

**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»**

Выполнил: Д. А. Алгиничев  
Группа: М8О-208БВ-24  
Преподаватель: Е. С. Миронов  
Дата сдачи: 20.10.2025

Москва, 2025

## **Условие**

Разработать программу параллельной сортировки массива методом QuickSort с использованием многопоточности. Программа должна:

- Принимать от пользователя максимальное количество потоков и размер массива
- Позволять вводить элементы массива
- Выполнять параллельную сортировку с использованием заданного количества потоков

## **Цель работы**

Отсортировать массив целых чисел при помощи параллельного алгоритма быстрой сортировки.

## **Задание**

Составить программу на языке Си, обрабатывающую данные в многопоточном режиме. При обработке 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. Так же необходимо уметь продемонстрировать количество потоков, используемое вашей программой с помощью стандартных средств операционной системы. В отчете привести исследование зависимости ускорения и эффективности алгоритма от входных данных и количества потоков. Получившиеся результаты необходимо объяснить.

## **Вариант**

2

## **Метод решения**

Для реализации параллельной сортировки используется модифицированный алгоритм QuickSort:

- Массив разбивается на части по количеству потоков
- Каждая часть сортируется в отдельном потоке
- После завершения всех потоков выполняется слияние отсортированных частей

Ключевые компоненты программы:

- `QuickSort.hpp` - заголовочный файл с объявлением класса `ParallelQuickSort`
- `main.cpp` - точка входа, взаимодействие с пользователем
- `quicksort.cpp` - реализация параллельного алгоритма сортировки
- `Thread.hpp/cpp` - обертка для работы с потоками POSIX

## **Описание программы**

### **Архитектура программы**

Программа использует объектно-ориентированный подход. Основной класс `ParallelQuickSort` инкапсулирует всю логику параллельной сортировки:

- Разделение массива на части для параллельной обработки
- Создание и управление потоками
- Реализация алгоритма QuickSort
- Слияние отсортированных частей

## **Основные функции**

### **Класс ParallelQuickSort:**

- `partition()` - разделение массива относительно опорного элемента
- `quicksort()` - рекурсивная реализация алгоритма QuickSort
- `quicksort_thread()` - функция для выполнения в потоке
- `sort()` - основной метод, организующий параллельную сортировку

### **Класс Thread:**

- `create()` - создание потока
- `join()` - ожидание завершения потока

## **Результаты тестирования**

### **Методика тестирования**

Программа была протестирована на массивах трех размеров: 1 миллион, 5 миллионов и 10 миллионов элементов. Для каждого размера массива измерялось время выполнения при различном количестве потоков (от 1 до 64).

## **Результаты для массива 1 млн элементов**

- Максимальное ускорение:  $\approx 9$  раз
- Оптимальное количество потоков: 4-8
- Эффективность при 4 потоках:  $\approx 50\%$
- Насыщение производительности: после 16 потоков

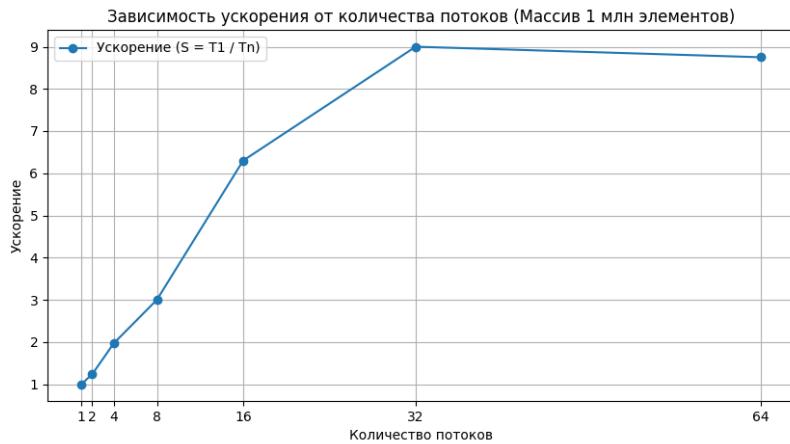


Рис. 1: График ускорения (массив 1 млн элементов)

