(int));
    int* arr_copy = (int*) malloc(N * sizeof(int));
    if (!arr || !arr_copy){
        write(2, "Memory allocation error\n", sizeof("Memory allocation error\n") - 1);
```

```
        return 1;
    }

    srand(time(NULL)); //генератор случайных чисел
    for (int i = 0; i < N; i++){
        arr[i] = rand() % 10000; //случайное число от 0 до 9999
        arr_copy[i] = arr[i]; //сохраняем копию для второй версии сортировки
    }

    struct timeval start, end; //для измерения времени (миллисекунды)

    gettimeofday(&start, NULL);
    merge_sort_recurs(arr_copy, 0, N - 1);
    gettimeofday(&end, NULL);

    double seq_time = (end.tv_sec - start.tv_sec) * 1000.0 + (end.tv_usec -
start.tv_usec) / 1000.0;

    write(1, "Sequential sort done\n", 21);
    sem_t sem;
    sem_init(&sem, 0, MAX_THREADS); //инициализируем семафор

    thread_arg_t *arg = malloc(sizeof(thread_arg_t)); //структура аргументов для
главного потока сортировки
    if (!arg){
        write(2, "Memory allocation error\n", sizeof("Memory allocation error\n")
- 1);
        return 1;
    }
    arg->arr = arr;
```

```
    arg->left = 0;
    arg->right = N - 1;
    arg->sem = &sem;

    gettimeofday(&start, NULL);
    sem_wait(&sem);
    pthread_t main_thread;
    pthread_create(&main_thread, NULL, merge_sort_thread, arg);
    pthread_join(main_thread, NULL);

    gettimeofday(&end, NULL);

    double par_time = (end.tv_sec - start.tv_sec) * 1000.0 + (end.tv_usec -
start.tv_usec) / 1000.0;
    write(1, "Parallel sort done\n", 19);

    char buffer[128];
    write(1, "Sequential: ", sizeof("Sequential: ") - 1);
    write_int((long) (seq_time)); //миллисекунды
    write(1, " ms\n", 4);

    write(1, "Parallel: ", sizeof("Parallel: ") - 1);
    write_int((long) (par_time));
    write(1, " ms\n", 4);

    write(1, "Sorted array:\n", 14);
    for (int i = 0; i < N; i++) {
        write_int(arr[i]);
        write(1, " ", 1);
```

```
    }

    write(1, "\n", 1);


    sem_destroy(&sem);

    free(arr);

    free(arr_copy);


    return 0;
}
```

Протокол работы программы

Strace:

```
604   execve("./laba2", ["./laba2", "4"], 0x7ffcea2bfa00 /* 25 vars */) = 0

604   brk(NULL)                         = 0x5afe7e492000

604   mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7e5c02361000

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

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

604   fstat(3, {st_mode=S_IFREG|0644, st_size=19391, ...}) = 0

604   mmap(NULL, 19391, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7e5c0235c000

604   close(3)                          = 0

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

604   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"...,
832) = 832

604   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

604   fstat(3, {st_mode=S_IFREG|0755, st_size=2125328, ...}) = 0

604   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

604   mmap(NULL, 2170256, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7e5c02000000

604   mmap(0x7e5c02028000, 1605632, PROT_READ|PROT_EXEC,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x28000) = 0x7e5c02028000

604   mmap(0x7e5c021b0000, 323584, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1b0000) = 0x7e5c021b0000

604   mmap(0x7e5c021ff000, 24576, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1fe000) = 0x7e5c021ff000

604   mmap(0x7e5c02205000, 52624, PROT_READ|PROT_WRITE,
MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS, -1, 0) = 0x7e5c02205000

604   close(3)                          = 0

604   mmap(NULL, 12288, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7e5c02359000

