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: М80-206Б-20

Студент: Волков А.Д.

Преподаватель: Миронов Е.С.

Оценка: _____

Дата: 03.11.2023

Москва, 2023

Постановка задачи

Вариант 1.

Отсортировать массив целых чисел при помощи битонической сортировки

Общий метод и алгоритм решения

Использованные системные вызовы:

- int pthread_create(pthread_t *restrict thread ,
  const pthread_attr_t *restrict attr,
  void *(* start_routine)(void *),
  void *restrict arg) – создает новый поток, в вызывающем процессе;
- int pthread_join(pthread_t thread, void **retval) – ждет завершения указанного потока;

Чтобы выполнить данную лабораторную работу, мне было необходимо изучить теоретическую базу, а именно потоки и сама битоническая сортировка. После этого я стал реализовывать однопоточный алгоритм битонической сортировки, а дальше уже и многопоточный. Программа может сама себе генерировать тестовые данные, единственное что нужно указать пользователю при запуске, это количество потоков и количество элементов в сортируемой последовательности. После того как программа обработала все пользовательские данные и проверила их (например проверка на степень двойки, т.к. битоническая сортировка работает только с последовательностями, количество элементов в которых равно степени двойки), она запускает функцию bitonic_sort, которая и будет заниматься сортировкой последовательности. В качестве аргументов, этой функции передается указатель на структуру, в которой хранятся все необходимые для сортировки данные, а также в ней находится семафор, который нужен для остановки работы в многопоточном режиме и продолжения работы в однопоточном, так как в течение сортировки, количества указанных пользователем потоков может не хватить для полной параллельной сортировки последовательности. Сама функция bitonic_sort делит последовательность на две части, после чего создает два потока, и передает им эти половинки. Потоки будут рекурсивно вызывать функцию bitonic_sort. Либо же, если указанных пользователем потоков уже не хватает, то функция будет работать по такому же алгоритму только в однопоточном режиме. Также необходимо отметить, что в любом из этих случаев у нас вызывается функция bitonic_merge, которая занимается слиянием этих половинок последовательности, с помощью принципа “Разделяй и властвуй”. Алгоритм этой функции почти аналогичен алгоритму работы функции bitonic_sort, за исключением того, что bitonic_merge формирует битонические последовательности, которые и нужны для битонической сортировки.

