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-209Б-23

Студент: Борисов Д.С.

Преподаватель: Миронов Е.С.

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата: 20.11.24

Москва, 2024

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc183047267)

[Общий метод и алгоритм решения 3](#_Toc183047268)

[Замеры эффективности 4](#_Toc183047269)

[Вывод 5](#_Toc183047270)

[Код программы 5](#_Toc183047271)

Постановка задачи

**Вариант 2.**

Составить программу на языке Си, обрабатывающую данные в многопоточном

режиме. При обработки 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.

Отсортировать массив целых чисел при помощи параллельного алгоритма быстрой сортировки.

Общий метод и алгоритм решения

Использованные системные вызовы:

* pthread\_create(**pthread\_t** \***thread**, **const** **pthread\_attr\_t** \*attr, **void** \*(\*start\_routine) (**void** \*), **void** \*arg); – создает новый поток, возвращает 0 при успехе
* pthread\_join(**pthread\_t** **thread**, **void** \*\*retval); – ожидает завершения указанного потока. Блокирует вызывающий поток до завершения целевого потока
* pthread\_mutex\_lock(**pthread\_mutex\_t** \*mutex); – блокирует мьютекс. Если мьютекс уже заблокирован, поток блокируется до освобождения
* pthread\_mutex\_unlock(**pthread\_mutex\_t** \*mutex); – разблокирует мьютекс. Позволяет другим потокам захватить мьютекс

Ключевые особенности:

1. Основан на алгоритме быстрой сортировки Хоара, использует стратегию "разделяй и властвуй"

2. Применяет параллельную обработку подмассивов

3. Эффективен на многоядерных системах

Работа алгоритма:

1. Инициализация:

- Получение максимального количества потоков из аргументов командной строки

- Инициализация массива случайными числами

- Настройка механизмов синхронизации

2. Процесс сортировки

Разделение массива на части:

- Выбор опорного элемента

- Перемещение элементов относительно опорного

- Получение точки разделения

Создание потоков:

- Проверка доступных слотов для новых потоков

- При наличии слотов - создание двух новых потоков

- При отсутствии слотов - рекурсивная сортировка в текущем потоке

Контроль потоков:

- Отслеживание количества активных потоков через мьютекс

- Ограничение общего числа потоков через семафор

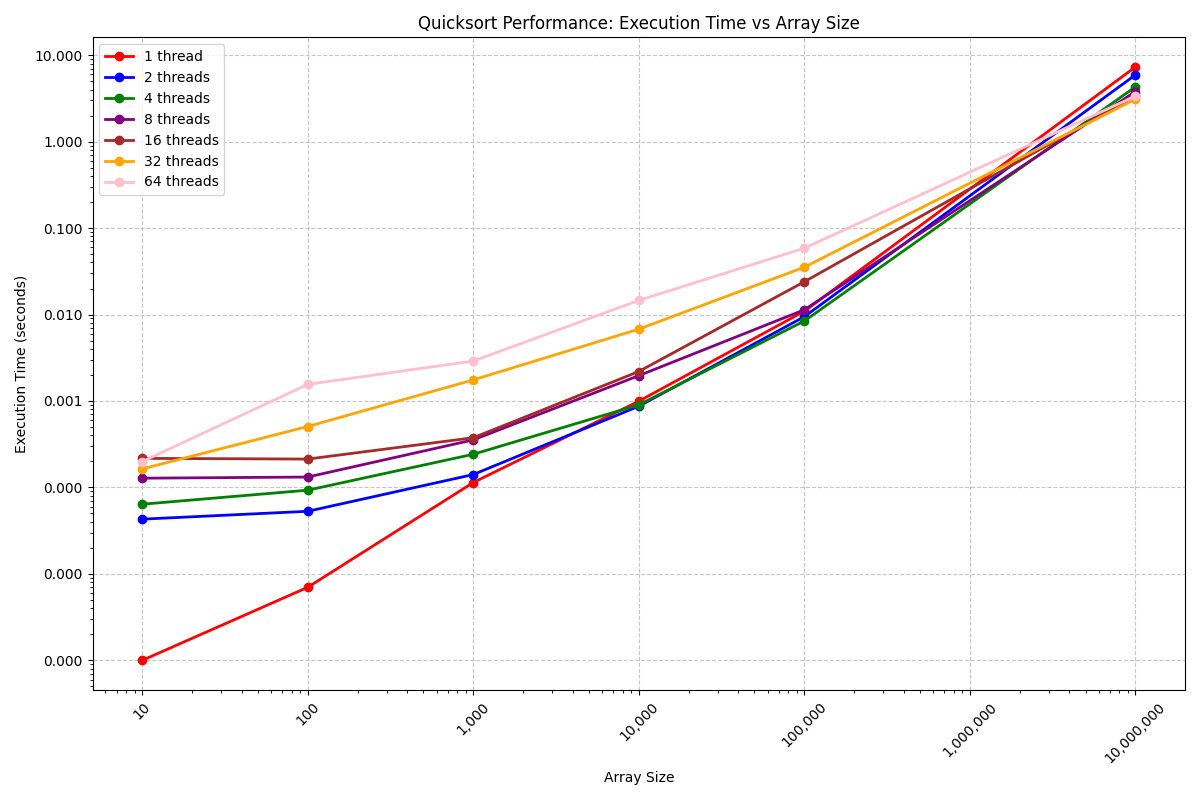
- Ожидание завершения дочерних потоков

# Замеры эффективности

Замеры проводились для 6 разных длин массивов. Количество потоков было от 1 до 64. Так как массив заполняется случайными значениями, то для каждого количества потоков проводилось 5 замеров и бралось среднее значение времени.

На моем процессоре доступно 8 физических потоков

График со всеми замерами. Оси с логарифмическими шкалами



**Анализ алгоритма**

График зависимости времени обработки данных от их размера наглядно демонстрирует зависимость. Очевидно, что при маленьком объеме данных сортировка с большим количеством потоком будет выполняться значительно дольше по ряду причин: возрастают накладные расходы на планирование потоков, увеличивается конкуренция за общие ресурсы (память, шина данных), растет сложность синхронизации между потоками, затрачивается время на контекстное переключение, деградация кеша и тд. Однако на большом объеме данным и даже огромном, ситуация становится абсолютно зеркальной. Сортировка с большим количеством ядер значительно превосходит сортировку с меньшим количеством ядер из за того, что больше времени стало затрачиваться на обработку подпоследовательности массива, а не на создание и переключение потоков.

# Вывод

Я считаю, данная сортировка является одной из самых эффективных, потому что она предсказуемо отображает время выполнения алгоритма от объема обрабатываемых данных. Данные алгоритм показывает, что время прямо пропорционально количеству ядер, требуемых для обработки набора данных. Сама лабораторная работа отлично демонстрирует работу с примитивами синхронизации, управление потоками и применение многопоточного программирования в практических задачах.

# Код программы

<https://github.com/DenisBorisov-lab/os-workshop/tree/main/lab2>

**Протокол работы программы**

SYSCALL(args) = return

Array before sorting:

1807 249 -4927 -1342 3930 -3728 2544 -4122 2923 2709 -560 3165 -508 -1958 2987 -2497 -2673 -3271 3840 -2388 -697 -1831 2709 2157 4560 -4067 -1901 -4722 -3184 335 4097 2826 -1488 4267 -1190 2633 -4021 4149 1579 3821 -3033 -4328 -3607 4336 485 -3255 228 -909 -4806 1357 1 -3847 1708 2944 668 -3510 3124 -2804 4530 -4097 2722 -334 3549 3024 2801 1853 -4023 2408 3228 -67 -4702 3981 3635 3013 -1135 4814 4063 -464 4425 -3331 -885 -4906 629 1501 1517 -805 -4895 -4596 4451 -702 -2812 -3877 4505 1882 1752 -3434 1716 -4663 -562 -1856

Sorting array of 100 elements with max 4 threads...

Sorting took 0.000117 seconds

Array after sorting:

-4927 -4906 -4895 -4806 -4722 -4702 -4663 -4596 -4328 -4122 -4097 -4067 -4023 -4021 -3877 -3847 -3728 -3607 -3510 -3434 -3331 -3271 -3255 -3184 -3033 -2812 -2804 -2673 -2497 -2388 -1958 -1901 -1856 -1831 -1488 -1342 -1190 -1135 -909 -885 -805 -702 -697 -562 -560 -508 -464 -334 -67 1 228 249 335 485 629 668 1357 1501 1517 1579 1708 1716 1752 1807 1853 1882 2157 2408 2544 2633 2709 2709 2722 2801 2826 2923 2944 2987 3013 3024 3124 3165 3228 3549 3635 3821 3840 3930 3981 4063 4097 4149 4267 4336 4425 4451 4505 4530 4560 4814