604   arch_prctl(ARCH_SET_FS, 0x7e5c02359740) = 0
```

```
604   set_tid_address(0x7e5c02359a10)   = 604

604   set_robust_list(0x7e5c02359a20, 24) = 0

604   rseq(0x7e5c0235a060, 0x20, 0, 0x53053053) = 0

604   mprotect(0x7e5c021ff000, 16384, PROT_READ) = 0

604   mprotect(0x5afe66a69000, 4096, PROT_READ) = 0

604   mprotect(0x7e5c02399000, 8192, PROT_READ) = 0

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

604   munmap(0x7e5c0235c000, 19391)     = 0

604   getrandom("\x9a\xce\x65\xd3\xfd\x97\xa5\x0b", 8, GRND_NONBLOCK) = 8

604   brk(NULL)                         = 0x5afe7e492000

604   brk(0x5afe7e4b3000)               = 0x5afe7e4b3000

604   write(1, "Sequential sort done\n", 21) = 21

604   rt_sigaction(SIGRT_1, {sa_handler=0x7e5c02099530, sa_mask=[],
sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7e5c02045330}, NULL,
8) = 0

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

604   mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7e5c017ff000

604   mprotect(0x7e5c01800000, 8388608, PROT_READ|PROT_WRITE) = 0

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

604
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLON
E_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7e5c01fff990,
parent_tid=0x7e5c01fff990, exit_signal=0, stack=0x7e5c017ff000, stack_size=0x7fff80,
tls=0x7e5c01fff6c0} => {parent_tid=[605]}, 88) = 605

605   rseq(0x7e5c01ffffe0, 0x20, 0, 0x53053053 <unfinished ...>

604   rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>

605   <... rseq resumed>)               = 0

604   <... rt_sigprocmask resumed>NULL, 8) = 0

605   set_robust_list(0x7e5c01fff9a0, 24 <unfinished ...>
```

```
604   futex(0x7e5c01fff990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 605, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
605   <... set_robust_list resumed>)    = 0
605   rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
605   mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7e5bf97ff000
605   munmap(0x7e5bf97ff000, 41947136)  = 0
605   munmap(0x7e5c00000000, 25161728)  = 0
605   mprotect(0x7e5bfc000000, 135168, PROT_READ|PROT_WRITE) = 0
605   mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) =
0x7e5c00ffe000
605   mprotect(0x7e5c00fff000, 8388608, PROT_READ|PROT_WRITE) = 0
605   rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
605
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLON
E_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7e5c017fe990,
parent_tid=0x7e5c017fe990, exit_signal=0, stack=0x7e5c00ffe000, stack_size=0x7fff80,
tls=0x7e5c017fe6c0} <unfinished ...>
606   rseq(0x7e5c017fefe0, 0x20, 0, 0x53053053 <unfinished ...>
605   <... clone3 resumed> => {parent_tid=[606]}, 88) = 606
606   <... rseq resumed>)               = 0
605   rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
606   set_robust_list(0x7e5c017fe9a0, 24 <unfinished ...>
605   <... rt_sigprocmask resumed>NULL, 8) = 0
606   <... set_robust_list resumed>)    = 0
605   mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0
<unfinished ...>
606   rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
605   <... mmap resumed>)               = 0x7e5c007fd000
606   <... rt_sigprocmask resumed>NULL, 8) = 0
605   mprotect(0x7e5c007fe000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>
606   mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0 <unfinished ...>
605   <... mprotect resumed>)           = 0
```

```
606   <... mmap resumed>)               = 0x7e5bf4000000
605   rt_sigprocmask(SIG_BLOCK, ~[],  <unfinished ...>
606   munmap(0x7e5bf8000000, 67108864 <unfinished ...>
605   <... rt_sigprocmask resumed>[], 8) = 0
606   <... munmap resumed>)             = 0
605
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7e5c00ffd990, parent_tid=0x7e5c00ffd990, exit_signal=0, stack=0x7e5c007fd000, stack_size=0x7fff80, tls=0x7e5c00ffd6c0} <unfinished ...>
606   mprotect(0x7e5bf4000000, 135168, PROT_READ|PROT_WRITE <unfinished ...>
607   rseq(0x7e5c00ffdfe0, 0x20, 0, 0x53053053 <unfinished ...>
606   <... mprotect resumed>)           = 0
605   <... clone3 resumed> => {parent_tid=[607]}, 88) = 607
607   <... rseq resumed>)               = 0
606   mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0
<unfinished ...>
607   set_robust_list(0x7e5c00ffd9a0, 24 <unfinished ...>
605   rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
607   <... set_robust_list resumed>)    = 0
606   <... mmap resumed>)               = 0x7e5bfb7ff000
607   rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
605   <... rt_sigprocmask resumed>NULL, 8) = 0
607   <... rt_sigprocmask resumed>NULL, 8) = 0
606   mprotect(0x7e5bfb800000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>
607   mmap(0x7e5bf8000000, 67108864, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0
<unfinished ...>
605   futex(0x7e5c017fe990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 606, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
607   <... mmap resumed>)               = 0x7e5bf0000000
606   <... mprotect resumed>)           = 0
607   mprotect(0x7e5bf0000000, 135168, PROT_READ|PROT_WRITE <unfinished ...>
```