Код программы

bitonic_sort_parallel.cpp

```
#include <bits/stdc++.h>

#include <pthread.h>

#include <ctime>

#include "generator.hpp"

typedef struct {

    int low_edge; int count; int order; unsigned NUM_OF_THREADS; unsigned last_threads;
std::vector<int> *arr;
```

```
} ThreadArg;


inline std::ostream& operator<<(std::ostream& os, std::vector<int> vec) {
    for (int i = 0; i < vec.size(); i++) {
        os << vec[i] << " ";
    }
    return os;
}


void compAndSwap(std::vector<int>& arr, int i, int j, int order) {
    if (order == (arr[i] > arr[j])) {
        std::swap(arr[i], arr[j]);
    }
}


void *bitonic_merge(void *arg) {
    ThreadArg *args = (ThreadArg *)arg;
    int low_edge = args->low_edge; int count = args->count; int order = args->order;
    if (count > 1) {
        int k = count / 2;
        for (int i = low_edge; i < low_edge + k; i++) {
            compAndSwap(*args->arr, i, i + k, order);
        }
        if (args->last_threads > 1) {
            ThreadArg arg1 {low_edge, k, order, args->NUM_OF_THREADS,
args->last_threads / 2, args->arr};
            ThreadArg arg2 {low_edge + k, k, order, args->NUM_OF_THREADS,
args->last_threads / 2, args->arr};
            pthread_t thread1, thread2;
            pthread_create(&thread1, NULL, bitonic_merge, &arg1);
pthread_create(&thread2, NULL, bitonic_merge, &arg2);
            pthread_join(thread1, NULL); pthread_join(thread2, NULL);
        } else {
            ThreadArg arg1 {low_edge, k, order, args->NUM_OF_THREADS,
args->last_threads, args->arr};
            ThreadArg arg2 {low_edge + k, k, order, args->NUM_OF_THREADS,
args->last_threads, args->arr};
            bitonic_merge(&arg1); bitonic_merge(&arg2);
```

```
        }

    }

    return nullptr;

}


void *bitonicSort(void *arg) {

    ThreadArg *threadarg = (ThreadArg *)arg;

    int low_edge = threadarg->low_edge; int count = threadarg->count; int order =
threadarg->order;

    if (count > 1) {

        int k = count / 2;

        if (threadarg->last_threads / 2 || (threadarg->last_threads ==
threadarg->NUM_OF_THREADS && threadarg->NUM_OF_THREADS)) {

            ThreadArg arg1 {low_edge, k, 1, threadarg->NUM_OF_THREADS,
threadarg->last_threads / 2, threadarg->arr};

            ThreadArg arg2 {low_edge + k, k, 0, threadarg->NUM_OF_THREADS,
threadarg->last_threads / 2, threadarg->arr};

            pthread_t thread1, thread2;

            pthread_create(&thread1, NULL, bitonicSort, &arg1);
pthread_create(&thread2, NULL, bitonicSort, &arg2);

            pthread_join(thread1, NULL); pthread_join(thread2, NULL);

            bitonic_merge(threadarg);

        } else {

            ThreadArg arg1 {low_edge, k, 1, threadarg->NUM_OF_THREADS,
threadarg->last_threads, threadarg->arr};

            ThreadArg arg2 {low_edge + k, k, 0, threadarg->NUM_OF_THREADS,
threadarg->last_threads, threadarg->arr};

            bitonicSort(&arg1); bitonicSort(&arg2);

            bitonic_merge(threadarg);

        }

    }

    return nullptr;

}


int main(int argc, char *argv[]) {

    if (argc != 3) {

        std::cerr << "Invalid quantity of arguments!\n";

        return -1;

    }
```

```
    unsigned NUM_OF_THREADS = std::stoi(argv[1]);

    bool isPowerOfTwo = NUM_OF_THREADS && !(NUM_OF_THREADS & (NUM_OF_THREADS - 1));

    if (!isPowerOfTwo) {

        std::cerr << "Quantity of threads is not a power of two!\n";

        return -1;

    }

    const int QUANTITY_OF_ELEMENTS = std::stoi(argv[2]);

    isPowerOfTwo = QUANTITY_OF_ELEMENTS && !(QUANTITY_OF_ELEMENTS &
(QUANTITY_OF_ELEMENTS - 1));

    if (!isPowerOfTwo) {

        std::cerr << "Quantity of elements is not a power of two!\n";

        return -1;

    }

    if (NUM_OF_THREADS > QUANTITY_OF_ELEMENTS) {

        std::cerr << "Quantity of threads is greater than size of array!\n";

        return -1;

    }

    unsigned power = 0;

    while (pow(2, power) < NUM_OF_THREADS) {

        power++;

    }

    NUM_OF_THREADS = power;

    std::vector<int> vec = generate_tests(QUANTITY_OF_ELEMENTS);

    ThreadArg mainArg {0, QUANTITY_OF_ELEMENTS, 1, NUM_OF_THREADS, NUM_OF_THREADS,
&vec};

    std::cout << "Initial array: " << vec << "\n";

    std::clock_t start = std::clock();

    bitonicSort(&mainArg);

    double duration = (std::clock() - start) / (double) CLOCKS_PER_SEC;

    std::cout << "Sorted array: " << vec << "\n";

    std::cout << "Sorted for " << std::fixed << duration * 1000 << "ms\n";

}
```

Протокол работы программы

Тестирование:

```
$ ./lab 1 4
Initial sequence: 3310 2154 454 1647
Sorted sequence: 454 1647 2154 3310
Sorted for 0.001000ms
$ ./lab 1 8
Initial sequence: 1961 107 877 2298 995 2744 1835 1401
Sorted sequence: 107 877 995 1401 1835 1961 2298 2744
Sorted for 0.009000ms
$ ./lab 2 1
Quantity of threads is greater than size of sequence!
$ ./lab 2 8
Initial sequence: 760 3717 3999 3831 4004 1451 3269 3722
Sorted sequence: 760 1451 3269 3717 3722 3831 3999 4004
Sorted for 0.935000ms
$ ./lab 4 4
Initial sequence: 1837 3238 3957 3101
Sorted sequence: 1837 3101 3238 3957
Sorted for 1.155000ms
```

Strace:

Посмотрим на вывод утилиты strace для 4-х поточной сортировки

```
$ strace -f ./lab 4 4
execve("./lab", ["./lab", "4", "4"], 0x7ffc8d5fff08 /* 76 vars */) = 0
brk(NULL)                               = 0x564f2dd75000
arch_prctl(0x3001 /* ARCH_??? */, 0x7ffc70a9f1c0) = -1 EINVAL (Недопустимый аргумент)
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f02c3494000
access("/etc/ld.so.preload", R_OK)      = -1 ENOENT (Нет такого файла или каталога)
openat(AT_FDCWD, "/etc/ld.so.cache", O_RDONLY|O_CLOEXEC) = 3
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=67971, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 67971, PROT_READ, MAP_PRIVATE, 3, 0) = 0x7f02c3483000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libstdc++.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\0\0\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=2260296, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 2275520, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f02c3200000
mprotect(0x7f02c329a000, 1576960, PROT_NONE) = 0
mmap(0x7f02c329a000, 1118208, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
 3, 0x9a000) = 0x7f02c329a000
mmap(0x7f02c33ab000, 454656, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
 0x1ab000) = 0x7f02c33ab000
mmap(0x7f02c341b000, 57344, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x21a000) = 0x7f02c341b000
mmap(0x7f02c3429000, 10432, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
 -1, 0) = 0x7f02c3429000
close(3)                                = 0
```

```
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libm.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\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=940560, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 942344, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f02c3119000
mmap(0x7f02c3127000, 507904, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
 3, 0xe000) = 0x7f02c3127000
mmap(0x7f02c31a3000, 372736, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
 0x8a000) = 0x7f02c31a3000
mmap(0x7f02c31fe000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
 3, 0xe4000) = 0x7f02c31fe000
close(3)                                = 0
openat(AT_FDCWD, "/lib/x86_64-linux-gnu/libgcc_s.so.1", O_RDONLY|O_CLOEXEC) = 3
read(3, "\177ELF\2\1\1\0\0\0\0\0\0\0\0\3\0>\0\1\0\0\0\0\0\0\0\0\0\0"..., 832) = 832
newfstatat(3, "", {st_mode=S_IFREG|0644, st_size=125488, ...}, AT_EMPTY_PATH) = 0
mmap(NULL, 127720, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f02c3463000
mmap(0x7f02c3466000, 94208, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x3000) = 0x7f02c3466000
mmap(0x7f02c347d000, 16384, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3, 0x1a000)
 = 0x7f02c347d000
mmap(0x7f02c3481000, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x1d000) = 0x7f02c3481000
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\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\0\244;\374\204(\337f#\315I\214\234\f\256\271\32"..., 68,
896) = 68
newfstatat(3, "", {st_mode=S_IFREG|0755, st_size=2216304, ...}, 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, 2260560, PROT_READ, MAP_PRIVATE|MAP_DENYWRITE, 3, 0) = 0x7f02c2e00000
mmap(0x7f02c2e28000, 1658880, PROT_READ|PROT_EXEC, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
3, 0x28000) = 0x7f02c2e28000
mmap(0x7f02c2fbd000, 360448, PROT_READ, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE, 3,
0x1bd000) = 0x7f02c2fbd000
mmap(0x7f02c3015000, 24576, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_DENYWRITE,
 3, 0x214000) = 0x7f02c3015000
mmap(0x7f02c301b000, 52816, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_FIXED|MAP_ANONYMOUS,
 -1, 0) = 0x7f02c301b000
close(3)                                = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f02c3461000
arch_prctl(ARCH_SET_FS, 0x7f02c34623c0) = 0
set_tid_address(0x7f02c3462690)         = 55277
set_robust_list(0x7f02c34626a0, 24)     = 0
rseq(0x7f02c3462d60, 0x20, 0, 0x53053053) = 0
```

```
mprotect(0x7f02c3015000, 16384, PROT_READ) = 0
mprotect(0x7f02c3481000, 4096, PROT_READ) = 0
mprotect(0x7f02c31fe000, 4096, PROT_READ) = 0
mmap(NULL, 8192, PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS, -1, 0) =
0x7f02c345f000
mprotect(0x7f02c341b000, 45056, PROT_READ) = 0
mprotect(0x564f2d2f3000, 4096, PROT_READ) = 0
mprotect(0x7f02c34ce000, 8192, PROT_READ) = 0
prlimit64(0, RLIMIT_STACK, NULL, {rlim_cur=8192*1024, rlim_max=RLIM64_INFINITY}) = 0
munmap(0x7f02c3483000, 67971)           = 0
getrandom("\x7d\x05\x55\x04\x8e\x8a\x65\x7d", 8, GRND_NONBLOCK) = 8
brk(NULL)                               = 0x564f2dd75000
brk(0x564f2dd96000)                     = 0x564f2dd96000
futex(0x7f02c342977c, FUTEX_WAKE_PRIVATE, 2147483647) = 0
newfstatat(1, "", {st_mode=S_IFCHR|0620, st_rdev=makedev(0x88, 0x1), ...},
AT_EMPTY_PATH) = 0
write(1, "Initial array: 4095 4046 443 89 "..., 33Initial array: 4095 4046 443 89
) = 33
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=3273801}) = 0
rt_sigaction(SIGRT_1, {sa_handler=0x7f02c2e91870, sa_mask=[],
 sa_flags=SA_RESTORER|SA_ONSTACK|SA_RESTART|SA_SIGINFO, sa_restorer=0x7f02c2e42520},
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) =
0x7f02c25ff000
mprotect(0x7f02c2600000, 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|CL
ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f02c2dff910
, parent_tid=0x7f02c2dff910, exit_signal=0, stack=0x7f02c25ff000, stack_size=0x7fff00,
tls=0x7f02c2dff640}strace: Process 55278 attached
 => {parent_tid=[55278]}, 88) = 55278
[pid 55277] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