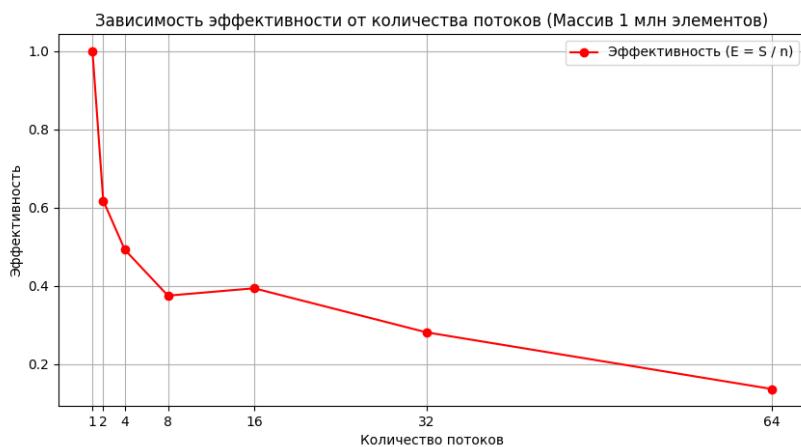


Рис. 2: График эффективности (массив 1 млн элементов)

## Результаты для массива 5 млн элементов

- Максимальное ускорение:  $\approx 10.3$  раз
- Оптимальное количество потоков: 8-16
- Эффективность при 8 потоках:  $\approx 50\%$
- Насыщение производительности: после 32 потоков

## Результаты для массива 10 млн элементов

- Максимальное ускорение:  $\approx 14$  раз
- Оптимальное количество потоков: 16-32
- Эффективность при 16 потоках:  $\approx 50\%$

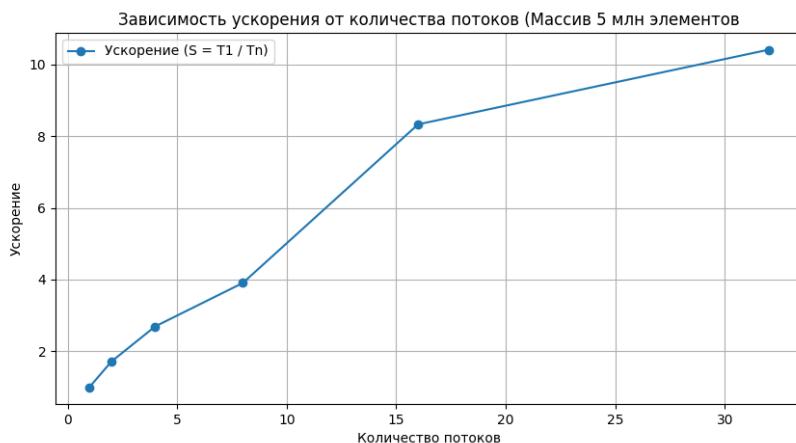


Рис. 3: График ускорения (массив 5 млн элементов)

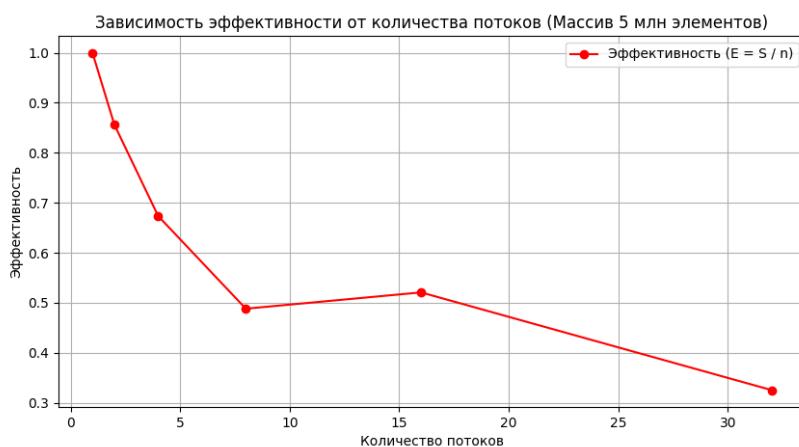


Рис. 4: График эффективности (массив 5 млн элементов)