```
606   rt_sigprocmask(SIG_BLOCK, ~[],  <unfinished ...>
607   <... mprotect resumed>)           = 0
606   <... rt_sigprocmask resumed>[], 8) = 0
607   rt_sigprocmask(SIG_BLOCK, ~[RT_1],  <unfinished ...>
606
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7e5bfbfff990, parent_tid=0x7e5bfbfff990, exit_signal=0, stack=0x7e5bfb7ff000, stack_size=0x7fff80, tls=0x7e5bfbfff6c0} <unfinished ...>
607   <... rt_sigprocmask resumed>NULL, 8) = 0
607   madvise(0x7e5c007fd000, 8368128, MADV_DONTNEED <unfinished ...>
608   rseq(0x7e5bfbffffe0, 0x20, 0, 0x53053053 <unfinished ...>
606   <... clone3 resumed> => {parent_tid=[608]}, 88) = 608
608   <... rseq resumed>)               = 0
607   <... madvise resumed>)            = 0
608   set_robust_list(0x7e5bfbfff9a0, 24 <unfinished ...>
606   rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
608   <... set_robust_list resumed>)    = 0
607   exit(0 <unfinished ...>
608   rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
606   <... rt_sigprocmask resumed>NULL, 8) = 0
607   <... exit resumed>)               = ?
608   <... rt_sigprocmask resumed>NULL, 8) = 0
606   mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0
<unfinished ...>
608   rt_sigprocmask(SIG_BLOCK, ~[RT_1],  <unfinished ...>
607   +++ exited with 0 +++
608   <... rt_sigprocmask resumed>NULL, 8) = 0
606   <... mmap resumed>)               = 0x7e5bfaffe000
608   madvise(0x7e5bfb7ff000, 8368128, MADV_DONTNEED <unfinished ...>
606   mprotect(0x7e5bfafff000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>
```

```
608   <... madvise resumed>)            = 0
606   <... mprotect resumed>)           = 0
608   exit(0 <unfinished ...>
606   rt_sigprocmask(SIG_BLOCK, ~[],  <unfinished ...>
608   <... exit resumed>)               = ?
606   <... rt_sigprocmask resumed>[], 8) = 0
608   +++ exited with 0 +++
606
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7e5bfb7fe990, parent_tid=0x7e5bfb7fe990, exit_signal=0, stack=0x7e5bfaffe000, stack_size=0x7fff80, tls=0x7e5bfb7fe6c0} <unfinished ...>
609   rseq(0x7e5bfb7fefe0, 0x20, 0, 0x53053053 <unfinished ...>
606   <... clone3 resumed> => {parent_tid=[609]}, 88) = 609
609   <... rseq resumed>)               = 0
606   rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
609   set_robust_list(0x7e5bfb7fe9a0, 24 <unfinished ...>
606   <... rt_sigprocmask resumed>NULL, 8) = 0
609   <... set_robust_list resumed>)    = 0
606   futex(0x7e5bfb7fe990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 609, NULL, FUTEX_BITSET_MATCH_ANY <unfinished ...>
609   rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
609   rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
609
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7e5bfbfff990, parent_tid=0x7e5bfbfff990, exit_signal=0, stack=0x7e5bfb7ff000, stack_size=0x7fff80, tls=0x7e5bfbfff6c0} <unfinished ...>
610   rseq(0x7e5bfbffffe0, 0x20, 0, 0x53053053 <unfinished ...>
609   <... clone3 resumed> => {parent_tid=[610]}, 88) = 610
610   <... rseq resumed>)               = 0
609   rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
```