munmap(0x1030C4000, 0x8C000) = 0 0

munmap(0x103150000, 0x8000) = 0 0

munmap(0x103158000, 0x4000) = 0 0

munmap(0x10315C000, 0x4000) = 0 0

munmap(0x103160000, 0x50000) = 0 0

crossarch\_trap(0x0, 0x0, 0x0) = -1 Err#45

open(".\0", 0x100000, 0x0) = 3 0

fcntl(0x3, 0x32, 0x16D007298) = 0 0

close(0x3) = 0 0

fsgetpath(0x16D0072A8, 0x400, 0x16D007288) = 47 0

fsgetpath(0x16D0072B8, 0x400, 0x16D007298) = 14 0

csrctl(0x0, 0x16D0076BC, 0x4) = -1 Err#1

\_\_mac\_syscall(0x1841D7F9F, 0x2, 0x16D007600) = 0 0

csrctl(0x0, 0x16D0076AC, 0x4) = -1 Err#1

\_\_mac\_syscall(0x1841D4DD2, 0x5A, 0x16D007640) = 0 0

sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16D006BC8, 0x16D006BC0, 0x1841D6A23, 0xD) = 0 0

sysctl([CTL\_KERN, 155, 0, 0, 0, 0] (2), 0x16D006C78, 0x16D006C70, 0x0, 0x0) = 0 0

open("/\0", 0x20100000, 0x0) = 3 0

openat(0x3, "System/Cryptexes/OS\0", 0x100000, 0x0) = 4 0

dup(0x4, 0x0, 0x0) = 5 0

fstatat64(0x4, 0x16D006751, 0x16D0066C0) = 0 0

openat(0x4, "System/Library/dyld/\0", 0x100000, 0x0) = 6 0

fcntl(0x6, 0x32, 0x16D006750) = 0 0

dup(0x6, 0x0, 0x0) = 7 0

dup(0x5, 0x0, 0x0) = 8 0

close(0x3) = 0 0

close(0x5) = 0 0

close(0x4) = 0 0

close(0x6) = 0 0

\_\_mac\_syscall(0x1841D7F9F, 0x2, 0x16D007140) = 0 0

shared\_region\_check\_np(0x16D006D60, 0x0, 0x0) = 0 0

fsgetpath(0x16D0072C0, 0x400, 0x16D007208) = 82 0

fcntl(0x8, 0x32, 0x16D0072C0) = 0 0

close(0x8) = 0 0

close(0x7) = 0 0

getfsstat64(0x0, 0x0, 0x2) = 10 0

getfsstat64(0x10320E090, 0x54B0, 0x2) = 10 0

getattrlist("/\0", 0x16D0071F0, 0x16D007160) = 0 0

stat64("/System/Volumes/Preboot/Cryptexes/OS/System/Library/dyld/dyld\_shared\_cache\_arm64e\0", 0x16D007550, 0x0) = 0 0

dtrace: error on enabled probe ID 1690 (ID 845: syscall::stat64:return): invalid address (0x0) in action #11 at DIF offset 12

stat64("/Users/denisborisov/os-workshop/lab2/quicksort\0", 0x16D006A00, 0x0) = 0 0

open("/Users/denisborisov/os-workshop/lab2/quicksort\0", 0x0, 0x0) = 3 0

mmap(0x0, 0xC7B8, 0x1, 0x40002, 0x3, 0x0) = 0x103250000 0

fcntl(0x3, 0x32, 0x16D006B18) = 0 0

close(0x3) = 0 0

munmap(0x103250000, 0xC7B8) = 0 0

stat64("/Users/denisborisov/os-workshop/lab2/quicksort\0", 0x16D006F70, 0x0) = 0 0

stat64("/usr/lib/libSystem.B.dylib\0", 0x16D005EB0, 0x0) = -1 Err#2

stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/libSystem.B.dylib\0", 0x16D005E60, 0x0) = -1 Err#2

stat64("/usr/lib/system/libdispatch.dylib\0", 0x16D003AD0, 0x0) = -1 Err#2

stat64("/System/Volumes/Preboot/Cryptexes/OS/usr/lib/system/libdispatch.dylib\0", 0x16D003A80, 0x0) = -1 Err#2