## Сравнительный анализ

- Зависимость ускорения от размера массива:
  - 1 млн элементов: ускорение до 9 раз
  - 5 млн элементов: ускорение до 10.3 раз
  - 10 млн элементов: ускорение до 14 раз
- Оптимальное количество потоков:
  - 1 млн: 4-8 потоков
  - 5 млн: 8-16 потоков
  - 10 млн: 16-32 потока
- Эффективность использования ресурсов:
  - Меньшие массивы достигают пиковой эффективности при меньшем количестве потоков

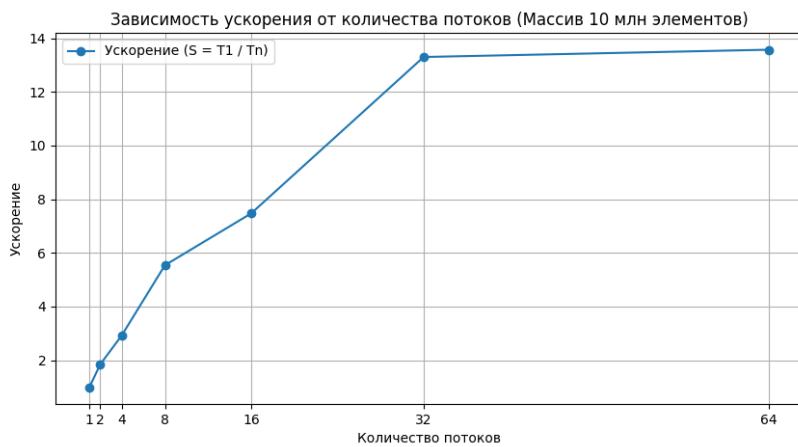


Рис. 5: График ускорения (массив 10 млн элементов)

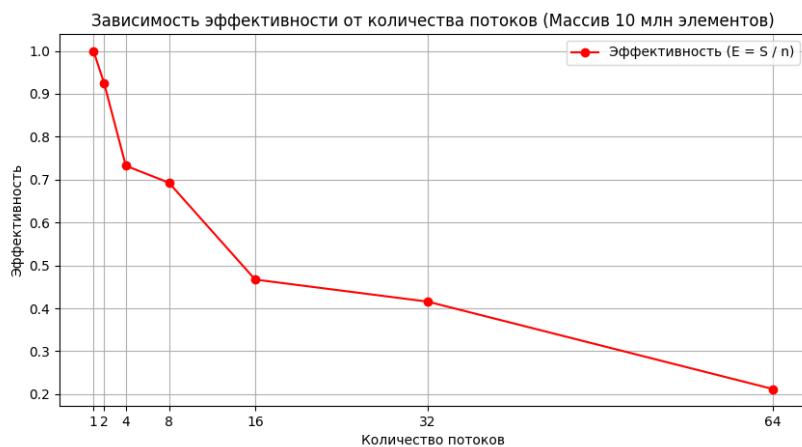


Рис. 6: График эффективности (массив 10 млн элементов)

- Большие массивы позволяют эффективно использовать больше потоков

## Выводы

- Программа демонстрирует хорошую масштабируемость, особенно для больших массивов данных
- Наблюдается четкая зависимость: чем больше размер массива, тем больше потоков можно эффективно использовать
- Максимальное ускорение достигает 14 раз для массива из 10 миллионов элементов
- Для практического применения рекомендуется выбирать количество потоков в зависимости от размера данных

## **Вывод**

### **Приобретенные знания и навыки**

В ходе выполнения лабораторной работы я приобрел ценные практические навыки в области многопоточного программирования. Научился:

- Реализовывать параллельные версии классических алгоритмов, в частности модифицировать алгоритм QuickSort для работы в многопоточной среде
- Работать с POSIX threads (pthreads) для создания и управления потоками в C++
- Организовывать синхронизацию между потоками и корректно завершать их работу
- Разрабатывать эффективные стратегии разделения данных между потоками
- Анализировать производительность параллельных алгоритмов с помощью метрик ускорения и эффективности

### **Личные впечатления и наблюдения**

Работа над параллельной сортировкой оказалась чрезвычайно познавательной. Особенно впечатлило:

- **Простота концепции, сложность реализации:** Идея разделения работы между потоками кажется простой, но на практике возникает множество нюансов синхронизации и балансировки нагрузки
- **Нелинейность ускорения:** Ожидал линейного роста производительности, но реальность показала, что после определенного предела добавление потоков дает затухание ускорение вплоть до его полной остановки
- **Важность размера данных:** Осознал, что эффективность многопоточности сильно зависит от объема обрабатываемых данных - для маленьких массивов однопоточная версия часто оказывается быстрее

### **Преимущества многопоточного подхода**

На личный взгляд, основные преимущества многопоточности:

- **Существенное ускорение:** Для больших объемов данных удалось достичь ускорения в 14 раз, что значительно экономит время вычислений
- **Эффективное использование ресурсов:** Современные процессоры имеют множество ядер, и многопоточность позволяет задействовать их все одновременно
- **Отзывчивость интерфейса:** В реальных приложениях многопоточность позволяет выполнять тяжелые вычисления в фоне без блокировки пользовательского интерфейса

## **Трудности и ограничения**

Однако выявились и определенные сложности:

- **Сложность отладки:** Многопоточные программы значительно сложнее отлаживать из-за недетерминированного поведения
- **Накладные расходы:** Создание потоков, синхронизация и слияние результатов требуют дополнительных ресурсов
- **Ограничения Амдала:** Даже идеально распараллеленный алгоритм имеет последовательные участки, ограничивающие максимальное ускорение
- **Проблемы с памятью:** Неправильная работа с разделяемой памятью может приводить к трудноуловимым ошибкам

## **Практическая ценность**

Полученный опыт крайне ценен для будущей профессиональной деятельности. Многопоточное программирование становится стандартом в разработке производительных приложений, и понимание его принципов необходимо современному программисту. Особенно полезными оказались знания о:

- Выборе оптимального количества потоков для конкретной задачи
- Методах анализа эффективности параллельных алгоритмов
- Техниках избежания типичных ошибок многопоточности
- Балансировке между производительностью и сложностью кода

## **Заключение**

Лабораторная работа не только позволила освоить технические аспекты многопоточного программирования, но и сформировала понимание того, когда и как следует применять этот мощный инструмент. Многопоточность - это не серебряная пуля, а точный инструмент, который при грамотном использовании позволяет создавать высокопроизводительные и эффективные приложения.

Полученные знания буду применять в будущих проектах, где требуется обработка больших объемов данных или высокая отзывчивость системы.