```
610   set_robust_list(0x7e5bfbfff9a0, 24 <unfinished ...>
609   <... rt_sigprocmask resumed>NULL, 8) = 0
610   <... set_robust_list resumed>)    = 0
609   futex(0x7e5bfbfff990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 610, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
610   rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
610   mmap(NULL, 134217728, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) = 0x7e5be8000000
610   munmap(0x7e5bec000000, 67108864)  = 0
610   mprotect(0x7e5be8000000, 135168, PROT_READ|PROT_WRITE) = 0
610   rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
610   madvise(0x7e5bfb7ff000, 8368128, MADV_DONTNEED) = 0
610   exit(0)                           = ?
610   +++ exited with 0 +++
609   <... futex resumed>)              = 0
609   rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
609   madvise(0x7e5bfaffe000, 8368128, MADV_DONTNEED) = 0
609   exit(0)                           = ?
609   +++ exited with 0 +++
606   <... futex resumed>)              = 0
606   rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
606   madvise(0x7e5c00ffe000, 8368128, MADV_DONTNEED) = 0
606   exit(0)                           = ?
606   +++ exited with 0 +++
605   <... futex resumed>)              = 0
605   rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
605   madvise(0x7e5c017ff000, 8368128, MADV_DONTNEED) = 0
605   exit(0)                           = ?
604   <... futex resumed>)              = 0
605   +++ exited with 0 +++
```

```
604   munmap(0x7e5bfb7ff000, 8392704)   = 0
604   write(1, "Parallel sort done\n", 19) = 19
604   write(1, "Sequential: ", 12)      = 12
604   write(1, "0", 1)                  = 1
604   write(1, " ms\n", 4)              = 4
604   write(1, "Parallel: ", 10)        = 10
604   write(1, "1", 1)                  = 1
604   write(1, "0", 1)                  = 1
604   write(1, "0", 1)                  = 1
604   write(1, " ms\n", 4)              = 4
604   write(1, "Sorted array:\n", 14)   = 14
604   write(1, "4", 1)                  = 1
604   write(1, "6", 1)                  = 1
604   write(1, " ", 1)                  = 1
604   write(1, "6", 1)                  = 1
604   write(1, "4", 1)                  = 1
604   write(1, " ", 1)                  = 1
604   write(1, "7", 1)                  = 1
604   write(1, "0", 1)                  = 1
604   write(1, " ", 1)                  = 1
604   write(1, "1", 1)                  = 1
604   write(1, "8", 1)                  = 1
604   write(1, " ", 1)                  = 1
604   write(1, "\n", 1)                 = 1
604   exit_group(0)                     = ?
604   +++ exited with 0 +++
```

Вывод программы:

```
nastik@HUAWEI:/mnt/c/nastya/laba2 OS$ gcc laba2.c -o laba2 -lpthread
nastik@HUAWEI:/mnt/c/nastya/laba2 OS$ ./laba2 6
Sequential sort done
Parallel sort done
Sequential: 2 ms
Parallel: 4 ms
Sorted array:
0 2 6 6 8 9 10 10 11 11 12 14 14 16 16 16 16 17 19 20 21 24 26 26 31 32 32 33 36 37 39 40 41 43 43 43 43 43 45 45 46 46 47 47 48 49 51 51 52 52 53 56 57 58 60 61 62 63 67
67 69 70 73 73 73 75 77 79 79 79 80 82 82 86 88 89 89 90 93 93 93 95 96 97 98 99 100 100 101 102 103 105 105 105 107 108 109 109 109 110 112 112 113 113 113 114 116 118 11
9 119 119 119 119 120 121 122 123 123 125 125 126 126 127 129 130 131 132 132 133 134 136 137 138 138 140 140 141 142 142 143 146 147 147 148 151 153 154 154 154 155 156 1
58 159 159 164 165 168 169 169 170 171 171 172 172 174 175 175 176 176 177 177 178 180 181 182 182 183 184 184 184 186 187 187 187 187 189 190 190 191 192 193 194 194 195
195 196 198 201 204 205 206 208 208 209 211 211 212 212 213 214 214 216 217 220 220 223 224 224 224 226 227 227 228 232 232 234 234 235 237 238 239 240 242 242 242 243 244
 245 245 246 246 247 247 247 248 248 250 251 251 251 252 252 253 253 254 256 258 262 263 264 264 265 265 269 272 275 277 277 277 278 279 281 281 282 282 283 284 284 284 28
4 285 286 287 288 288 288 288 289 290 291 291 293 295 295 296 300 300 301 303 306 306 306 307 308 308 308 309 310 310 312 312 313 314 317 317 317 318 318 319 321 321 322 3
22 323 324 326 327 328 329 330 330 331 332 332 332 334 335 336 337 337 338 340 341 341 342 342 345 346 347 348 348 350 352 352 352 353 355 356 356 357 359 359 359 359 360
360 360 361 362 363 363 364 365 366 368 368 369 372 372 373 373 375 377 380 382 383 383 384 384 385 385 386 386 386 386 388 392 392 393 395 395 397 398 399 399 399 400 400
 400 400 402 402 402 403 406 406 406 406 407 407 409 410 411 412 413 415 419 420 422 423 423 425 429 431 431 432 432 433 436 436 437 437 439 439 440 440 441 441 443 443 44
3 444 445 445 446 446 447 447 449 449 449 449 452 454 454 454 454 456 456 457 460 461 462 462 463 465 469 470 471 471 475 476 478 479 479 481 482 483 485 485 487 489 490 492 4
93 493 494 494 494 496 497 498 499 500 500 501 504 505 505 507 509 510 514 515 516 518 518 519 519 519 520 520 521 521 522 522 523 524 525 526 526 527 527 527 529 530 530
533 533 536 536 537 539 542 543 543 545 547 549 549 550 551 552 553 554 561 561 561 562 565 566 566 566 569 570 570 571 571 573 575 575 575 575 575 578 578 578 578 580 588
 580 582 582 582 582 582 583 584 584 586 586 587 587 588 591 591 591 592 593 593 593 595 595 599 601 602 604 605 606 607 610 612 612 612 613 614 614 615 615 616 617 618 61
```