stat64("/usr/lib/system/libdispatch.dylib\0", 0x16D003AD0, 0x0) = -1 Err#2

open("/dev/dtracehelper\0", 0x2, 0x0) = 3 0

ioctl(0x3, 0x80086804, 0x16D005AA8) = 0 0

close(0x3) = 0 0

open("/Users/denisborisov/os-workshop/lab2/quicksort\0", 0x0, 0x0) = 3 0

\_\_mac\_syscall(0x1841D7F9F, 0x2, 0x16D0051B0) = 0 0

map\_with\_linking\_np(0x16D005050, 0x1, 0x16D005080) = 0 0

close(0x3) = 0 0

mprotect(0x102DFC000, 0x4000, 0x1) = 0 0

shared\_region\_check\_np(0xFFFFFFFFFFFFFFFF, 0x0, 0x0) = 0 0

mprotect(0x10320C000, 0x40000, 0x1) = 0 0

access("/AppleInternal/XBS/.isChrooted\0", 0x0, 0x0) = -1 Err#2

bsdthread\_register(0x1844D9D2C, 0x1844D9D20, 0x4000) = 1073746399 0

getpid(0x0, 0x0, 0x0) = 1420 0

shm\_open(0x184374F51, 0x0, 0x6C2F7273) = 3 0

fstat64(0x3, 0x16D005E50, 0x0) = 0 0

mmap(0x0, 0x4000, 0x1, 0x40001, 0x3, 0x0) = 0x103258000 0

close(0x3) = 0 0

ioctl(0x2, 0x4004667A, 0x16D005EFC) = 0 0

mprotect(0x103264000, 0x4000, 0x0) = 0 0

mprotect(0x103270000, 0x4000, 0x0) = 0 0

mprotect(0x103274000, 0x4000, 0x0) = 0 0

mprotect(0x103280000, 0x4000, 0x0) = 0 0

mprotect(0x103284000, 0x4000, 0x0) = 0 0

mprotect(0x103290000, 0x4000, 0x0) = 0 0

mprotect(0x10325C000, 0xA0, 0x1) = 0 0

mprotect(0x10325C000, 0xA0, 0x3) = 0 0

mprotect(0x10325C000, 0xA0, 0x1) = 0 0

mprotect(0x103294000, 0x4000, 0x1) = 0 0

mprotect(0x103298000, 0xA0, 0x1) = 0 0

mprotect(0x103298000, 0xA0, 0x3) = 0 0

mprotect(0x103298000, 0xA0, 0x1) = 0 0

mprotect(0x10325C000, 0xA0, 0x3) = 0 0

mprotect(0x10325C000, 0xA0, 0x1) = 0 0

mprotect(0x103294000, 0x4000, 0x3) = 0 0

mprotect(0x103294000, 0x4000, 0x1) = 0 0

mprotect(0x10320C000, 0x40000, 0x3) = 0 0

mprotect(0x10320C000, 0x40000, 0x1) = 0 0

objc\_bp\_assist\_cfg\_np(0x184109C00, 0x80000018001C1048, 0x0) = -1 Err#5

issetugid(0x0, 0x0, 0x0) = 0 0

mprotect(0x10320C000, 0x40000, 0x3) = 0 0

getentropy(0x16D005578, 0x20, 0x0) = 0 0

mprotect(0x10320C000, 0x40000, 0x1) = 0 0

mprotect(0x10320C000, 0x40000, 0x3) = 0 0

mprotect(0x10320C000, 0x40000, 0x1) = 0 0

getattrlist("/Users/denisborisov/os-workshop/lab2/quicksort\0", 0x16D005DE0, 0x16D005DF8) = 0 0

access("/Users/denisborisov/os-workshop/lab2\0", 0x4, 0x0) = 0 0

open("/Users/denisborisov/os-workshop/lab2\0", 0x0, 0x0) = 3 0

fstat64(0x3, 0x153E044A0, 0x0) = 0 0

csrctl(0x0, 0x16D00600C, 0x4) = 0 0

fcntl(0x3, 0x32, 0x16D005CC8) = 0 0

close(0x3) = 0 0

open("/Users/denisborisov/os-workshop/lab2/Info.plist\0", 0x0, 0x0) = -1 Err#2

proc\_info(0x2, 0x58C, 0xD) = 64 0

csops\_audittoken(0x58C, 0x10, 0x16D006050) = 0 0

sysctl([unknown, 3, 0, 0, 0, 0] (2), 0x16D0063A8, 0x16D0063A0, 0x187903D3D, 0x15) = 0 0