## Листинги программ

Листинг 1: main.cpp - точка входа программы

```
1 // src/main.cpp
2 #include "QuickSort.hpp"
3 #include <iostream>
4 #include <vector>
5 #include <limits>
6 #include <cstdlib>
7 #include <chrono>
8 #include <random>
9
10 void print_usage(const char* program_name) {
11     std::cout << "Usage: " << program_name << " -t <number>" << std::endl;
12     std::cout << " -t <number> Maximum number of threads to use (positive integer)" <<
13         std::endl;
14 }
15 void generate_random_array(std::vector<int>& arr, int size) {
16     std::random_device rd;
17     std::mt19937 gen(rd());
18     std::uniform_int_distribution<int> dis(1, 1000000);
19
20     arr.resize(size);
21     for (int i = 0; i < size; ++i) {
22         arr[i] = dis(gen);
23     }
24 }
25
26 int main(int argc, char* argv[]) {
27     int max_threads = -1;
28
29     if (argc == 1) {
30         print_usage(argv[0]);
31         return 1;
32     }
33
34     for (int i = 1; i < argc; ++i) {
35         std::string arg = argv[i];
36         if (arg == "-t") {
37             if (i + 1 < argc) {
38                 try {
39                     max_threads = std::stoi(argv[i + 1]);
40                     if (max_threads <= 0) {
41                         std::cerr << "Error: max-threads must be a positive number" <<
42                             std::endl;
43                         return 1;
44                     }
45                 } catch (const std::exception& e) {
46                     std::cerr << "Error: Invalid number for max-threads" << std::endl;
47                     return 1;
48                 }
49             } else {
50                 std::cerr << "Error: -t requires a value" << std::endl;
51                 print_usage(argv[0]);
52                 return 1;
53             }
54         }
55     }
56 }
```

```

53     }
54 } else if (arg == "--help" || arg == "-h") {
55     print_usage(argv[0]);
56     return 0;
57 } else {
58     std::cerr << "Error: Unknown argument '" << arg << "'" << std::endl;
59     print_usage(argv[0]);
60     return 1;
61 }
62 }
63
64 if (max_threads == -1) {
65     std::cerr << "Error: -t argument is required" << std::endl;
66     print_usage(argv[0]);
67     return 1;
68 }
69
70 int array_size;
71
72 std::cout << "Enter the size of the array: ";
73 while (!(std::cin >> array_size) || array_size <= 0) {
74     std::cout << "Please enter a positive number: ";
75     std::cin.clear();
76     std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
77 }
78
79 std::vector<int> arr;
80
81 std::cout << "Generating random array of size " << array_size << "..." << std::endl;
82 generate_random_array(arr, array_size);
83
84 std::cout << std::endl;
85
86 auto start_time = std::chrono::high_resolution_clock::now();
87
88 ParallelQuickSort sorter(arr, max_threads);
89 sorter.sort();
90
91 auto end_time = std::chrono::high_resolution_clock::now();
92 auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end_time -
93     start_time);
94
95 std::cout << "\n==== Performance Metrics ===" << std::endl;
96 std::cout << "Maximum threads specified: " << max_threads << std::endl;
97 std::cout << "Array size: " << array_size << std::endl;
98 std::cout << "Execution time: " << duration.count() << " microseconds" << std::endl;
99 std::cout << "Execution time: " << duration.count() / 1000.0 << " milliseconds" <<
100    std::endl;
101 std::cout << "Execution time: " << duration.count() / 1000000.0 << " seconds" <<
102    std::endl;
103
104 }
```

Листинг 2: quicksort.cpp - реализация параллельной сортировки

```
1 #include "QuickSort.hpp"
2 #include <climits>
3 #include "Thread.hpp"
4 #include <cmath>
5
6 ParallelQuickSort::ParallelQuickSort(std::vector<int>& vector, int threads, int start,
7     int end)
8     : vector(vector), max_threads(threads > 0 ? threads : 1), low(start), high(end)
9 {
10     if (max_threads > static_cast<int>(vector.size()))
11     {
12         max_threads = vector.size();
13     }
14 }
15 int ParallelQuickSort::partition(int low, int high)
16 {
17     int pivot = vector[high];
18     int i = low - 1;
19     int k = 0;
20     int j = 0;
21     for (j = low; j < high; ++j)
22     {
23         if (vector[j] < pivot)
24         {
25             ++i;
26             k = vector[i];
27             vector[i] = vector[j];
28             vector[j] = k;
29         }
30     }
31
32     k = vector[i + 1];
33     vector[i + 1] = vector[high];
34     vector[high] = k;
35
36     return i + 1;
37 }
38
39 void ParallelQuickSort::quicksort(int low, int high)
40 {
41     if (low < high)
42     {
43         int pivot = partition(low, high);
44         quicksort(low, pivot - 1);
45         quicksort(pivot + 1, high);
46     }
47 }
48
49 void * ParallelQuickSort::quicksort_thread(void* arg)
50 {
51     ParallelQuickSort* sorter = static_cast<ParallelQuickSort*>(arg);
52     sorter->quicksort(sorter->low, sorter->high);
53     delete sorter;
54     return nullptr;
55 }
```

```

56
57 void ParallelQuickSort::sort()
58 {
59     int n = vector.size();
60     size_t i = 0;
61     int pos = 0;
62
63     std::vector<int> bounds(max_threads + 1);
64     bounds[0] = 0;
65     bounds[max_threads] = n;
66     int part_size = n / max_threads;
67     int other = n % max_threads;
68     for (i = 1; i < max_threads; ++i)
69     {
70         bounds[i] = bounds[i - 1] + part_size + (i <= other ? 1 : 0);
71     }
72
73     std::vector<Thread> threads;
74     threads.reserve(max_threads - 1);
75     for (i = 0; i < max_threads; ++i)
76     {
77         if (bounds[i] >= bounds[i + 1]) continue;
78         ParallelQuickSort* sorter = new ParallelQuickSort(vector, max_threads, bounds[i],
79             bounds[i + 1] - 1);
80         Thread th;
81
82         if (th.create(quicksort_thread, sorter) != 0)
83         {
84             delete sorter;
85             quicksort(bounds[i], bounds[i + 1] - 1);
86         }
87         else
88         {
89             threads.push_back(th);
90         }
91     }
92     if (bounds[max_threads - 1] < bounds[max_threads])
93     {
94         quicksort(bounds[max_threads - 1], bounds[max_threads] - 1);
95     }
96
97     for (auto& th: threads)
98     {
99         th.join(nullptr);
100    }
101
102    std::vector<int> temp(n);
103    std::vector<int> indices(max_threads);
104
105    for (i = 0; i < max_threads; ++i)
106    {
107        indices[i] = bounds[i];
108    }
109
110    for (pos = 0; pos < n; ++pos)
111    {
112        int min_val = INT_MAX;