```
nastik@HUAWEI:/mnt/c/nastya/laba2 OS$ ./laba2 4
Sequential sort done
Parallel sort done
Sequential: 0 ms
Parallel: 1 ms
Sorted array:
18 158 271 321 436 528 692 727 753 829 1036 1063 1285 1355 1473 1563 1745 1761 1792 1799 1899 1917 1941 2471 2629 2648 2847 2920 2959 3002 3147 3204 3456 3576 3580 3630 37
27 3780 3932 4015 4128 4226 4379 4564 5333 5347 5667 5735 5784 5863 5961 5974 6012 6102 6113 6220 6228 6348 6373 6383 6749 6971 7137 7274 7549 7557 7566 7659 7712 7743 793
0 7945 7948 7964 8009 8045 8465 8661 8691 8748 8882 8893 8929 9019 9027 9215 9315 9363 9421 9431 9469 9514 9675 9682 9743 9798 9882 9939 9950 9992
nastik@HUAWEI:/mnt/c/nastya/laba2 OS$ ./laba2 10
Sequential sort done
Parallel sort done
Sequential: 0 ms
Parallel: 4 ms
Sorted array:
108 330 394 427 437 438 532 706 771 921 1015 1127 1212 1251 1313 1486 1627 1661 1667 1732 1833 1897 1975 2398 2424 2446 2663 2679 2693 2818 3028 3271 3321 3322 3513 3651 3
742 3743 3832 3892 4096 4156 4186 4232 4337 4485 4560 4564 4640 4693 4800 4853 4971 4994 5057 5170 5232 5343 5427 5454 5527 5549 5746 5764 5847 5881 5922 6012 6131 6147 62
77 6532 6541 6668 6728 7010 7197 7254 7463 7511 7524 7991 8348 8365 8523 8977 9039 9069 9103 9178 9204 9243 9327 9330 9422 9517 9523 9707 9735 9877
```

.

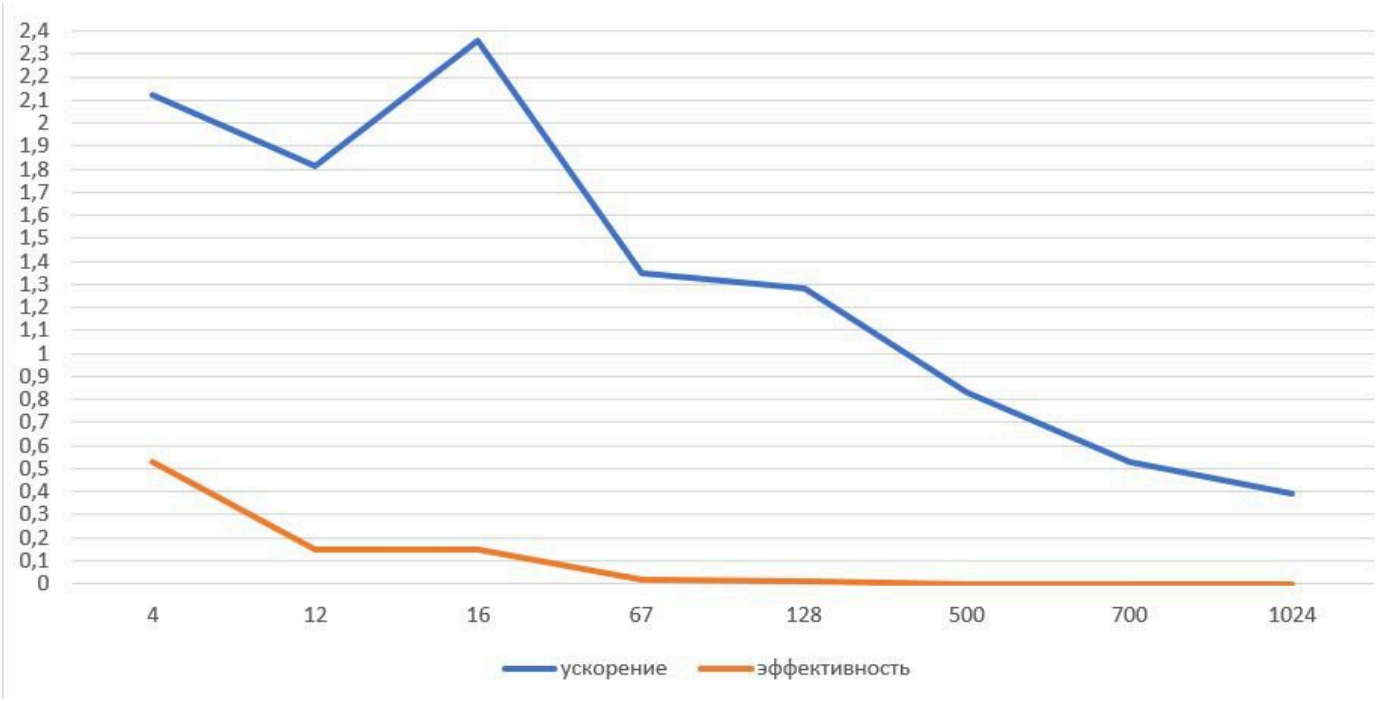
Сравнительная таблица

У меня в системе 12 логических ядер.