[pid 55278] rseq(0x7f02c2dfffe0, 0x20, 0, 0x53053053 <unfinished ...>
[pid 55277] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 55278] <... rseq resumed>)         = 0
[pid 55277] mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0
<unfinished ...>
[pid 55278] set_robust_list(0x7f02c2dff920, 24 <unfinished ...>
[pid 55277] <... mmap resumed>)         = 0x7f02c1dfe000
[pid 55278] <... set_robust_list resumed>) = 0
[pid 55277] mprotect(0x7f02c1dff000, 8388608, PROT_READ|PROT_WRITE <unfinished ...>
[pid 55278] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
[pid 55277] <... mprotect resumed>)     = 0
[pid 55278] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 55277] rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
[pid 55278] rt_sigprocmask(SIG_BLOCK, ~[RT_1],  <unfinished ...>
[pid 55277]
```

```
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CLONE_S
      ETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f02c25fe910,
      parent_tid=0x7f02c25fe910, exit_signal=0, stack=0x7f02c1dfe000, stack_size=0x7fff00,
      tls=0x7f02c25fe640} <unfinished ...>
      [pid 55278] <... rt_sigprocmask resumed>NULL, 8) = 0
      [pid 55278] madvise(0x7f02c25ff000, 8368128, MADV_DONTNEED <unfinished ...>
      [pid 55277] <... clone3 resumed> => {parent_tid=[55279]}, 88) = 55279
      [pid 55278] <... madvise resumed>)      = 0
      [pid 55277] rt_sigprocmask(SIG_SETMASK, [], strace: Process 55279 attached
      NULL, 8) = 0
      [pid 55278] exit(0 <unfinished ...>
      [pid 55277] futex(0x7f02c2dff910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 55278, NULL,
      FUTEX_BITSET_MATCH_ANY <unfinished ...>
      [pid 55279] rseq(0x7f02c25fefe0, 0x20, 0, 0x53053053 <unfinished ...>
      [pid 55278] <... exit resumed>)         = ?
      [pid 55279] <... rseq resumed>)         = 0
      [pid 55277] <... futex resumed>)        = 0
      [pid 55279] set_robust_list(0x7f02c25fe920, 24 <unfinished ...>
      [pid 55278] +++ exited with 0 +++
      [pid 55277] futex(0x7f02c25fe910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 55279, NULL,
       FUTEX_BITSET_MATCH_ANY <unfinished ...>
      [pid 55279] <... set_robust_list resumed>) = 0
      [pid 55279] rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
      [pid 55279] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
      [pid 55279] madvise(0x7f02c1dfe000, 8368128, MADV_DONTNEED) = 0
      [pid 55279] exit(0)                     = ?
      [pid 55277] <... futex resumed>)        = 0
      [pid 55279] +++ exited with 0 +++
      rt_sigprocmask(SIG_BLOCK, ~[], [], 8)   = 0
      clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CL
      ONE_SETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f02c25fe910
      , parent_tid=0x7f02c25fe910, exit_signal=0, stack=0x7f02c1dfe000, stack_size=0x7fff00,
      tls=0x7f02c25fe640}strace: Process 55280 attached
       => {parent_tid=[55280]}, 88) = 55280
      [pid 55280] rseq(0x7f02c25fefe0, 0x20, 0, 0x53053053 <unfinished ...>
      [pid 55277] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
      [pid 55280] <... rseq resumed>)         = 0
      [pid 55277] <... rt_sigprocmask resumed>NULL, 8) = 0
      [pid 55280] set_robust_list(0x7f02c25fe920, 24 <unfinished ...>
      [pid 55277] rt_sigprocmask(SIG_BLOCK, ~[],  <unfinished ...>
      [pid 55280] <... set_robust_list resumed>) = 0
      [pid 55277] <... rt_sigprocmask resumed>[], 8) = 0
      [pid 55280] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
      [pid 55277]
      clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|CL
      ONE_S
      ETTLS|CLONE_PARENT_SETTID|CLONE_CHILD_CLEARTID, child_tid=0x7f02c2dff910,
       parent_tid=0x7f02c2dff910, exit_signal=0, stack=0x7f02c25ff000, stack_size=0x7fff00,
```

```
tls=0x7f02c2dff640} <unfinished ...>
[pid 55280] <... rt_sigprocmask resumed>NULL, 8) = 0
strace: Process 55281 attached
[pid 55280] rt_sigprocmask(SIG_BLOCK, ~[RT_1],  <unfinished ...>
[pid 55277] <... clone3 resumed> => {parent_tid=[55281]}, 88) = 55281
[pid 55281] rseq(0x7f02c2dfffe0, 0x20, 0, 0x53053053 <unfinished ...>
[pid 55277] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
[pid 55280] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 55277] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 55281] <... rseq resumed>)         = 0
[pid 55277] futex(0x7f02c25fe910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 55280, NULL,
 FUTEX_BITSET_MATCH_ANY <unfinished ...>
[pid 55280] madvise(0x7f02c1dfe000, 8368128, MADV_DONTNEED <unfinished ...>
[pid 55281] set_robust_list(0x7f02c2dff920, 24 <unfinished ...>
[pid 55280] <... madvise resumed>)      = 0
[pid 55281] <... set_robust_list resumed>) = 0
[pid 55280] exit(0 <unfinished ...>
[pid 55281] rt_sigprocmask(SIG_SETMASK, [],  <unfinished ...>
[pid 55280] <... exit resumed>)         = ?
[pid 55281] <... rt_sigprocmask resumed>NULL, 8) = 0
[pid 55277] <... futex resumed>)        = 0
[pid 55280] +++ exited with 0 +++
[pid 55277] futex(0x7f02c2dff910, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 55281, NULL,
FUTEX_BITSET_MATCH_ANY <unfinished ...>
[pid 55281] rt_sigprocmask(SIG_BLOCK, ~[RT_1], NULL, 8) = 0
[pid 55281] madvise(0x7f02c25ff000, 8368128, MADV_DONTNEED) = 0
[pid 55281] exit(0)                     = ?
[pid 55277] <... futex resumed>)        = 0
[pid 55281] +++ exited with 0 +++
clock_gettime(CLOCK_PROCESS_CPUTIME_ID, {tv_sec=0, tv_nsec=3854666}) = 0
write(1, "Sorted array: 89 443 4046 4095 \n", 32Sorted array: 89 443 4046 4095
) = 32
write(1, "Sorted for 0.581000ms\n", 22Sorted for 0.581000ms
) = 22
exit_group(0)                           = ?
+++ exited with 0 +++
```