```

```

113     int min_index = -1;
114
115     for (i = 0; i < max_threads; ++i)
116     {
117         if (indices[i] < bounds[i + 1] && vector[indices[i]] < min_val)
118         {
119             min_val = vector[indices[i]];
120             min_index = i;
121         }
122     }
123
124     if (min_index != -1)
125     {
126         temp[pos] = min_val;
127         indices[min_index]++;
128     }
129 }
130
131 for (i = 0; i < n; ++i)
132 {
133     vector[i] = temp[i];
134 }
135 }
```

Листинг 3: Thread.cpp - работа с потоками

```

1 #include "Thread.hpp"
2
3 Thread::Thread(): created(false) {}
4
5 Thread::~Thread() {}
6
7
8 int Thread::create(void* (*start_routine)(void*), void* arg)
9 {
10    int result = pthread_create(&tid, nullptr, start_routine, arg);
11    if (result == 0)
12    {
13        created = true;
14    }
15
16    return result;
17 }
18
19 int Thread::join(void** retval)
20 {
21    if (!created)
22    {
23        return -1;
24    }
25
26    return pthread_join(tid, retval);
27 }
```

## Системные вызовы

```
execve("./quicksort", ["../quicksort", "-t", "16"], 0x7ffefb84d840 /* 36 vars */)
= 0
```





```
rt_sigprocmask(SIG_BLOCK, ~[], [], 8) = 0
clone3({flags=CLONE_VM|CLONE_FS|CLONE_FILES|CLONE_SIGHAND|CLONE_THREAD|CLONE_SYSVSEM|0, stack_size=0}, 88) = 124453
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e806d3d8000
mprotect(0x7e806d3d9000, 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|0, stack_size=0}, 88) = 124454
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e806cbd7000
mprotect(0x7e806cbd8000, 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|0, stack_size=0}, 88) = 124455
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e806c3d6000
mprotect(0x7e806c3d7000, 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|0, stack_size=0}, 88) = 124456
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e806bb65000
mprotect(0x7e806bb66000, 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|0, stack_size=0}, 88) = 124457
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e806b3d4000
mprotect(0x7e806b3d5000, 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|0, stack_size=0}, 88) = 124458
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e806abd3000
mprotect(0x7e806abd4000, 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|0, stack_size=0}, 88) = 124459
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e806a3d2000
mprotect(0x7e806a3d3000, 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|0, stack_size=0}, 88) = 124460
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e8069bd1000
mprotect(0x7e8069bd2000, 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_SETSIGP|CLONE_PARENT_SETSIGP|CLONE_CHILD_CLEARTID|CLONE_CHILD_SETSIGP}, {parent_tid=[124461]}, 88) = 124461
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e80693d0000
mprotect(0x7e80693d1000, 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_SETSIGP|CLONE_PARENT_SETSIGP|CLONE_CHILD_CLEARTID|CLONE_CHILD_SETSIGP}, {parent_tid=[124462]}, 88) = 124462
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e8068bcf000
mprotect(0x7e8068bd0000, 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_SETSIGP|CLONE_PARENT_SETSIGP|CLONE_CHILD_CLEARTID|CLONE_CHILD_SETSIGP}, {parent_tid=[124463]}, 88) = 124463