Число потоков N	Объем входных данных	Время исполнения в одном потоке (мс)	Время исполнения в N потоках (мс)	Ускорение	Эффективность
4	10000	2	3	0.667	0.167
4	10000	2	2	1	0.25
4	10000	2	2	1	0.25
4	10000	2	2	1	0.25
12	10000	2	8	0.25	0.021
16	10000	2	12	0.167	0.01
128	10000	2	95	0.021	0.0002
1024	10000	2	496	0.004	0.000004
4	1000000	308	145	2.124	0.531
4	1000000	314	176	1.784	0.446

12	1000000	294	162	1.815	0.151
16	1000000	292	124	2.355	0.147
128	1000000	284	221	1.285	0.01
1024	1000000	291	738	0.394	0.0003

Диаграмма зависимости ускорения и эффективности от количества потоков



При более большом объеме данных эффективней использовать многопоточность. Мой график убывает, тк объем данных не велик, и время затраченное на сортировку сопоставимо или даже меньше времени создания потока, поэтому получается, что на небольшом объеме данных выгоднее использовать однопоточную сортировку. При увеличении объема данных затраты времени на создание процессов становятся незначительными по сравнению с вычислениями.

Вывод

В ходе выполнения лабораторной работы я получила новые знания и навыки в области работы с потоками. В результате работы была реализована многопоточная сортировка слиянием, в которой использовались POSIX-потоки и семафоры для ограничения количества одновременно выполняющихся потоков. Так же было измерено время работы последовательного и параллельного алгоритмов, построен график зависимости ускорения и эффективности от количества потоков.