Наглядно видим, что при запуске программы в 4-х потоках, родительский процесс (pid 55277) порождает 4 потока (pid 55278, pid 55279, pid 55280, pid 55281).

Вывод

Главная проблема, с которой я столкнулся при выполнении этой лабораторной работы, была рекурсией, а именно рекурсивное создание потоков. Вся проблема состояла (и возможно даже состоит) в неполном понимании того, как работать с потоками при рекурсии, из-за чего приходилось много раз менять идею реализации сортировки. В ходе лабораторной работы я научился работать с семафорами и смог успешно их применить. Если говорить про потоки и их эффективность, то в моем случае многопоточность не улучшила производительность при

адекватном количестве входных данных (до 4096 элементов последовательности), а даже наоборот, ухудшило. Это можно наглядно увидеть в табличке ниже:

	16 элементов	32 элемента	1024 элемента
1 поток	0.012ms	0.031ms	0.326ms
2 потока	0.588ms	0.572ms	1.204ms
4 потока	0.687ms	0.734ms	2.467ms
16 потоков	0.949ms	0.964ms	7.514ms

Как по мне это происходит как раз-таки из-за рекурсии, так как хоть и создание потока дешевле, чем создание процесса, но все равно оно не бесплатное. Когда мы рекурсивно создаем потоки, хоть и операции у нас выполняются параллельно, но мы тратим время и ресурсы на создание потоков, и за это время, обычная рекурсия обгоняет, даже несмотря на абсолютное не параллельное выполнение алгоритма. В ходе опросов разбирающихся в теме людей, оказалось что это абсолютно нормальное явление.