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e80683ce000
mprotect(0x7e80683cf000, 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_SETSIGP|CLONE_PARENT_SETSIGP|CLONE_CHILD_CLEARTID|CLONE_CHILD_SETSIGP}, {parent_tid=[124464]}, 88) = 124464
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e8067bcd000
mprotect(0x7e8067bce000, 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_SETSIGP|CLONE_PARENT_SETSIGP|CLONE_CHILD_CLEARTID|CLONE_CHILD_SETSIGP}, {parent_tid=[124465]}, 88) = 124465
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e80673cc000
mprotect(0x7e80673cd000, 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_SETSIGP|CLONE_PARENT_SETSIGP|CLONE_CHILD_CLEARTID|CLONE_CHILD_SETSIGP}, {parent_tid=[124466]}, 88) = 124466
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e8066bcc000
mprotect(0x7e8066bcc000, 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_SETSIGP|CLONE_PARENT_SETSIGP|CLONE_CHILD_CLEARTID|CLONE_CHILD_SETSIGP}, {parent_tid=[124467]}, 88) = 124467
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
mmap(NULL, 8392704, PROT_NONE, MAP_PRIVATE|MAP_ANONYMOUS|MAP_STACK, -1, 0) = 0x7e80663ca000
mprotect(0x7e80663cb000, 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_SETSIGP|CLONE_PARENT_SETSIGP|CLONE_CHILD_CLEARTID|CLONE_CHILD_SETSIGP}, {parent_tid=[124468]}, 88) = 124468
rt_sigprocmask(SIG_SETMASK, [], NULL, 8) = 0
futex(0x7e806e3d9990, FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME, 124453, NULL, FUTEX_BITSET|FUTEX_CLOCK_REALTIME) = 0
```

```
futex(0x7e806cbd6990,FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME,124456,NULL,FUTEX_BITSET_1)
= 0
futex(0x7e806c3d5990,FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME,124457,NULL,FUTEX_BITSET_1)
= 0
munmap(0x7e806dbd9000,8392704)          = 0
munmap(0x7e806d3d8000,8392704)          = 0
munmap(0x7e806cbd7000,8392704)          = 0
munmap(0x7e806c3d6000,8392704)          = 0
munmap(0x7e806bbd5000,8392704)          = 0
munmap(0x7e806b3d4000,8392704)          = 0
munmap(0x7e806abd3000,8392704)          = 0
munmap(0x7e806a3d2000,8392704)          = 0
munmap(0x7e8069bd1000,8392704)          = 0
munmap(0x7e80693d0000,8392704)          = 0
munmap(0x7e8068bcf000,8392704)          = 0
futex(0x7e8066bca990,FUTEX_WAIT_BITSET|FUTEX_CLOCK_REALTIME,124468,NULL,FUTEX_BITSET_1)
= 0
munmap(0x7e80683ce000,8392704)          = 0
mmap(NULL,40001536,PROT_READ|PROT_WRITE,MAP_PRIVATE|MAP_ANONYMOUS,-1,0) = 0x7e806bdb4000
munmap(0x7e806bdb4000,40001536)          = 0
write(1,"\\n",1)                         = 1
write(1,"==== Performance Metrics ===\\n",28) = 28
write(1,"Maximum threads specified: 16\\n",30) = 30
write(1,"Array size: 10000000\\n",21)     = 21
write(1,"Execution time: 686039 microseco",36) = 36
write(1,"Execution time: 686.039 millisec",37) = 37
write(1,"Execution time: 0.686039 seconds",33) = 33
write(1,"\\n",1)                         = 1
write(1,"First 10 sorted elements: 1 1 1 ...",47) = 47
munmap(0x7e806e3da000,40001536)          = 0
lseek(0,-1,SEEK_CUR)                    = -1 ESPIPE (Illegal seek)
exit_group(0)                           = ?
+++ exited with 0 +++
```