sysctl([CTL\_KERN, 153, 0, 0, 0, 0] (2), 0x16D006438, 0x16D006430, 0x0, 0x0) = 0 0

csops(0x58C, 0x0, 0x16D0064DC) = 0 0

mprotect(0x10320C000, 0x40000, 0x3) = 0 0

sem\_unlink(0x102DFBEF7, 0x0, 0x0) = -1 Err#2

sem\_open(0x102DFBEF7, 0xA00, 0x1A4) = 3 0

getrlimit(0x1008, 0x16D007478, 0x0) = 0 0

fstat64(0x1, 0x16D007470, 0x0) = 0 0

ioctl(0x1, 0x4004667A, 0x16D0074BC) = 0 0

write\_nocancel(0x1, "Array before sorting:\n\0", 0x16) = 22 0

write\_nocancel(0x1, "1807 249 -4927 -1342 3930 -3728 2544 -4122 2923 2709 -560 3165 -508 -1958 2987 -2497 -2673 -3271 3840 -2388 -697 -1831 2709 2157 4560 -4067 -1901 -4722 -3184 335 4097 2826 -1488 4267 -1190 2633 -4021 4149 1579 3821 -3033 -4328 -3607 4336 485 -3255 228 -909", 0x211) = 529 0

write\_nocancel(0x1, "Sorting array of 100 elements with max 4 threads...\n\0", 0x34) = 52 0

sem\_wait(0x3, 0x0, 0x0) = 0 0

sem\_wait(0x3, 0x0, 0x0) = 0 0

bsdthread\_create(0x102DFB7D4, 0x600001CFC000, 0x16D08F000) = 1829302272 0

bsdthread\_create(0x102DFB7D4, 0x600001CFC020, 0x16D11B000) = 1829875712 0

thread\_selfid(0x0, 0x0, 0x0) = 11868 0

sem\_wait(0x3, 0x0, 0x0) = 0 0

thread\_selfid(0x0, 0x0, 0x0) = 11869 0

sem\_wait(0x3, 0x0, 0x0) = 0 0

\_\_disable\_threadsignal(0x1, 0x0, 0x0) = 0 0

bsdthread\_create(0x102DFB7D4, 0x600001CF0000, 0x16D1A7000) = 1830449152 0

bsdthread\_create(0x102DFB7D4, 0x600001CF0020, 0x16D233000) = 1831022592 0

thread\_selfid(0x0, 0x0, 0x0) = 11871 0

thread\_selfid(0x0, 0x0, 0x0) = 11870 0

\_\_disable\_threadsignal(0x1, 0x0, 0x0) = 0 0

\_\_disable\_threadsignal(0x1, 0x0, 0x0) = 0 0

ulock\_wake(0x1000002, 0x16D1A7034, 0x0) = 0 0

ulock\_wait(0x1020002, 0x16D1A7034, 0xD03) = 0 0

sem\_post(0x3, 0x0, 0x0) = 0 0

sem\_post(0x3, 0x0, 0x0) = 0 0

\_\_disable\_threadsignal(0x1, 0x0, 0x0) = 0 0

ulock\_wake(0x1000002, 0x16D08F034, 0x0) = 0 0

ulock\_wait(0x1020002, 0x16D08F034, 0xB07) = 0 0

sem\_post(0x3, 0x0, 0x0) = 0 0

sem\_post(0x3, 0x0, 0x0) = 0 0

write\_nocancel(0x1, "Sorting took 0.000117 seconds\n\0", 0x1E) = 30 0

write\_nocancel(0x1, "Array after sorting:\n\0", 0x15) = 21 0

write\_nocancel(0x1, "-4927 -4906 -4895 -4806 -4722 -4702 -4663 -4596 -4328 -4122 -4097 -4067 -4023 -4021 -3877 -3847 -3728 -3607 -3510 -3434 -3331 -3271 -3255 -3184 -3033 -2812 -2804 -2673 -2497 -2388 -1958 -1901 -1856 -1831 -1488 -1342 -1190 -1135 -909 -885 -805 -702 -697 -56", 0x211) = 529 0

sem\_close(0x3, 0x0, 0x0) = 0 0

sem\_unlink(0x102DFBEF7, 0x0, 0x0) = 0 0